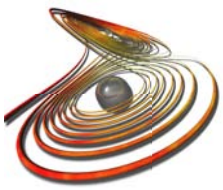
Volker Settgast

Processing Semantically Enriched Content for Interactive 3D Visualizations

A thesis submitted for the degree
of Doctor of Philosophy (PhD)
in Engineering Sciences (Computer Science)
by Dipl.-Inform. Volker Settgast,
Graz University of Technology, Austria

February 2013

Graz University of Technology



Institute of ComputerGraphics and
Knowledge Visualization

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

.....	.....
date	(signature)

Acknowledgements

I would like to gratefully thank my supervisor Prof. Dr. Dieter W. Fellner for his support and his confidence in my work. I would also like to thank all my colleagues of the Institute of Computer Graphics and Knowledge Visualization at Graz University of Technology. Working with highly committed colleagues was very inspiring and had a significant influence on my work. In this context, I would like to express my gratitude to all the co-authors of the scientific articles published within their and my work. I would particularly like to thank Torsten Ullrich for his endless support, constructive criticism and excellent advice for the scientific work and the writing of this thesis.
This work is dedicated to my parents.

Abstract

Interactive 3D graphics has become an essential tool in many fields of application: In manufacturing companies, e.g., new products are planned and tested digitally. The effect of new designs and testing of ergonomic aspects can be done with pure virtual models. Furthermore, the training of procedures on complex machines is shifted to the virtual world. In that way support costs for the usage of the real machine are reduced, and effective forms of training evaluation are possible.
Virtual reality helps to preserve and study cultural heritage: Artifacts can be digitalized and preserved in a digital library making them accessible for a larger group of people. Various forms of analysis can be performed on the digital objects which are hardly possible to perform on the real objects or would destroy them. Using virtual reality environments like large projection walls helps to show virtual scenes in a realistic way. The level of immersion can be further increased by using stereoscopic displays and by adjusting the images to the head position of the observer.
One challenge with virtual reality is the inconsistency in data. Moving 3D content from a useful state, e.g., from a repository of artifacts or from within a planning work flow to an interactive presentation is often realized with degenerative steps of preparation. The productiveness of Powerwalls and CAVEs™ is called into question, because the creation of interactive virtual worlds is a one way road in many cases: Data has to be reduced in order to be manageable by the interactive renderer and to be displayed in real time on various target platforms. The impact of virtual reality can be improved by bringing back results from the virtual environment to a useful state or even better: never leave that state.
With the help of semantic data throughout the whole process, it is possible to speed up the preparation steps and to keep important information within the virtual 3D scene. The integrated support for semantic data enhances the virtual experience and opens new ways of presentation. At the same time the goal becomes feasible to bring back data from the presentation for example in a CAVE™ to the working process. Especially in the field of cultural heritage it is essential to store semantic data with the 3D artifacts in a sustainable way.
Within this thesis new ways of handling semantic data in interactive 3D visualizations are presented. The whole process of 3D data creation is demonstrated with regard to semantic sustainability. The basic terms, definitions and available standards for semantic markups are described. Additionally, a method is given to generate semantics of higher order automatically. An important aspect is the linking of semantic information with 3D data. The thesis gives two suggestions on how to store and publish the valuable combination of 3D content and semantic markup in a sustainable way.
Different environments for virtual reality are compared and their special needs are pointed out. Primarily the DAVE in Graz is presented in detail, and novel ways of user interactions in such

immersive environments are proposed. Finally applications in the field of cultural heritage, security and mobility are presented.
The presented symbiosis of 3D content and semantic information is an important contribution for improving the usage of virtual environments in various fields of applications.

Contents

Chapter 1
Introduction

Interactive three-dimensional (3D) graphics has become an essential tool in many fields of application. The majority of manufacturing companies have switched to a digital work flow for the planning of new products. It is an advantage over competing companies to reduce the development time of a product and to release new products early. The effect of new designs and testing of ergonomic aspects can be done with pure virtual models. It is not sufficient to show renderings of the product, interactive experience and dynamic modification is demanded. Also the training of procedures on complex machines for example planes is shifted to the virtual world. In that way support costs for the usage of the real machine are reduced and effective forms of training evaluation are possible.

Virtual reality helps to preserve and study cultural heritage: Artifacts can be digitalized and conserved in a digital library making them accessible for a larger group of people. Archeologists world wide are able to query museums for artifacts. They get accurate 3D models to examine without leaving the desk. Digital artifacts can be analyzed in novel ways without damaging the real objects.

Using virtual reality environments like large projection walls helps to show virtual scenes in a realistic way. The level of immersion can be further increased by using stereoscopic displays and by adjusting the images to the head position of the observer.

1.1 Motivation

Novel virtual reality environments offer new ways to present interactive graphics. The CAVE™ environment is able to show virtual 3D scene with a high level of "immersiveness". That means the user feels himself being a part of the artificial world. One challenge is to control a distributed rendering system, multiple render machines, to generate large areas of high resolution displays.

Non-standard environments also demand new forms of interaction methods. Users want to travel through the virtual world and also rearrange and modify the scene. Such interactions should be intuitive to operate because then a larger group of users can be addressed. Furthermore the experience should be comparable with reality. For example the estimation of sizes and traveled distances in a simulated world has to be transferable to reality in order to utilize results from the simulation environment.

Another challenge with virtual reality is the inconsistency in data. Moving 3D content from a useful state e.g. from a repository of artifacts or from within a planning work flow to an interactive presentation is often done with degenerative steps of preparation. The productiveness of Powerwalls and CAVEs™ is called into question because the creation of interactive virtual

worlds is a one way road in many cases. Data has to be reduced in order to be manageable by the interactive renderer and to be displayed in real time on various target platforms. The impact of virtual reality can be improved by bringing back results from the virtual environment to a useful state or even better: never leave that state.

If possible, the semantic enrichment should start right at the beginning with the creation of the geometry. Although it is possible to add semantic information on existing 3D scenes, the effort is much higher. With the help of semantic data throughout the whole process of creation and presentation, it is possible to keep important information within the virtual 3D scene. The semantic data helps to speed up the preparation steps for interactive visualization: With machine-readable knowledge of shape an automatic process is able to select parts of the geometry according to the application. At the same time the integrated support for semantic data enhances the virtual experience and opens new ways of presentation. Finally the goal becomes feasible to bring back the data from the presentation for example in a CAVE™ to the working process.

The union of shape and it's semantic meaning is advancing in the field of virtual engineering. But a general, standardized solution for the large variety of 3D representations is certainly not yet in sight. Especially in the field of cultural heritage it is strongly required to store semantic data with the 3D artifacts in a sustainable way.

1.2 Structure

Within this thesis new ways of handling semantic data in interactive 3D visualizations are presented. The thesis is structured into four parts. At first the whole process of 3D data creation is described with regard to semantic sustainability. Different types of representations and file formats form the basis for working with 3D content. Describing all of them, even without going into detail, would go beyond the scope of this thesis. The most relevant data representations and file formats for this work are presented. Various methods of creating 3D content are discussed to further emphasize the diversity of 3D content.

The second part describes semantic information in combination with 3D content. The basics for semantic markups and naming conventions in this work are described. Different criteria for the classification of semantic information are discussed. For sustainability it is important to examine standards and file formats for semantic data. An automatic process for the creation of semantic knowledge is presented. The linking of semantic information with 3D data is proposed and how to publish the valuable combination of 3D content and semantic markup in a sustainable way.

In the third part the interactive presentation of semantically enriched 3D content is described. Graphics systems and interfaces are presented. Different environments for virtual reality are compared, and their special needs are pointed out. Especially the DAVE in Graz is explained in detail and various novel ways of user interactions in immersive environments are described. The process of preparing arbitrary 3D content for interactive presentation is discussed. A method is proposed of how to integrate legacy software into environments like CAVEs™ and large tiled screens.

The next part describes applications in the field of cultural heritage, security and mobility. The symbiosis of 3D content and semantic information is an important contribution for improving the usage of virtual environments in various fields of applications.

Finally a conclusion summarizes the contributions and benefits of this work and describes possible future research derived from the thesis.

Chapter 2
The Creation of 3D Content

Abstract The process of building an interactive visualization starts with the creation of content. In this chapter the basics for the creation of 3D data are discussed. Types of data representations for digital 3D objects are described and the file format issue is addressed. Finally some of the various methods of creating 3D content are compared.

Showing interactive 3D graphics demands suitable data structures. Those structures have to be stored to a file system preferably in a standardized way. Unfortunately there is a large variety of representations for 3D data. And even more file formats exist specialized for one or multiple representations and with various features suitable for different fields of applications. Various ways of content creation exist from manual modeling over automatic acquisition to programming shapes with a modeling language.

2.1 Types of 3D Data Representations

Many different types of data representations exist to describe three dimensional objects in a digital form. To name a few, there are

Boundary Representation
Boundary and surface representations are the most common kind of data representations in computer graphics. These representations comprehend lists of triangles, polygonal meshes, spline surfaces, and subdivision surfaces, etc.

Point Clouds
In a digital documentation process, the input data set is very often a simple point cloud – measured points in space without any additional structure – recorded by e.g. a laser scanner.

Volume Data
Computer tomography and similar acquisition techniques generate volumes of measured regions consisting of many layers of high-resolution 2D images. This is the predominant acquisition technique in biomedical sciences.

Procedural Model Descriptions
In contrast to the other representations, shapes can also be described as a sequence of processing steps. This can be seen as an algorithm that generates a shape or scene.

Each type has advantages in some fields of application and there is not the single best type suitable for all purposes. However, since the modern graphic hardware is highly optimized

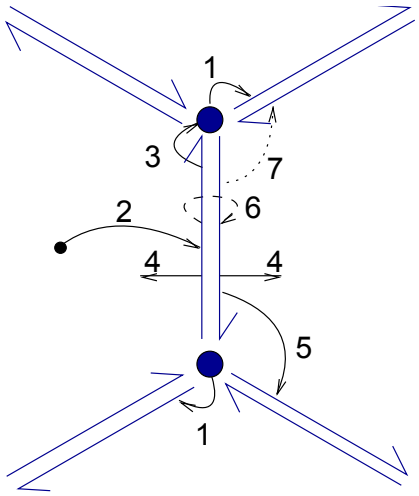
to render triangles, the polygon mesh is the most commonly used data type for interactive visualizations.

This section describes some of the representation types for surface representations. Volume representations which are important for medical visualizations and simulations in engineering are not described. An introduction to volume representations and interactive rendering of volume data can be found for example in Klaus-Dieter Engels *"Real-Time Volume Graphics"* [Eng06].

2.1.1 Polygon Meshes

In computer graphics the shape of a 3D object is often encoded in a polygon mesh. The mesh is an approximation of the surface of the shape. It is a polyhedral object. A mesh M consists of *vertices* v_i, *edges* e_j and *faces* f_k. The number of edges incident to one vertex defines the *valency* (or degree) of the vertex. The valency of the face is defined by the number of edges incident to one face.

The faces of a mesh are commonly convex primitives consisting of a few vertices. A mesh consisting only of quadrangular faces is called a quadrangular mesh (short: quad-mesh). If only triangles are used it is called a *triangle mesh*. Leonhard Euler discovered that for polyhydra the number of vertices and edges compared to the number of faces is constant $v-e+f=\chi$, which is called the Euler Characteristic. For simply connected convex polyhydra it is $\chi=2$. With the Euler Characteristic the average valency and the ratio of vertices:edges:faces can be derived for triangle meshes. If the number of vertices goes to infinity, the average valency is six. The ratio of vertices:edges:faces is $v:e:f=1:3:2$. A polygon mesh is called *manifold*, if every vertex has a disc-shaped neighborhood. In particular every edge of the mesh has to be incident to exactly two faces and every vertex has to have an edge cycle of all incident edges. That means each pair of edges and the vertex are part of the same face. If one of the two criteria is not met, the mesh is called *non-manifold*. A special case are meshes with boundaries. In computer graphics they can be defined as *manifold with boundary* if every edge of the mesh is incident to at most two faces. A mesh is called a *valid mesh* if it is manifold and all faces can be oriented in a consistent way. Some processes can only be successful if the mesh is valid.



Each vertex points to one outgoing halfedge (1).
Each face points to one adjacent half edge (2).
Each half edge points to one vertex (3),
the adjacent face (4),
the next half edge (5),
the opposite half edge (6),
optionally also the previous half edge (7).

Fig. 2.1 Example of the half edge data structure.

There are many ways to store a mesh in the memory of a computer. Some are optimized to be very compact and fast to transfer to the graphics hardware. Others are more suited to be used for mesh processing. For example the *half edge data structure* is a representation for easy navigation on the mesh. It is based on the winged-edge mesh representation which was introduced by Bruce G. Baumgart in 1975. Each edge is represented as two half edges, one for each incident face. Each half edge stores pointers to it incident face and vertices and to the opposite half edge (see Figure 2.1). In this way a traversal from edge to edge or to face etc. is possible. At the same time the data structure is very flexible and can contain face primitives of arbitrary valency.

In contrast *indexed face sets* are at least two arrays, one containing the vertices and the other containing indices. The indices are references to the vertices defining faces. If the arrays are defining a triangle mesh, it may be sufficient to interpret three sequent indices as one face. On the one hand, geometry processing and traversal in indexed face sets is more complicated. But on the other hand indexed face sets are more compact and easy to transfer to the graphics hardware. In the X3D file format definition of an indexed face set, the special index -1 is used to mark the end of a face definition. In other implementations, for example in the scene graph system OpenSG (see Section 4.1.2.1), the indices may also define quadrangles or triangle fans and strips by storing the length of one sequence in an additional array. This mixed form also stores the associated primitive types in an array. In OpenSG it is possible to have multiple indices in one index array. This can be used to address normals or colors (see Figure 2.2).

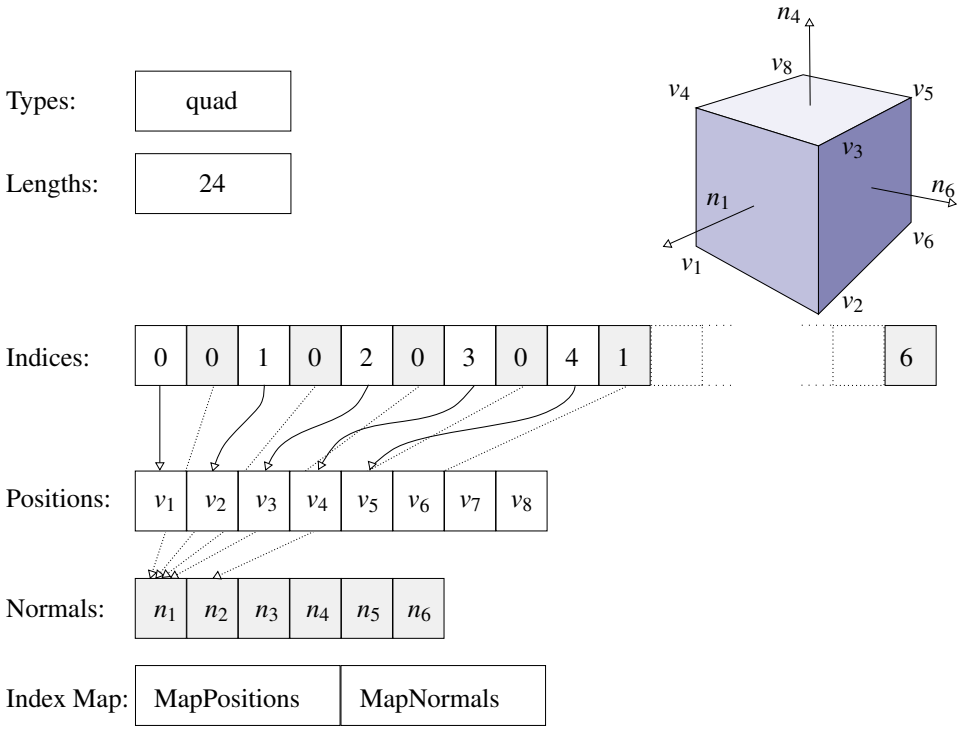


Fig. 2.2 Example of the multi-indexed geometry data structure in OpenSG.

Large meshes can be compressed to reduce the consumed disk space or to speed up data transfer over networks. A good implementation of mesh compression is the OpenCTM library by Marcus Geelnard. If the mesh is even larger it is possible that memory boundaries are reached. In that case it is possible to process the mesh *out-of-core*. The mesh is di-

vided into smaller pieces and each piece is handled separately. In 2003 Martin Isenburg and Stefan Gumbold proposed an out-of-core compression algorithm. They used a 32-bit machine to render for example a mesh consisting of 82 million triangles in 78 seconds [IG03]. Since the wide distribution of 64-bit machines the out-of-core processing is not that important any more and the general interest in this subject has fallen. Assuming a vertex or normal needs 24 byte (three double values) and a triangle is stored very wasteful as a set of three vertices, three normals and three indices to neighboring triangles, a mesh consisting of 82 million triangles needs less than 14 GB of memory. The limit for a 64 bit Windows 7 Enterprise system is 192 GB. Theoretical a mesh with a billion triangles can fit into 192 GB of memory.

An elegant implementation for the half edge data structure is *OpenMesh* by the group of Leif Kobbelt at RWTH Aachen University. The open source C++ library can be found at `http://www.OpenMesh.org`. OpenMesh is used for the development of new geometry processing algorithms. Another powerful library for triangle and tetrahedral meshes is the open source *VCG Library* by Paolo Cignoni. It can be found at `http://vcg.sourceforge.net`.

The Computational Geometry Algorithms Library (CGAL) offers a wide range of data structures and algorithms for geometry processing and shape analysis. It is available as open source at `http://www.cgal.org`.

2.1.1.1 Vertex Attributes and Textures

Besides the definition of the geometry additional vertex attributes are often used to support the graphics hardware in the render process. Vertex *normals* are necessary to describe the smoothness of the surface and for correct lighting calculations. On sharp edges a vertex has different normals for each face. For a mesh with a mixture of smooth and sharp edges a multi-index array can be used (see Figure 2.2). Another way of storing sharp edges is to duplicate the vertices with multiple normals.

To create colored meshes one or more *colors* can be connected to each vertex or to each face. The color can also be used to store precalculated lighting values. The quality highly depends on the resolution of the mesh. For independently colored surfaces a texture map is used, an image that is mapped to the geometry. One or multiple sets of *texture coordinates* can be added to map each face of the mesh to an area of a texture image. Also tiled textures can be used to add a recurring pattern to the surface.

Textures are used not only to add color but also additional lighting attributes and normals to the surface. The color of the surface is often called diffuse texture or *diffuse map*. Specularity is defined by a *specular map*. Parts of the surface can be specular while others are not. *Self illumination maps* define parts of the surface that are emitting light. A *normal map* is used to give the impression of geometric structure. This structure is visible under changing light conditions and can be used to displace a high resolution mesh with only a few polygons. Only at the silhouette the coarse mesh is still recognizable.

Natalya Tatarchuk and others have proposed shaders to simulate dents and bumps on a mesh with even more details than with normal mapping [Tat06]. The technique is called *parallax occlusion mapping*. A height map defines the offset to the surface of the mesh using gray values. A fragment shader calculates the appearance of the surface. Also self shadowing is applied. Figure 2.3 shows an example for bump mapping using a normal map (middle row) and parallax occlusion mapping using a height map (bottom row) compared to regular texture mapping (top row).

Fig. 2.3 Regular texture mapping (top row) compared to bump mapping using a normal map (middle row) and parallax occlusion mapping using a height map (bottom row).

High complexity of the geometry can be transfered to texture maps to gain performance. Tatarchuk et al. show on an example that the performance of their solution is much higher: A mesh consisting of 1.5 million triangles is converted to a mesh with 1.100 polygons using a high resolution texture map for parallax occlusion mapping. The high resolution mesh could

be rendered with 32 frames per second on their test machine while the low resolution mesh with parallax occlusion mapping was rendered with 255 frames per second. Also the memory consumption was better using the shader. The high resolution mesh needed 45 MB for the geometry and the low resolution mesh just 85 KB for the geometry and 13 MB for the texture.

2.1.2 Free Form Surfaces

Curved surfaces can be approximated using polygon meshes. For an exact representation of arbitrary surfaces other forms have to be used. In the early 1960s Pierre Bézier and Paul de Casteljau independently developed the basis for freeform curves and surfaces in the automotive industry. The most commonly used free form representation is the non-uniform rational basis spline (NURBS). They are used in automotive and industrial design to define arbitrary surfaces. NURBS patches can be stored efficiently and evaluated for every surface point. A surface curve can be defined to cut out portions of a patch. This is called a trim curve. To represent complex shapes, many trimmed NURBS patches have to be combined. For more information about NURBS see *"An introduction to NURBS: with historical perspective"* by David F. Rogers [Rog01].

In 1978 Edwin Catmull and Jim Clark presented subdivision rules for the representation of free form surfaces [CC78]. In the last years subdivision surfaces have become a popular alternative to NURBS. A variety of schemes with different subdivision rules have been developed since for geometric design and graphics applications. An overview on subdivision surface modeling in the context of computer-aided design has been presented, e.g., by Weiyin Ma [Ma05].

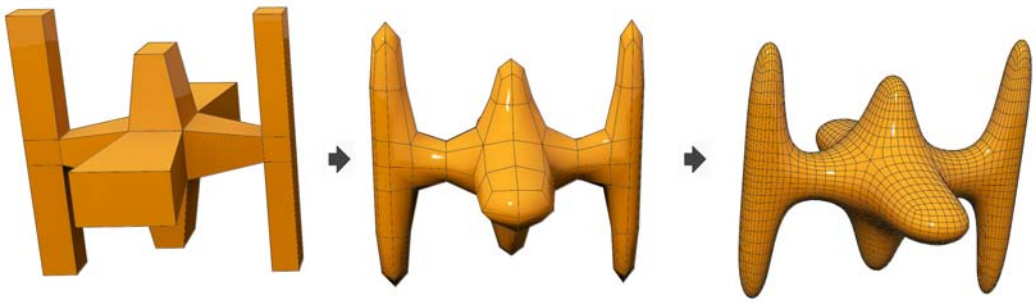


Fig. 2.4 Subdivision surfaces are defined by a control mesh (left) that is subdivided to generate a smooth surface.

A subdivision surface is created by an infinite subdivision process. In contrast to parametric surfaces which provide a finite evaluation algorithm, a subdivision surface does not come with a direct evaluation method at arbitrary parameter values. Currently, it can be evaluated via

- **Uniform subdivision:** If the subdivision rules are applied sufficiently often, the resulting mesh will be a tight approximation of the limit surface [CC78]. For non-interpolating subdivision schemes, e.g., Catmull-Clark, the resulting mesh points will not lie on the limit surface in general. In order to decrease the deviation, limit point rules can calculate the point on the limit surface for a subdivision mesh point [Zor00].

- **Adaptive subdivision:** Due to the exponential need of memory it is a good strategy to subdivide the mesh adaptively. This results in a subdivision process with varying subdivision depth but constant overall accuracy [MH00]. The use of limit point rules is essential for the connection of mesh parts with different subdivision depths.
- **Exact evaluation & conversion:** Stationary subdivision schemes, e.g., Catmull-Clark, allow an exact evaluation at arbitrary parameter values [Sta98]. Jos Stam makes use of the property that regular patches (control mesh faces with all vertices of valence 4) can be evaluated as uniform, bicubic b-spline patches. The region around irregular points (non-valence 4) shrinks successively when subdividing the irregular patches, and the Eigen-structure of the subdivision matrix is used to determine the limit there. Two alternative parameterizations for irregular patches were proposed by Boier-Martin and Zorin [BMZ04], which ensure non-degenerate derivatives of the parameterization. For Catmull-Clark subdivision, a regular quad patch can even be represented as a single bicubic Bézier patch.

Subdivision schemes can be distinguished by their type of refinement. If the edges or corners of the mesh are being cut off in the subdivision process, it is often called a *corner-cutting* subdivision scheme. The Catmull-Clark subdivision scheme belongs to the schemes of *vertex insertion* methods. New vertices are added on the edges and faces of the mesh. Some types of subdivision surface rules are limited to a specific face primitive for example only work on triangle meshes. The Catmull-Clark scheme works for any polygon mesh. The resulting mesh for this scheme is always a quadrangular mesh.

We also distinguish between *approximating* and *interpolating* subdivision methods. Using an interpolating process, the control points lie on the limit surface. In contrast, when approximating rules are used, the control vertices are typically not on the limit surface. The Catmull-Clark subdivision surfaces are approximating.

A subdivision surface is defined by a *base mesh* M^0 and the subdivision rules. For the Catmull-Clark scheme the base mesh can be of arbitrary topology. The vertices of a base mesh are called *control points* cp^0. Applying the subdivision rules to the base mesh results into a finer mesh M^1 with control vertices cp^1. The subdivision rules can be applied again to M^1. The upper index of the mesh indicates the number of subdivision steps that have been applied. The mesh M^n with control points $cp^n_0 \ldots cp^n_m$ is subdivided n times and consists of m vertices. The number of subdivision steps applied to a mesh is also called subdivision depth of the mesh. If the subdivision rules are applied infinite times, the resulting mesh converges to a limit surface $L(M^0)$.

To divide each face of the mesh, new vertices are created by an affine combination of the *1-neighborhood* of different elements of the control mesh. The 1-neighborhood of cp^l_i includes all control points, which lie on an incident edge from cp^l_i. The 1-neighborhood of a face F^l consists of all faces which are adjacent to the vertices of F^l and is denoted as $N(F^l)$. The *edge neighbors* of F^l are all faces that have a common edge with F^l. A control point $lp_i := lim_{l\to\infty} cp^l_i$ is called *limit point*. The *limit normal* ln_i is the normal of $L(M^l)$ at the border point lp_i.

The Catmull-Clark subdivision scheme consists of three rules: new vertex, new edge point and new surface point. The new vertex rule generates from a control point cp^l_i of the base mesh M^l a new vertex on cp^{l+1}_i for the mesh M^{l+1}. cp^{l+1}_i is an affine combination of the 1-neighborhood of cp^l_i and cp^l_i itself. For each edge of M^l the new edge point rule creates a new control point cp^{l+1}_j in M^{l+1}. That will be the cornerstones of the combined two vertices adjacent to the edge of space. The last rule creates for each surface F^1 of M^l a new control point cp^{l+1}_k. The weights of the rules are described in detail in the work of E. Catmull and J. Clark [CC78] and in the Siggraph Course Notes by D. Zorin and P. Schroeder [ZSD*00].

As already mentioned, the control points are usually not on the boundary that is approximated by the procedure. The limit point of a control point cp_i^l can also be calculated by an affine combination of the 1-neighborhood. In this way it is not necessary to apply the infinite subdivision rules for each control point but the corresponding limit point can be determined directly. The rules are similar to the subdivision rules for control points, but use different weights. For the limit normal the cross product is calculated for two tangent vectors. The two tangent vectors can be calculated from the 1-neighborhood of cp_i^l. The corresponding weights to calculate the limit points and limit normals can be found for example in the work of H. Biermann [BLZ00] and H. Halstead [HKD93]. By modifying the subdivision rules with other weights special features such as sharp edges can be added. Of course, the rules for calculating the limit points and limit normals need to be adjusted. Such rule modifications are described for example by T. DeRose [DKT98]. The advantages of subdivision surfaces to NURBS have brought this type of free form representation to prominent creation and construction tools. But for replacing the NURBS, subdivision surfaces need further extensions for mathematical evaluation. Kerstin Müller et al. have proposed a new subdivision scheme to combine both NURBS and Catmull-Clark subdivision surfaces [MFR*10].

Controlling the curvature of the subdivision surface around irregular vertices is a problem for many subdivision schemes. New schemes are explored to improve the visual quality of the limit surface. U. Augsdörfer et al. have presented methods for analyzing artifacts of subdivision schemes and for tuning schemes to give the best possible behavior near irregular vertices with respect to curvature variation [ADS06].

2.1.2.1 Slates for Catmull-Clark subdivision surfaces

To limit the amount of data sent to the graphics hardware, an adaptive tesselation method can be used. Only the necessary parts are tesselated. For example back facing patches can be left out. And the method can determine the necessary tesselation depth by calculating the curvature and the current size of patches. The adaptive subdivision algorithm does not need to modify the base mesh. Instead a separate data structure can be used consisting of two so-called *slates*.

A slate is composed of a two-dimensional array table of size

$$(2^{sd}+3)^2$$

and four one-dimensional corner arrays of size

$$(val-4)\cdot 2,$$

where *sd* is the maximum subdivision depth and *val* the maximum valence. For performance reasons, the slates are allocated statically as they can be reused for each face to be tessellated.

The subdivision process firstly collects the 1-neighborhood of the considered face f and stores it in the first slate. The vertices of f and the vertices of its edge neighbor faces are stored in the table. If one of the vertices of f has valence greater than four, the remaining vertices are stored in the dedicated corner arrays. Figure 2.5 illustrates this storage scheme for a quad. The subdivision algorithm processes the vertices row by row and stores the result of one subdivision step in the second slate. For the next subdivision step, source and destination slates are swapped. Other configurations and further details on slates can be found in *"Adaptive Tessellation of subdivision surfaces"* [SMFF04].

Ever since programmable graphics hardware is available there have been attempts for Subdivion Surfaces to move the creation of triangles to the GPU. Loop and Schaefer have proposed

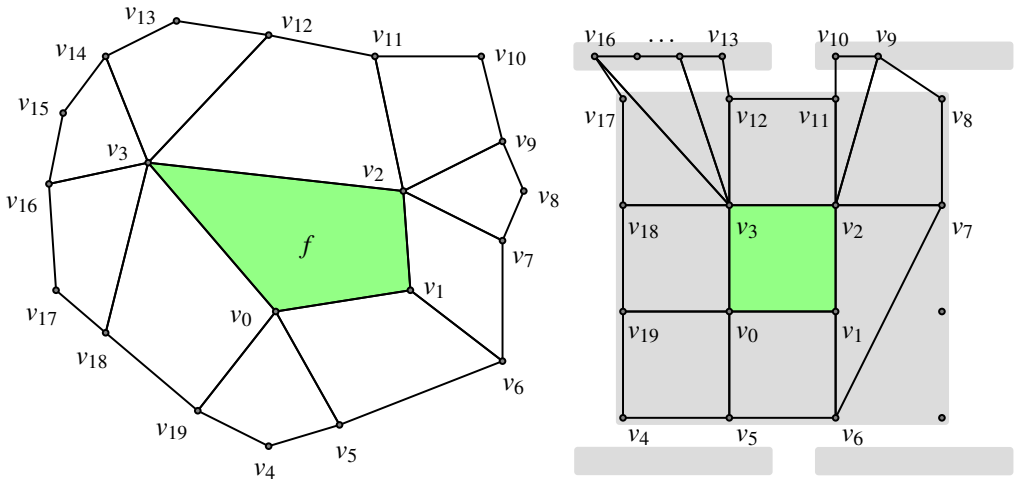


Fig. 2.5 The adaptive subdivision algorithm stores the collected 1-neighborhood of a face f from the base mesh (left) in a data structure called slate (right).

a GPU-based method approximating Catmull-Clark subdivision surfaces with Bézier patches in 2007. In 2012 Nießner, Loop et al. have presented a new method for GPU based rendering of arbitrary Catmull-Clark surfaces using the tesselation units of modern graphics hardware [NLMD12]. Their method is exact and it is the first to implement the full RenderMan specification of Catmull-Clark surfaces, including arbitrary base mesh topology, semi-sharp creases, and hierarchically defined detail on the GPU.

2.1.3 Procedural Representations

In contrast to other representations, shapes can also be described as a sequence of processing steps. This can be seen as an algorithm that generates a shape or scene. Procedural, or generative, techniques become increasingly popular because they trade processing time for data size. Instead of precomputing and compressing the tessellation of the final shape the needed information is unfolded from descriptions only at run-time. The resulting geometry can be much more complex than the description of the processing steps. Also the quality of the representation (e.g. the tessellation depth) can be adopted depending on the application.

The most common approaches use shape grammars, for example Wonka [WWSR03] and Müller [MWH*06] or scripting languages, for example the one by Snyder [SK92] and the Generative Modeling Language by Havemann [Hav05].

The Generative Modeling Language (GML) is a concrete implementation of the generative approach. It is available at `http://www.generative-modeling.org/` for Windows, Linux and MacOS as library and also in form of a plugin for Internet Explorer and Gecko-based browsers like Firefox. GML is a stack-based scripting language designed for 3D modeling. The core language is very similar to Adobe's PostScript and provides a number of operators for three-dimensional modeling, from low-level Euler operators for mesh editing, to high-level modeling tools, such as subdivision surface modeling. Another type of representation in GML is convex polyhedrons. But those representations are only the resulting data structures for rendering the shapes. The original shape definition is always stored as GML

code, a program describing the necessary steps of construction. The stack-based approach has proven very powerful to represent 2D graphics with PostScript or 3D objects with GML.

Generative techniques have great advantages since they

- make complex models manageable as they allow to identify a shape's high-level parameters, [HF04],
- are extremely compact to store [BFH05], as only the process itself is described, not the processed data,
- make the analogy of 3D modeling and programming explicit [HF01], and
- lead to much better results concerning model-based indexing and retrieval tasks [FH05].

For more information about the creation of procedural models see Section 2.4.5.

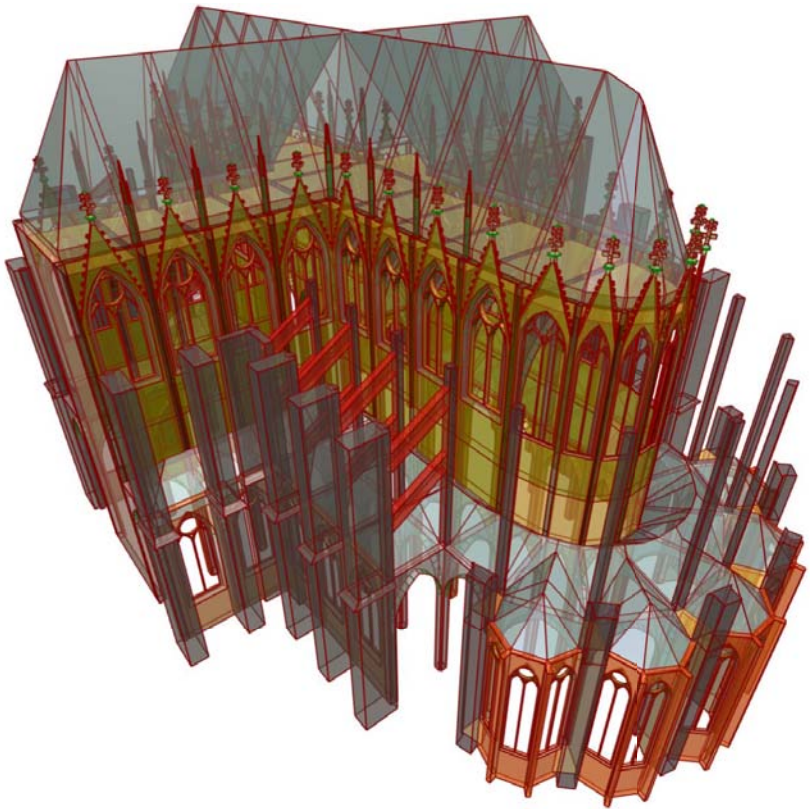


Fig. 2.6 A partial model of the Cologne Cathedral with 564.700 faces (depth 2 static subdivision). The GML code file size is only 133 KB (uncompressed text). (image courtesy of Sven Havemann).

2.2 Scene Graphs

Virtual worlds are composed of many 3D objects. It is not uncommon to use different kinds of representations for 3D objects at the same time. The collection of objects forms a 3D scene. This collection can be organized in an array and processed one by one. For large scenes a hierarchical structure like a graph or a tree is more convenient and faster to process. Typically a scene graph is a directed acyclic graph: Each node may have arbitrary children but only

one parent node, cycles are not allowed. The first node of the hierarchy is called root node from which all of the nodes in the scene can be accessed by traversing through the graph. The nodes are parts of the visible geometry of the scene or represent other functionality like transformations and groups.

Effects on a node are passed on to the children. Spacial relations between objects can be expressed like this. If for example a cup node is added as child of a table node the transformations applied to the table node will also affect the cup.

With a scene graph it is possible to apply high level optimization like visibility culling. This is hardly possible using a low level API because the renderer is not aware of the whole scene.

A visibility test for a node is typically performed by intersecting the node with the planes of the view frustum. Often a bounding volume hierarchy is used for fast visibility tests. Bounding volumes of low complexity for example a sphere or a box are fast to compare. The bounding volume of a node is defined by the bounding volumes of the nodes children. If a visibility test for a node fails, also its children will not be visible and the whole subgraph does not need to be send to the graphics hardware.

In many scene graphs it is possible to reuse geometry in multiple locations. This is called instancing. Multiple nodes contain a reference to the same geometry. In the same way other resources of the scene may also be reused like materials, shaders and transformations. Instancing helps to reduce memory consumption and can speed up the rendering of identical objects.

2.3 File Formats for 3D Content

To store 3D data to hard disk and to exchange data between machines and applications, file formats have been developed. For every data representation there a many different formats which leads to hundreds of file types for 3D content. To get an idea of the huge amount of different formats, have a look at *Deep Exploration*[1]. Deep Exploration is a popular conversion tool. It can read in and write out 3D data in many formats. The documentation for version 6.5 lists 52 file formats for 3D data which are supported by the standard edition of the tool. The CAD edition adds another 26 file formats which are mainly used by applications for computer-aided design (CAD). Table 2.1 lists the formats and their file extension.

CAD applications and other content creation tools often use proprietary file formats. For example the AutoCAD Drawing Object format (DWG) is a closed format owned by AutoDesk. The specification is restricted and other applications than AutoCAD are not allowed to read or write the format unless a license agreement is met with Autodesk.

JT

A quasi-standard format in the automotive industry in Germany is JT[2] which stands for *Jupiter Tessellation*. It was developed by Engineering Animation, Inc. and Hewlett Packard, later by Unigraphics for 3D product data. Since 2007 it is part of the Siemens product lifecycle management (PLM) software. JT supports the typical CAD representations and features like NURBS geometry, polygons and metadata. The format also supports data compression, levels of detail for large data sets and different levels of access control. As many CAD applications support JT it can be used for transferring CAD data from one application to another. The specification was

[1] http://www.righthemisphere.com/products/dexp/
[2] http://www.jtopen.com/

File Format	Extension	File Format	Extension
AutoCAD Design Web Format	DWF	AutoCAD Drawing Object	DWG
AutoCAD Drawing Interchange	DXF	AutoCAD Sheet Set File	DST
Autodesk 3D Studio	3DS	Autodesk 3D Studio ASCII	ASC
Autodesk 3D Studio Max ASCII	ASE	Autodesk 3D Studio Project	PRJ
AOFF File Format	GEO	Caligari trueSpace Object	COB
Caligari trueSpace Scene	SCN	CINEMA 4D	C4D
Collada	DAE	Deep Vision Scene	DVS,IDS
DirectX Model	X	FiLMBOX	FBX
Gerber File	GBR, GBP, GPT, GTL, GTO, GTS, GKO, GBL, GBP	Google SketchUp Document	SKP
		Imagine Geometry	IOB
		ISO G Code Files	ISO, NC
		LightWave 3D and Binary Object	LWO, LW
		LightWave Scene	LWS
Max File	MAX	Maya ASCII Scene	MA
Maya Binary Scene	MB	Nendo	NDO
Object File Format Vector Graphics	OFF	Open Inventor File	IV
OpenFlight Scene Description	FLT	OpenGL CPP Code (export only)	CPP
Points File	PTS	Polygon Model Format	PLY
Portable Graymap Graphics File	PGM	POV-Ray RAW Triangle Format	RAW
Power Render Object	PRO	Protein Data Bank	PDB
Quicktime 3D Metafile	3DM, 3DMF	RAX Extended RAW Triangles	RAX
Rhinoceros 3D Model	3DM	Right Hemisphere Binary	RH
Shattered Assembly File	XML	Shockwave 3D File (export only)	W3D
Shockwave Web Page (export only)	HTML	Silo3D	SIA
SOFTIMAGE—XSI	XSI	Stereolithograpy Rapid Prototyping	STL
Universal 3D	U3D	Visual Design Stream	VDS
Visual XML	RHZ	VRML Worlds	WRL, VRML
Wavefront Object	OBJ	XAML (exportv only)	XAML
ACIS Part and Assembly	SAT	CADDS Part and Assembly	_PD
CATIA4 Assembly	SESSION, EXP	CATIA4 Part	DLV, MODEL
CATIA5 Assembly	CATPRODUCT	CATIA5 Part	CATPART, CATSHAPE
CATIA5 Drawing	CATDRAWING		
CATIA5 Graphical Representation	CGR	IGES Files	IGES, IGS
Inventor Assembly	IAM	Inventor Part	IPT
JT File	JT	MicroStation CAD Graphic	DGN
NX Parts and Assemblies	PRT	Parasolid CAD File	X_B, X_T, XMT, XMT_TXT
Pro/ENGINEER Assembly	ASM, XAS		
Pro/ENGINEER Neutral	NEU	Assembly and Part Files	
Pro/ENGINEER Part	PRT, XPR	Pro/ENGINEER Drawing	DRW
SolidEdge 3D CAD Model Assembly	ASM	SolidEdge 3D CAD Model Part	PAR, PWD, PSM
SolidWorks Assembly Files	SLDASM, ASM	SolidWorks Part Files	SLDLFP, SLDPRT, PRT
SolidWorks Drawing	SLDDRW		
STEP Files	STEP, STP	VDA File	VDA, VDAF

Table 2.1 3D file formats supported by Deep Exporation version 6.5. The formats in the lower part are only available in the CAD edition (Source: Deep Exploration documentation)

published as an ISO standard in late 2012, the official software tool kit is a commercial product of Siemens.

There are also open file formats for 3D content. The standard solution for extensible data formats is XML. Its greatest practical advantage is that it makes writing parsers obsolete, which is tedious and error-prone. A single parser, the XML parser, is sufficient to support all XML based data formats. The structural integrity of a file can even be tested automatically when a *document type definition* (DTD) is available.

Its usefulness is illustrated by the VRML ISO standard from 1997 [VRM97]. Originating from SGI's *Inventor* format, but designed as extensible (via the infamous *PROTO*), the success of VRML was impeded by the very complex parser it requires. This problem was solved using XML.

X3D

X3D, which is short for Extensible 3D, is the XML-based successor to the Virtual Reality Modeling Language (VRML). It is like VRML focused on but not restricted to the presentation of 3D scenes in a web browser. X3D was approved an official ISO standard in 2004 and is maintained by the Web3D consortium. [BD07]

X3D files can be encoded in three formats: XML, VRML syntax and a compressed binary which can also be encrypted. The standard includes the definition of 2D and 3D geometry. 3D Geometry can be defined as primitives, like box, sphere etc. and as polygons called indexed face set. X3D also supports trimmed NURBS patches, CAD geometry and volume data. Materials and shaders are supported in many variations. Animations can be defined as keyframe animations. JavaScript can be embedded into an X3D file.

X3D still carries many legacy concepts. The main problem is the ambitious intention to create a general language for the description of "virtual worlds", meaning complete VR applications. As a consequence, X3D describes many data that are today part of the application. X3D specifies the navigation (walk/fly/orbit), an event system with triggers and interpolators, and a data flow model (*ROUTE*). The interaction facilities are rather limited, of course, compared to high-end computer games (Doom, Halflife etc.). Things like the outdoor background in a 3D computer game, for example, are so complex that they are realized by the application. The simple VRML backdrop (color gradient or texture) is insufficient. X3D's generalization to the *background stack* on the other hand nails viewers down to a particular background model, a very limiting policy as well.

Collada

Collada (COLLAborative Design Activity) was initiated in 2004 for Sony Computer Entertainment by Rémi Arnaud and Mark C. Barnes [AB06]. It is maintained by the Khronos[3] group, a non-profit industry consortium. The idea of Collada is to define an exchange format for 3D computer games content creation tools. It is meant to be an additional way for different middleware and tools to communicate with each other.

The XML-based format defines geometry in the form of polygons and lines and with Version 1.5 also boundary representation structures. Materials and shaders are supported as well as animations.

Import/export plugins exist for example for Maya, 3DStudio Max, Softimage, Blender, the primary digital content creation (DCC) tools in the sector of game development. Google Earth uses the Collada file format for augmenting the 2D map with 3D models of buildings created, e.g., using Google Sketchup.

A Collada file has two main parts, the library and the scene. The library defines the entities (geometry, materials, lights, etc) for the scene, where they may be used once or multiple times ("multiple instancing"). Almost every major Collada node type, also the scene itself, can have an <extra> element attached. It contains one or more <technique> subtrees, which is the Collada mechanism to cope with different software capabilities: Maya, for instance, stores an object as a Collada standard mesh, and attaches Maya specific information using <technique profile="Maya">. The Collada specification demands that also "unknown" <technique> content has to be preserved, i.e., the attachments survive import and subsequent export by any Collada compliant software.

[3] http://www.khronos.org/collada

PDF

The Portable Document Format (PDF) is the standard file format for textual documents first released in 1993. It is developed and maintained by Adobe Systems.

Since version 1.6 (from 2005) PDF supports the inclusion of 3D content. PDF is used intensively in the manufacturing industry as exchange format for annotated 3D models. Sections and 2D drawings are insufficient for describing free-form surfaces. So PDF export functionality is available in almost all major CAD applications, just to mention Dassault Catia V5, Bentley MicroStation and Graphisoft ArchiCAD. Commercial solutions like PDF3D from Visual Technology Services[4] allow integrating PDF export also with existing production work flows and applications. PDF is used for product presentation and explanation, e.g., for 3D user manuals that explain unambiguously the assembly and disassembly of complex machine parts.

And also free and open source software exists for adding 3D content to a PDF document. The Meshlab software by Paolo Cignoni et al. allows editing 3D meshes and converting many 3D file formats to Universal 3D (U3D). The media9 package for LaTeX by Alexander Grahn provides means for embedding various multimedia objects into PDF files generated by LaTeX. 3D data has to be converted to the U3D format or to the Product Representation Compact (PRC) format to be used inside a PDF file. U3D files and PRC files can be embedded, and JavaScript can also be attached to the PDFs' 3D data streams. However, the annotation of a 3D model must still be done using an authoring tool like Adobe Acrobat.

Barnes and Fluke [BF08] propose the incorporation of static 3D content in PDF documents as a substantial improvement for the visualization of multi-dimensional data in astronomy research papers. Since multi-dimensional data is no astronomical phenomenon, their ideas for a more appropriate knowledge transfer by using 3D visualizations is of course also applicable in numerous other fields. Strobl et al. [SBS*09] describe a workflow for the generation of PDF documents that contain 3D objects, which are dynamically enriched with embedded annotations to provide additional information on some specific parts of the 3D content. Berndt et al. propose an updated workflow for publishing cultural heritage objects as 3D models with embedded annotations and additional information in a single PDF file [BBH*10] (described in Section 3.4.2).

Since PDF is an ISO standard (ISO 32000-1:2008) and the Acrobat Reader is available on a variety of platforms, it is an ideal format for bringing rich and interactive 3D content to a large audience.

2.4 Methods of Creation

Depending on the area of application different methods have been and are still developed to create 3D content. Often a combination of those methods is used to produce 3D content. The most common methods are *modeling tools* to create digitally born models and *laser scanning* to digitalize real-world objects. Relatively new is the possibility to create 3D content from a set of photos. The long known method of defining algorithms to create 3D models has become more popular lately to generate large amounts of geometry automatically.

[4] http://www.pdf3d.com/

2.4.1 3D Modeling Tools

Creating 3D content is an essential part of many projects and it is an important cost factor. Software tools for 3D modeling are available in many variations and they all want to enable the creation of 3D shapes in a convenient way. Tool providers come up with good arguments to distinguish themselves from their competitors. The big players like Autodesk or Dassault offer integrated solutions for modeling, animation, simulation and rendering. Their advantage is that 3D content can more easily be exchanged between the editor and the render or the simulator for example.

Other providers of creation tools are specialized on one or a few aspects. For example McNeel offers the excellent NURBS modeling tool Rhinoceros. And the sculpting tool ZBrush by Pixologic is very popular among artists. For applications like these it is important to create data which can be integrated into an existing tool chain.

Generally creation tools can be roughly grouped into the two categories *3D Construction* and *3D Entertainment*.

3D Construction

These tools are optimized to support the process of computer-aided engineering, architecture, etc. The resulting 3D data has to be of high precision and ready to be used in the production workflow. Main users are professionals like draftsmen and architects.

Often the creation process starts with a set of technical drawings. In this way the resulting content can also include construction plans that can be used directly for the production of the object. For example architecture plans are used to plan the construction on a building.

Common basic primitives of these creation tools are curves on which the 3D shape is defined. For the creation of surfaces it is important to have full control over the parameters. This is probably a reason for the preference to NURBS instead of subdivision surfaces. NURBS can be evaluated easily for every point of the surface. This is essential for the correct construction of real world objects. Subdivision surfaces are used in the conception phase though.

For Product-Lifecycle-Management it is important that the construction tools can be integrated in existing production systems. For example they can access a data base of components and keep track of their usage. Collections of components can be used to create a detailed listing of the construction parts with a cost estimate etc.

3D Entertainment

Many tools specialize on the entertainment sector which has become an important area for 3D content creation. The results of 3D modeling are not used to generate a functional machine or building but rather fictional objects which are used in movies, video games and for marketing purposes. Precision is not the main focus of these tools. More important is the visual quality, low render times or matching polygon counts. These tools are mainly used by professional designers and artists.

Lately creation tools have become more convenient by adding sculpting and sketching inputs. This is inspired by the classical work of an sculptor or painter. And the performance of todays work stations is sufficient to simulate for example behavior of clay in pure digital form (excluding the haptics). This is proved by the popularity of tools like ZBrush by Pixologic or Mudbox by Autodesk.

More and more also semi professional users and hobby artists are addressed by tools like Google Sketchup. Especially virtual 3D city maps need a lot of content and many users are willing to contribute. And also the games industry is encouraging their customers to create custom content. Those creation tools have to be easy to use on the one hand but give enough freedom to create interesting content.

2.4.2 Laser Scan

A common way of digitalizing a real life object is to measure its surface with the help of an optical device. In many scanning devices it is laser light that is used to illuminate a specific portion of the object. Therefore, the generic term for digitalizing real-world objects is *laser scanning*, even if other light sources are used.

With the help of triangulation it is possible to generate 3D points from the illuminated surface portions. An array of those points is a so called *range image*. It is an image with a depth value for each pixel. Multiple range images from different points of view can be arranged to generate a 3D model of the scanned object. This process is called registration. Depending on the complexity of the scanned object, hundreds of object parts have to be registered to create one consistent data set.

The registration is relatively easy if the exact location and orientation of the scanner is known. This can be achieved by tracking the scanning device with appropriate hardware. As the tracking of the overall scanning process is laborious' especially for large objects like cathedrals, a software solution for registration is desired.

The software registration process is typically separated into a coarse and a fine registration step. During the coarse registration step, all object parts are transformed into a less detailed representation. In the last few years, many of these representations have been proposed. Their purpose is to reduce the quadratic complexity of pairwise data set alignment. Furthermore, the computation time is shortened by selecting only the most important features of the data set. The main task of automatic registration is to minimize the distance of overlapping parts. One of the algorithmic challenges of the coarse step is to figure out the correct arrangement of repeating object structures. This procedure takes a lot of time, particularly if it needs user interaction.

The Visual Computing Lab (VCL) in Pisa, which is supported by the Italian National Research Council's Institute of Information Science and Technologies, has developed an automatic registration technique. It considers the knowledge of the regular scanner pose pattern. Under the condition of a sufficient overlap (15-20%) and of regular scan stripes, it simplifies the sequential arrangement of range maps.

To align one range map to another one, three pairs of corresponding points are needed. Normally, the alignment problem is solved iteratively until an error threshold condition is met.

The entire registration of the VCL algorithm relies on a variance classification for each point of the range map regarding its neighborhood. This variance helps to cluster the range map into parts of similar curvature. As regions with high variance may be created due to scanning errors and regions of low variance do not contain enough characteristics, the algorithm does not take them into account. The iterative alignment procedure limits the choice of point pairs to points of similar variance.

The fine registration of the range maps is typically done using the iterative closest point (ICP) algorithm. Details of both algorithms are described by Pingi et al. in *"Exploiting the scanning sequence for automatic registration of large sets of range maps."* [PFC*05]. A de-

tailed overview of laser scanning can be found in *"The 3DModel Acquisition Pipeline"* by Bernardini and Rushmeier [BR02].

Different types of laser scanners are available on the market for different purposes: Cultural heritage objects can be documented and preserved by scanning them. Engineers use laser scanning for the planning of renovations. Complex product parts can be measured and documented for quality control. It is used for reverse engineering to construct spare parts for objects for which no plans exist. Designers can use traditional ways of modeling, for example using clay, and scan the result to get a virtual 3D model.

In Civil Engineering entire landscapes are scanned to examine and control construction sites. In the archeology scanning is used to measure and document excavation sites.

Scanners for small objects are not expansive. The Next Engine[5] desktop scanner for example costs less than US $3000. And construction descriptions to build a laser scanner for less can be found for example in an article by Simon Winkelbach et al. [WMW06].

Fig. 2.7 This is an example for scanning an object. From top left to right: The Object is scanned using the Next Engine Desktop Scanner, each scan covers only a fragment of the object, fragments of 10 scans, there are still holes in the object; bottom left to right: colored fragments, fused to a single mesh with automatic hole filling, ambient occlusion and texture filtering.

An example for scanning an object is shown in Figure 2.7. The scan is done with the Next Engine desktop scanner. A figurine made of clay is used as example object. The dull surface

[5] http://www.nextengine.com/

and the light brown color of the small bear can be scanned without any preparation. Dark or glossy surfaces have to be prepared with talc powder or spray paint.

The object is placed onto a rotating table which is directly controlled by the scanner software. For example, a 360 degree scan was selected with 9 scans. The scanner automatically scans the object, rotates the table and scans the next part. Before the actual laser scan starts, the scanner also captures the texture by taking a photo. After the 9 scans another single scan was necessary for the top of the head as the scanner could not measure it from the side view. In the end all the fragments are registered to each other. That means they are placed in space to connect to one object. If the overlap of the fragments is big enough, the scanner software can register the parts automatically. If not, the user can place reference points on the fragments manually. This had to be done for the single top view scan.

The scanned object may still have some holes. In the example the top area of the feet of the object is not scanned. If the holes are small, an automatic hole filling can be performed. For big holes and missing parts additional scans have to be done. For objects with many self occlusions the scan process can be very time consuming because many additional scans have to be done to cover the surface. For every additional scan there has to be enough overlap to previous scans in order to register the resulting range image. In some cases it was easier to add additional objects to the scan volume and use them as reference for the registration.

The raw scanning data consists of a range image for each scan and the scanners configuration parameters. The scanner software creates triangles and texture coordinates. For the final 3D model the color of the object can be stored per vertex or as texture maps. For each range image there is one texture map. This large amount of data may have to be further processed to generate for example a useful 3D model for interactive rendering.

For the example object of Figure 2.7 highly tessellated meshes with vertex colors were created and further processed in MeshLab[6]. The open source application has far more features for processing the meshes than the Next Engine scanner software. A single mesh was created using the Poisson surface reconstruction filter. The vertex colors were transfered to the new mesh. In the color information there were some borders visible from the stitching of the different texture images. After applying the Laplacian smooth filter for the vertex colors the borders disappeared. Finally Ambient Occlusion was added to enhance the contrast of the colors and visually increase the structure.

Laser scanning gives an accurate representation of real objects. The method works reliable for all kinds of dull surfaces. It tends to fail for very dark material, because the surface absorbs the laser light. In the result the optical device cannot receive enough information. Also very glossy surfaces and transparent materials are problematic. In those cases it helps to prepare the surface with talc powder or, if possible, repaint the object with a matte white color. For colored objects a dulling spray can be used to create a matte surface while preserving the colors.

2.4.3 Image-based 3D Reconstruction

3D content can be generated from a sequence of images. This discipline is called image-based 3D reconstruction. From the usability point of view, an image-based reconstruction is easier than laser scanning – everyone can handle a camera. The complexity of this approach is hidden within the algorithms used to generate a 3D model.

To build a 3D model out of a sequence of images the extrinsic and intrinsic camera parameters have to be calculated. The extrinsic parameters are the position and orientation of

[6] http://meshlab.sourceforge.net/

the camera. They change with every image. The focal length, the image format and the principle point are the intrinsic camera parameters. They are independent from the position and orientation of the camera and in general do not change with every image.

The camera parameters are calculated by finding a set of corresponding points in the photo sequence. Two points in different photos are corresponding if they show the same 3D point of the photographed object. At least seven pairs of corresponding points have to be determined for two photos (see Fig. 2.8).

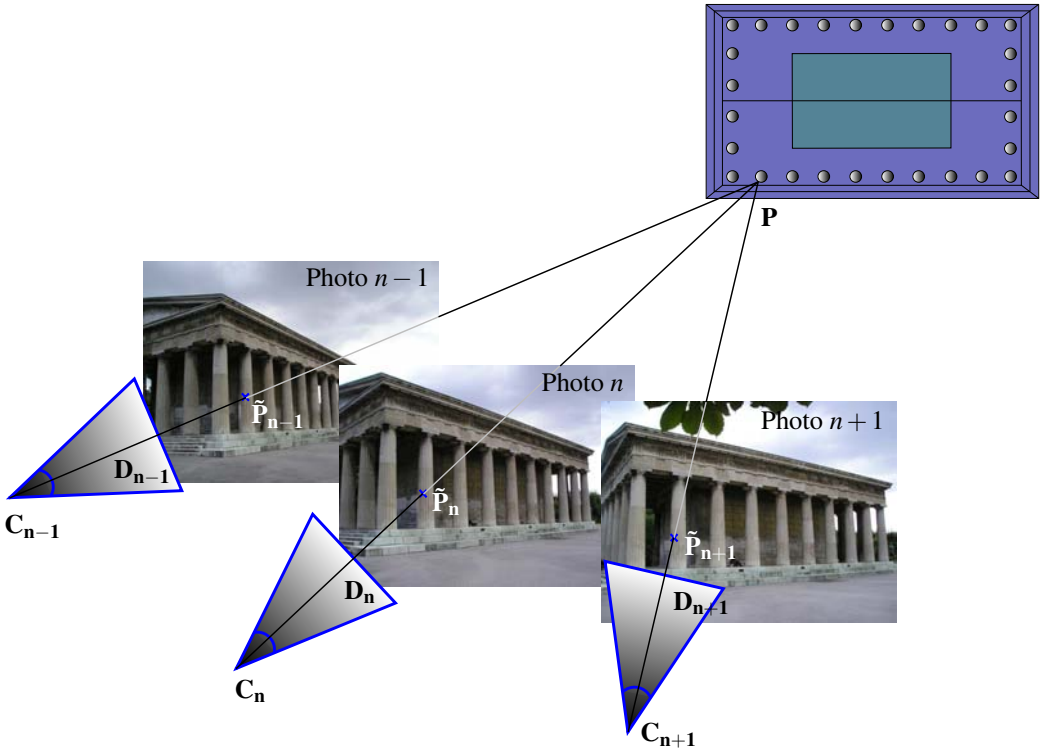


Fig. 2.8 A photogrammetric reconstruction uses differences in a sequence of images to determine the three dimensional coordinates of an object. One of the main steps of an image-based reconstruction is to identify corresponding points $\tilde{\mathbf{P}}_\mathbf{i}$ which are images of the same 3D point $\mathbf{P}$. Having identified a set a corresponding points in the sequence of images, the theory of epipolar geometry allows to determine the cameras' parameters (position $\mathbf{C_i}$, orientation $\mathbf{D_i}$, etc.).

Finding the corresponding points can be solved in many ways. Several common approaches can work iteratively to solve this chicken and egg problem. It is easy to identify corresponding points when the underlying 3D model is known. It also is possible to reconstruct the photographed point in 3D when its correspondences in all images are known.

A more general technique has been proposed by Shinagwa and Kunii. They use so-called multiresolutional critical-point filters to automatically match images [SK98].

The main idea is to trace corresponding points recursively through a multiresolution hierarchy. To calculate the point correspondence of two points at level i the algorithm uses the already calculated correspondence at level $i-1$ and an error function. At level 0 the images have a size of 1×1 pixel and the point correspondence is trivial, as there is only one possibility.

The fact that no prior knowledge about the objects is necessary makes the algorithm attractive for a wide range of applications such as stereo photogrammetry, fully automatic morphing, object recognition, and volume segmentation.

To reduce the amount of potentially corresponding points, almost every algorithm solving the correspondence problem uses an upstream filter, e.g., a Sobel or Laplace filter as shown in Figure 2.9. In many cases, filters produce similar results for corresponding points in both images. For example, a point on an edge will most likely have a corresponding point that is also located on an edge. Therefore, an edge-detection filter reduces the number of possible correspondences significantly.

Fig. 2.9 A Laplace filter eliminates same-color regions and highlights edges. Assuming that a point on an edge will have corresponding points that are also located on edges, a Laplace filter reduces the amount of input data to solve the corresponding point problem and to find a stable solution.

Having the camera parameters, a dense matching can be performed for many photos leading to range maps similar to the laser scanning process. Those range maps are used to create a 3D model of the photographed object. An example is shown in Figure 2.10. This step is similar to the fusion of laser scanning range maps. But in contrast to laser scanning, to generate true-to-scale geometry, reference points are needed.

The quality of the range maps depends on the image quality of the photos and the dense matching results. If a pixel of an image is matched consistently in many other images, the accuracy of the pixels 3D position is increased. But to get results even for a small number of photos, the reconstruction algorithms allow a certain amount of inaccuracy. Therefore there is more noise in the range maps of an image-based 3D reconstruction compared to a laser scan.

For smooth surfaces some filtering can be applied in the surface reconstruction but this leads to smoothened edges as well. Areas of the same color, for example uniform walls cannot be matched. But also the sky or the reflection of the sky in windows is impossible to reconstruct.

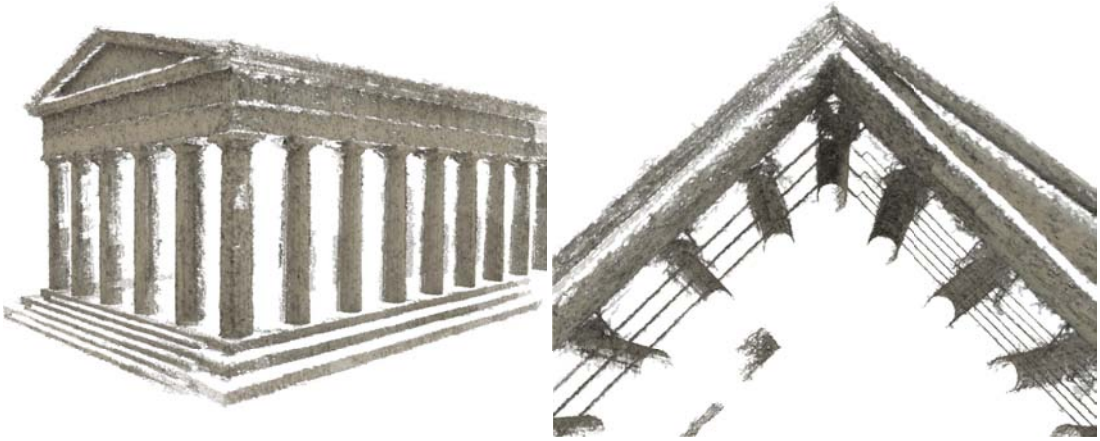


Fig. 2.10 The image-based 3D reconstruction of the Theseus temple for one sequence of photos. To reconstruct the whole temple, several photo sequences are necessary.

For *"A Publishing Workflow for Cultural Heritage Artifacts from 3D-Reconstruction to Internet Presentation"* we used the image based photogrammetry for over 20 objects of the Gipsmuseum in Graz (`http://gipsmuseum.uni-graz.at`). The museum has a collection of mostly plaster statues held by the Institute of Archeology, University of Graz. The creation of replica of statues and other decorated objects has a long tradition. At first the plaster copies were mainly used by sculptors to serve as masters for copies of the original works. With the appearance of the modern universities in the 18th century they also served another purpose. The plaster sculptures were and are still used as a three dimensional reference of the ancient art in lectures and studies of arts and history. Some of the exhibits are quite old and a few are the only remains because the original statues were damaged or destroyed. There are enough reasons to digitalize and preserve the plaster copies. In 2008, our group started to reconstruct some of the exhibits of the museum. The goal was always to generate precise digital models in a cost efficient way. We decided to reconstruct the statues using the ARC 3D Webservice of the University of Leuven [VG06]. This service is freely available for the task of preserving cultural heritage. It uses a sequence of photographs to calculate depth images. Those depth images can be used to create a 3D model.

A group of four summer interns was trained to shoot photos of high quality. The photos were taken with a Nikon D60, a 6 megapixel single-lens reflex camera. A tripod and a remote-control release were used to avoid camera shake. Because of the homogeneous color of the statues a photogrammetric reconstruction is not trivial even for good algorithms. We decided to use a random color pattern which was projected onto the plaster using a projector (see Fig. 2.11). We tried to match the size of a pixel of the projected pattern with the size of a pixel on the photo. To also capture the correct color of the surface it would be necessary to shoot a second photo without the projected pattern. Due to lack of time this was not done for all photos but only for three shots (start, middle, end) of a sequence.

To get good results, the process of taking photos has to meet some special demands which are described on the ARC 3D web page. For example a sequence of photos has to be shot from different positions and not by just rotation like shooting a panorama. It helps to plan the

Fig. 2.11 This is one of the photos used for reconstruction. The projected random color pattern helps to identify feature points on the otherwise homogeneous surface.

movement of the camera in advance to make sure, the object of interest is fully visible in all shots. All of the photos have to be sharp, defocused and blurry images will cause problems. Adjustments of the zoom are not allowed during a sequence as well as changing the resolution of the photos. For that version of the ARC 3D service (2009) it was important to use the same camera settings for all shots of a sequence. In the meantime some of the restrictions are loosed. To make it more convenient for the user, the team of the ARC 3D service has made the process of finding matching features more robust. It is now possible to shot images in any order and with any settings. Still the images have to be sharp, of course, and covering the scene with enough overlap (ideally 90 %).

Within the time of the internship we managed to shoot 25 exhibits. For each statue an average of 8 sequences was taken with approximately 20 photos per sequence. For the more complex statues we had to take more sequences. Also sometimes it was hard to move around the exhibits so we had to take more sequences with only a few photos. In total we took about 4300 photos and sent them to the ARC 3D Webservice.

The resulting depth maps were then processed using Meshlab. In the first step, the resulting 3D fractures of each photo sequence were created using the Poisson surface reconstruction approach. To get a single model for each statue, the fractures were then registered to each other. In this step it was necessary to scale the parts of the model according to measurements taken from the real statues. In contrast to a laser scan, the photo reconstruction is not creating accurate measurements. Manual corrections were needed to generate exact digital copies. The scaling and the final assembling of the parts can also be done in Meshlab. But because of some stability issues we used GeoMagic Studio 3, a commercial solution for the depth map registration[7]. An example statue is shown in Figure 2.12.

Photogrammetry is the easiest way for non-expert users to digitalize objects. Learning how to take photos the right way for reconstruction is done within minutes. In the beginning the procedure to get good 3D models was a lot of manual work. But commercial products like 123 D Catch (Autodesk) make it really easy to generate useful content. The user selects photos,

[7] http://www.geomagic.com/

Fig. 2.12 A photo of a plaster statue and the reconstructed 3D model in Meshlab.

waits for the cloud to compute the range maps, checks for errors and gets a 3D model with a single texture map. The reconstruction has problems with large uniform areas with no detailed structure as well as with glass and other transparent material. Also the process fails of course if too few photos are taken of the object.

2.4.4 Model Reconstruction

The creation of consistent and accurate model descriptions is known as reverse engineering and comprehends fitting, approximation, and numerical optimization techniques.

If the underlying model description is able to describe every 3D object in a consistent and integrative way without the need of an additional superstructure, the reconstruction process can be called complete. Otherwise, an object is described by several small parts for which the orientation to each other is stored in a superstructure, e.g., a scene graph. This approach is a subpart reconstruction.

In 1992, Hoppe et al. presented *"Surface reconstruction from unorganized points"*, an algorithm that fits a polyhedral surface to an unorganized cloud of points [HDD*92]. The result is a polygonal mesh that describes the complete object. Further development of polygonal reconstruction has led to algorithms, like the crust algorithm, with outputs that are guaranteed to be topologically correct and convergent to the original surface as the sampling density increases.

As polyhedral surfaces are not the only way to describe a 3D object in a unified way, the class of complete reconstructions also comprises algorithms to fit radial basis functions, constructive solid geometry, or subdivision surfaces, to name a few.

The subpart fitting approaches can be divided into two categories depending on whether or not the input data has to be segmented and partitioned in advance.

Among others, no preceding segmentation is needed by RANdom SAmple Consensus (RANSAC)-based algorithms. The basic idea of RANSAC methods is to compute the parameters of a model from an adequate number of randomly selected samples. Then, all samples vote on whether or not they agree with the proposed hypothesis. This process is repeated until a sufficiently broad consensus is achieved. Two major advantages of this approach are its ability to ignore outliers without explicit handling, and be extended to extract multiple model instances in a data set. In this way, it is possible to determine, for example, the planes that describe the input data best. Although these algorithms can take advantage of feature extractions, they do not rely on them. Ullrich et al. show how to speed up RANSAC-based algorithms using statistical estimators [UF11].

Feature extraction algorithms determine feature lines such as crease loops and junctions, or border lines. These algorithms use principal component analysis or heuristics on local neighborhoods to classify points according to the likelihood that they belong to a feature line.

These local feature vectors separate the input data into smaller regions, which can be approximated individually by a single surface as illustrated in Figure 3.5. A sequence of tests splits a large point cloud into smaller and smaller subregions until no further subdivision is sensible.

These model fragmentations are needed, for example, by subpart reconstructions based on nonuniform rational B-splines (a commonly used surface description in computer-aided design), developable surfaces, or least squares fitting techniques.

The aim of the remeshing and reconstruction process is to extract high-level geometry and to generate consistent and accurate model descriptions as shown in Figure 2.13.

2.4.5 Algorithmic Creation

Algorithmic solutions for creating 3D content are often used for vegetation and similar organic objects. For example the Lindenmayer systems can be used to generate plants and trees. As the need for large 3D scenarios has increased drastically for movie productions and video games, manual modeling would take much too long. The solution is to give a machine the ability to create variations of equivalent models itself. For example building a detailed city model leads to a lot of cloning and individual modeling for a huge amount of very similar objects. This process can be automated using an algorithmic approach.

The most common approaches use shape grammars [WWSR03], [MWH*06] or scripting languages [SK92], [Hav05]. Using these techniques urban modeling has become manageable [PM01], [WW08]. All approaches base on shape templates, which are a representation of a construction process. Therefore, they are similar to algorithms. A shape template does not represent a single object but a family of objects. Various modeling frameworks offer the possibility to define shape templates easily [PPV95], [GK07] [LN03], [MPB05].

If the structure follows a set of rules, an algorithmic approach is able to create a large variety of 3D models. For example the creation of Gothic windows using the Generative Modeling Language (GML) is described in detail in the PhD thesis of Sven Havemann [Hav05] (see also Section 2.1.3).

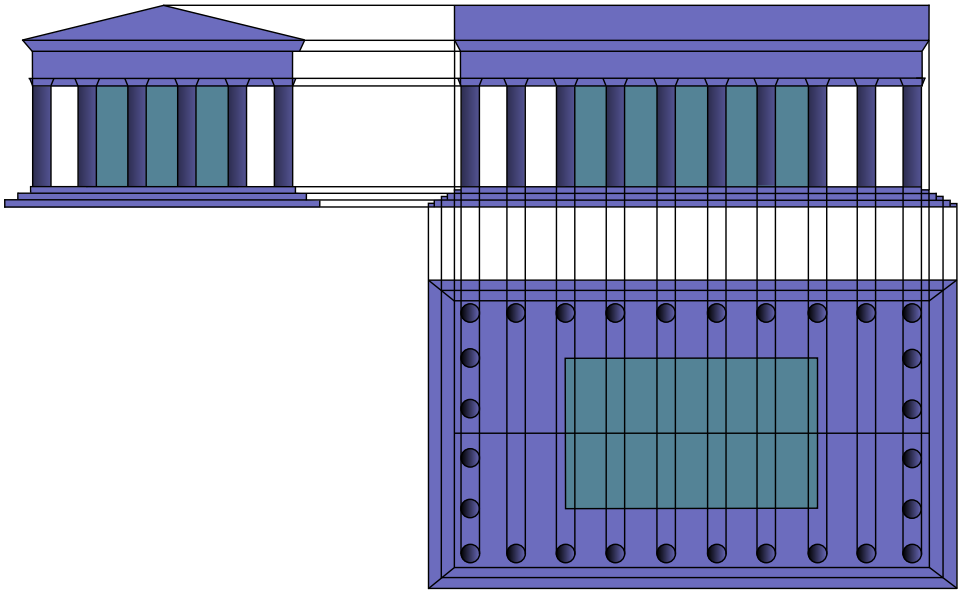


Fig. 2.13 A successful reconstruction of a building ends up in a consistent and accurate CAD plan.

The GML is based on concepts of Adobe Postscript and is designed to describe geometric shapes in 3D. It uses polygonal meshes and subdivision surfaces and provides data types such as numbers, strings, vertices, edges, faces, etc. as well as methods to create and modify these data types.

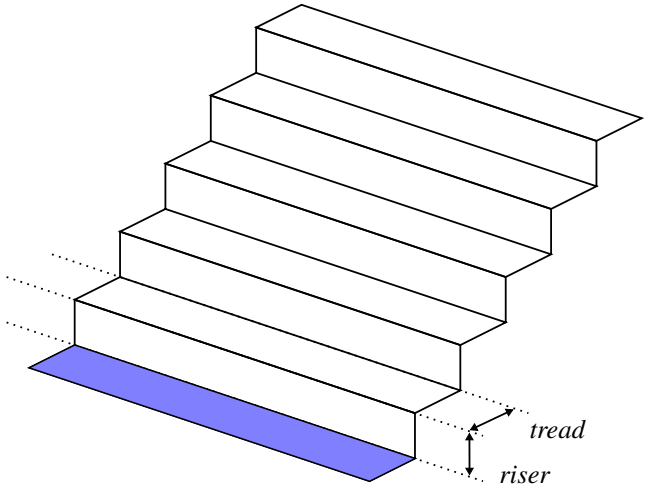
A commented, exemplary method call to create stairs is

```
% push base vertices of base polygon
% on the GML stack
[ (0,0,0) (10,0,0) (10,1,0) (0,1,0) ]

% push additional parameters on stack
0.2   1   5

% call library function to create stairs
Lib.stairs
```

The function to create stairs takes the vertices of a base polygon (and an implicit half-edge), the length of the riser, the length of tread, and the number of steps.



A simplified implementation of the `stairs` function (without any error or reasonability checks) is listed beneath:

```
% pop parameters from stack and use them by names
usereg
!nSteps !nTread !nRiser !polyBase

% create base polygon from array of vertices
:polyBase 3 poly2doubleface

% initialize first step
0 :nRiser 3 vector3 extrude
dup faceCW edgedirection length !nDepth

% create following steps with a for-loop
1 1 :nSteps
{
   pop edgemate
   0 :nRiser 3 vector3 extrude
   edgemate
   0 :nTread 3 vector3 extrude
   0 :nRiser 3 vector3 extrude
   edgemate
   0 :nDepth 3 vector3 extrude
   edgemate
} for

pop
```

Further information about the Generative Modeling Language, interpreters for various platforms and examples can be found at:

http://www.generative-modeling.org

Havemann shows that it is important to understand the original construction rules to derive a general shape template. He has studied the construction of buildings and windows of the

Gothic era to understand the development of the shapes. After that he was able to transfer this knowledge to the GML. Now all variations of Gothic windows are available and can be used in any GML scene [HF04].

Although powerful in the hands of experts, the GML is a complex programming language, which needs to be learnt even by computer scientists with common knowledge of algorithms. This is a significant hurdle for designers and other non-expert users who want to express their shape ideas in GML. Schinko et al. introduce an alternative for programming GML directly, a code translator [SSUF11]. It is included in the generative modeling framework called *Euclides*[8] [SST10]. It is a JavaScript translator allowing users to define shape descriptions in the easy-to-use scripting language. The code is then converted to one of the supported target platforms, for example GML or the XML-based documentation target. The meta-modeller concept of Euclides offers a more general idea of procedural modeling. Shapes can be translated to a target platform while keeping the generative representation. It gives non-expert users a valuable tool for procedural modeling.

The creation process can also be supported by some clever software tools and made available to an even larger group of users. The Castle Construction Kit by René Berndt is a tool built on top of GML [BGHF05]. It supports the user on creating Medieval castles and fortress systems by defining many building blocks. For example, castle walls are created automatically from a polyline which is created by clicking onto a floor surface. Parameters of the wall like height and shape variations are modified with a graphical user interface. Still the resulting content is stored as GML code. Changes and variations to the 3D model are easily implemented by experts and added to the editor. Also, as mentioned before, the algorithmic description is much smaller than the resulting content.

Many classical 3D modeling tools support some kind of scripting language to add algorithmic creation methods. For the production of the 2005 King Kong movie a digital version of New York City of 1933 was created. A large variation in buildings and facades was realized using the scripting language Mel in Autodesk Maya [Whi06]. Natural variations of shapes and compositions of shapes can be added with random parameter changes. This is done not only for the creation of cities but also for landscapes, plants and trees. Even for a large crowd of animated characters a similar approach can be used for the appearance as well as for variations in the animations. The results should be reproducible for example by using the same random seed or pseudo random numbers from a look up table. In this way it is possible to store such scenes in the procedural code also create them again and again with the same result.

2.5 Insight

Content creation is a large field with many variations in data structure, file formats and creation work flow. Depending on the intended purpose the resulting data may be unstructured lists of primitives or a detailed description of the creation process.

For non-scientific visualizations (e.g. marketing renderings) the content is appealing and looks plausible. But such data often lacks structure and only includes as much detail as is necessary for the specific rendering job. The creation is most likely done by an artist with the focus on creating the most appealing result in a restricted amount of time. The meaning of the shapes — the semantic information — seems to be not important and is rarely added manually to this kind of data.

[8] http://www.cgv.tugraz.at/euclides

On the other hand there is a large demand for very precise CAD data for digital engineering. Engineers aim to create an exact representation of 3D objects with the goal to actually construct them. The 3D data is reused for simulations and to further develop products. Many components are used for different resulting objects. Semantic information is very important for this kind of data.

3D content related to cultural heritage can be seen as a mixture of precise data (exact replica of existing artifacts) and also content looking plausible for not-existing reconstructed objects. Laser scanning and photo reconstruction are suitable methods for the digitalization of artifacts. For all the other visualizations a digital artist or some clever modeling tool is needing to create the content.

Semantic information is essential for digital artifacts in the field of archeology. The creation of semantics and its organization in the digital world are described in the next chapter.

Chapter 3
Semantic Markup for 3D Data

Abstract Additional knowledge is added to the previously created content. In this chapter the semantic markup for 3D data is described. Various kinds of semantics are defined and classified. Then a short overview of available standards for semantic data is given. The creation of semantics and its organization in the digital world are described. Two solutions for the sustainable linking of semantic information with 3D data are proposed.

The term *semantics* is used in many different ways in the field of linguistics, psychology etc. In computer science semantics are also used in the analysis of programming languages. And the encoding of shapes with vertices and faces needs semantics. Without semantic information the 3D data is nothing more than a bitstream. However, in this work the focus is on the *meaning of shapes*. Semantic markups define the meaning of a shape in the context of a scene. For example a box can be part of a wall or a door or a window. Geometrically its shape may be identical in those three cases but the extra semantic information is important for many applications. A clever visualization of a building switches walls transparent but leaves floors opaque.

The importance of semantic metadata becomes obvious in the context of electronic product data exchange and storage or, more generally, of digital libraries. For a heap of primitives without valuable metadata, it is hard, if not impossible, to realize the mandatory services required by a digital library such as markup, indexing, and retrieval.

In a very restricted and simple approach, this information may consist of attributes like title, creator, time of creation, and original place of the objects. But for many tasks, this is not enough.

For example, the Theseus Temple in Vienna consists of 28 Doric columns. To search for these types of columns in a large data set of scanned temples, the search engine needs to know that the temples consist of columns and what kind of columns they are. To allow navigation through the data sets (in 3D as well as on a semantic level), it is necessary to have a column's position within the data set of the temple. In analyzing the column, it might be necessary to find other artifacts of the same mason or the same period of time.

The following section gives a definition of the used terms and a classification for different kinds of semantics and metadata. Then standards and file formats for storing semantic information are discussed and the acquisition of semantic data is described. The last section is about the data organization of semantic data. The linking of semantic information with 3D data is proposed and how to publish the valuable combination of 3D content and semantic markup in a sustainable way.

3.1 Definition and Classification

Documentation standards and annotation processes are used in various fields of applications. Unfortunately, each branch of science has slightly different definitions of bibliographical terms. To clarify these terms and to avoid misunderstandings and misconceptions this section names the relevant definitions of terms used in this thesis. These definitions have been discussed at the 14th International Conference on Electronic Publishing (elPub) in *"Semantic Enrichment for 3D Documents Techniques and Open Problems"* by T. Ullrich, V. Settgast and R. Berndt.

A document is any object, "preserved or recorded, intended to represent, to reconstruct, or to demonstrate a physical or conceptual phenomenon". This definition has first been verbalized by Suzanne Briet in her manifesto on the nature of documentation: *Qu'est-ce que la documentation?* [Bri51]. In Michael K. Buckland's article "What is a document?" various document definitions are given and compared to each other [Buc97]. As we will concentrate on 3D data sets, the "physical or conceptual phenomenon" will always be a three-dimensional phenomenon.

A distinct, separate subpart of a document is called entity. Other authors refer to a subpart as segment. Metadata about documents or parts of documents are defined as "structured, encoded data that describe characteristics of information-bearing entities to aid in the identification, discovery, assessment, and management of the described entities." The American Library Association formalized this definition in its Task Force on Metadata Summary Report [CoC99]. According to this definition metadata is always structured. Unstructured, encoded data, such as comments and free texts, are hereinafter called annotations. As metadata is always structured, it can be specified in a formal, explicit way: an ontology is a "formal, explicit specification of a shared conceptualisation". It provides a shared vocabulary, which can be used to model a domain; i.e. the type of objects and/or concepts that exist, and their properties and relations. Tom Gruber established this definition in his article *A translation approach to portable ontology specifications* [Gru93]. The connections between a document and its metadata or annotations are called markup instructions. They provide local or global reference points in a document.

In the context of computer-aided design, reconstruction, and archival storage, a document is very often the result of a process chain. Data describing a single processing step or a document's process chain is termed paradata.

Metadata and annotations – semantic information in general – can be classified in several ways. Depending on the field of application, they can be classified according to the following criteria.

3.1.1 Document data type

Semantic information enriches a document. As documents can be grouped according to their type, these categories can be transfered to metadata and annotations as well. This work's focus lies on 3D data, which can be subdivided further into different kinds of 3D representations (also see Section 2.1).

Boundary Representation Several annotation systems allow users to leave text messages on the surface of a 3D model [JGD02] or to draw annotations on the surface and in free space of a virtual scene [OMS06]. The main field of applications is architectural model annotation. Adobe embeds 3D models into PDF documents [AS05] and combines this technique with its annotation system.

Point Clouds Although many tools exist (for example: Geomagic[1], Leica Cyclone[2]) to annotate and markup point clouds, the automatic documentation of the recording process has many gaps.

Volume Data This is the predominant acquisition technique in biomedical sciences, in which documentation, markup and annotation has always been put into practice. Consequently, many established annotation and markup systems exist; for example the volume data annotation tool called VANO by Peng et al. [PLM09] and the Annot3D Project by Balling et al. [Bal04].

Miscellaneous Besides these "main data types" numerous data representations specialized in its field of application are available. As annotation is a key activity of data analysis, many visualization systems offer annotation capabilities [LH94], [OF08].

3.1.2 Scale of Semantic Information

Semantic information can be added for the entire data set or only for a fragment of the object. For some metadata like "author" it can be sufficient to mark the entire document. But 3D data creation is often a collaborative task with many people working on one complex object. For comments and detailed descriptions a specific place within the 3D data set is needed. Also to communicate suggestions for improvement during an evaluation process it is necessary to make comments on some parts of an object but independent from the entities of the object. This can be done by defining an anchor, like a point, a surface or a volume in addition to the actual data set. It is essential that an anchor can be defined independently from the object. In this way it is also possible to annotate something which is missing in a specific place. Often also the viewers parameters are stored together with the added information to make it easier to read.

3.1.3 Semantic Level of Detail

The level of detail or the level of abstraction of the semantic data can be defined arbitrarily depending on the application. This level is not necessarily static: In the same way as 3D data can be organized in multiple levels of detail also the semantic data can have a similar structure. Especially for large data sets it is more convenient to work on such a hierarchical structure. For example a virtual factory consisting of many machines can include detailed information about the construction of each machine down to the screw. This information is not relevant while placing the machines as a single entity. On the other hand it is desirable to also keep the detailed information in the semantic data set.

3.1.4 Type of Semantic Information

The "Metadata Encoding & Transmission Standard"[3] defines the following types of metadata and annotation

[1] http://www.geomagic.com
[2] http://leica.loyola.com
[3] http://www.loc.gov/standards/mets/

Descriptive Information
Descriptive information describe the content of an object and comprehend amongst others the Dublin Core metadata set [SBW02].

Administrative Metadata
Administrative metadata provide information regarding how a document was created and stored, as well as intellectual property rights, information regarding the provenance of a document, transformation/conversion information, etc.

Structural Metadata
A structural map and structural links outline a hierarchical structure for a digital library object and embed a document into a context.

3.1.5 Type of creation

The process of semantic enrichment of 3D documents falls basically in two categories: manual or automatic. Most of the metadata (especially administrative and descriptive metadata) can be generated automatically, but depending on the domain certain fields need support from an expert. Especially categorizing 3d models can be a difficult task for automatic indexing, e.g. classify buildings according to architectural theory or genres (e.g. the Getty Art & Architecture Thesaurus (AAT)).

Annotations, in terms of free text, like comments and remarks, are usually entered manually (the translation of a comment using an automatic translation service would be an example for automatically created annotations). While the manual processing of structured metadata is done by experts, comments or remarks can also come from non-expert users. This method (social tagging) has become very popular within Web 2.0.

3.1.6 Data organization

The data organization is an important aspect thinking of the sustainability of the annotation. There are two basic concepts how programs can store annotations.

The truth is in the file
The metadata and annotations are stored within the original documents. EXIF or XMP are good examples for that strategy. The main drawback is that the file format must support the possibility to add such arbitrary data. While modern 3d formats like Collada offer this functionality (e.g. the extra tag), others do not. For these a sidecar file can be an appropriate location for storing the annotation. Putting the original document and the sidecar file(s) in a container is a very popular technique. In most cases the container is a simple zip archive, but with a different and unique extension (examples are the Open Document Format (ODF) or Microsoft Office Open XML).

The truth is in the database
In this concept the metadata and annotations are stored within a database system. This guarantees that the user will access up-to-date data, but he needs to retrieve the associated annotations separately. In addition, the application must be able to access the database.

3.1.7 Information comprehensiveness

Semantic enrichment can be further classified by the comprehensiveness of the information. The amount of comprehensiveness can vary from low to high in any gradation. An example for a low comprehensiveness would be the Dublin Core meta data set [Ini95]. It allows 15 properties to be added as semantic information. In contrast, for example the CIDOC Conceptual Reference Model (CRM) is a scheme with a high comprehensiveness. It is a framework for the definition of relationship networks of semantic information in the context of cultural heritage [Gro03]. CIDOC CRM offers a formal ontology with 90 object classes and 148 properties (in version 5.0.1) to describe all possible kinds of relations between objects.

3.2 Standards for Semantic Data

An important aspect of semantic enrichment is to agree on standards. For semantic information it is more a question of organizing documents in a standardized way. In the area of 3D data representations an important aspect is the file format and its ability to support semantic enrichment. Many concepts for encoding semantic information can be applied to 3D data but only a few 3D data formats support semantic markup.

The definition and storage of semantic information develops mainly around textual documents. Established standard schemes like Dublin Core and CIDOC CRM can also be used for 3D content. The Moving Picture Experts Group (MPEG) defined a standard for the annotation of media resources. MPEG-7 is a scheme to describe metadata for audio and visual contents. 3D models however are not part of the scheme. Bilsco et al. propose an extension to the standard for 3D data [BGVOM06].

Classical Metadata: Dublin Core

Different levels of semantic information exist. Commonly perceived as the most basic are the classical metadata. They are very similar in every public library in the world: *author*, *title*, *year of creation*, a unique ID, e.g., the *ISBN* number for books, and a few more. Many different metadata schemes have evolved over the centuries in the various countries. One of the major attempts for standardization is the "Dublin Core Metadata Initiative" (DCMI), or short *Dublin Core* (DC) [dub04], which defined in 1995 a list of 15 core fields for all bibliographic records. It was subsequently refined [dub04][Mil96].

Semantic Networks: CIDOC CRM

DC is quite useful, but has severe limitations. Most annoying is the lack of expressiveness with respect to relations. This leads to unacceptably "flat" knowledge: A field *author* is fine for a book record, but how can the knowledge be expressed that certain authors are relatives, in order to find all books from one family? One author may have been a student of another, a book was written during a certain period at a certain location. More complex, and much more important, are relations in cultural heritage. This is illustrated by the famous CIDOC/CRM example, the network of relations of the 1945 Yalta conference [Doe05]. It involves three allied

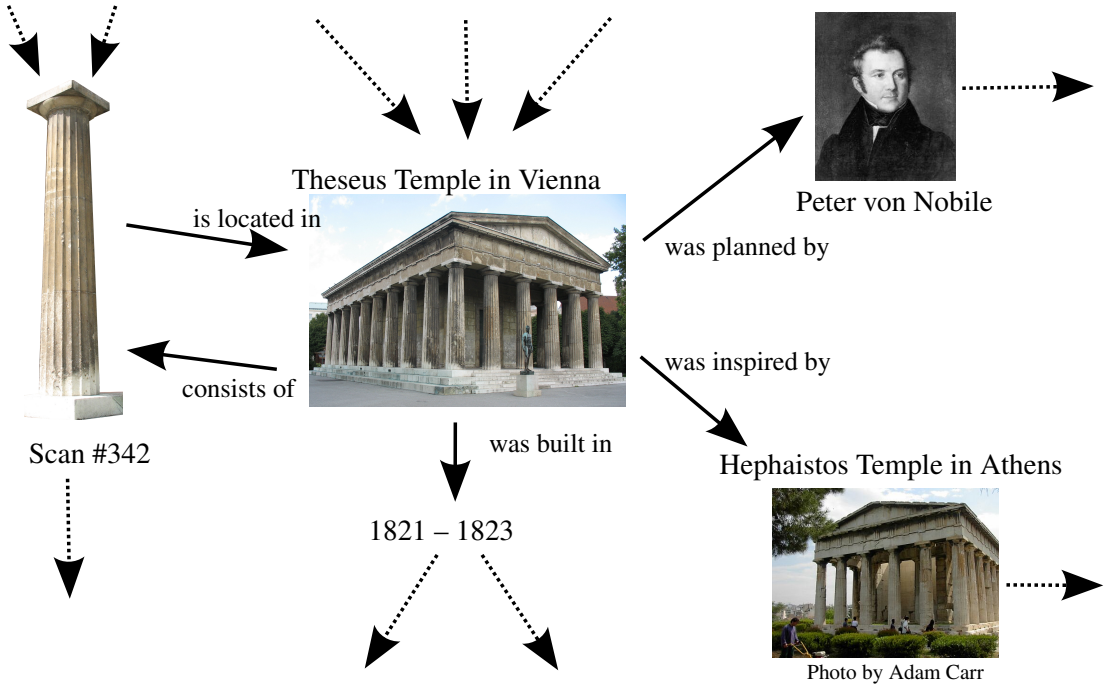


Fig. 3.1 A semantic network helps when working with different cultural heritage objects. In this example, the column to the left was scanned and then connected to the semantic network of which only a cutout is shown.

statesmen, the famous photograph of them, the jointly agreed document, a geographic location, a negotiation period, and the signing date.

In 2000, the International Committee for Documentation of the International Council of Museums released the conceptual reference model (CIDOC-CRM), a standard for representing such relational semantic networks in cultural heritage. The CRM specification 4.2 [CDG*05] provides 84 classes for the following entities:

- actor (person)
- place
- event
- time-span and
- man-made object

which may contain various information objects. For example man-made object has a sub-class information object that is further differentiated into image and text document etc.

It also defines 141 relations that include the following:

- participated in
- performed
- at some time within
- took place at
- is referred to by
- has created.

Figure 3.1 shows a simplified cutout of a semantic network.

3.2.1 3D Content

As described earlier in Section 2.3 many different file formats with a large variation of shape descriptions and features are in practical use. For 3D data there is not one single standard format like, for example, JPEG can be seen as standard for digital photos. And for semantic information on 3D data it is even harder to identify one or a few standard formats because most of the commonly used 3D formats do rarely support semantic enrichment.

On the one hand it is possible to extend existing 3D file formats to support metadata and annotations. Especially XML based formats are well suited to be extended while still being readable by existing applications. Some extensions have been proposed for Extensible 3D (X3D) and for Collada. On the other hand there are document formats which have been extended to support 3D content, like PDF 3D.

Collada and **X3D**

The Collada description and the X3D definition allow to store some metadata like title, author, revision etc. not only on a global scale but also for parts of the scene like nodes and geometry. Custom extensions to the format are possible as part of the XML scheme.
Niccolucci et al. have demonstrated the integration of 3D into semantic databases based on the X3D format using the MAD/SAD framework [ND06].
The Collada format can be found in Google Warehouse and the Google Earth application. Metadata like the location of the 3D data on the virtual earth however is stored in separate files. A suggestion for storing semantic data in Collada files is described in Section 3.4.1.

PDF 3D

PDF 3D allows to store annotations (logically) separated from the 3D data of the annotated object. Typically the exported 3D data is modified to get a smaller file size. But only if the model is stored without lossy compression it is possible to extract the original data and use the PDF 3D as an exchange format. The PRC format supports product manufacturing information and allows to use the geometry data as input for computer aided manufacturing. U3D is mainly used as a visualization and publication format.
Also the PDF format allows to add additional annotations to the model and even annotate the annotations. 3D PDF has the potential to become a standard for 3D models with annotations. The viewer application is widely spread and PDF documents are the quasi standard for textual documents.

The export of 3D content containing metadata or annotations to another file format often leads to information loss. To minimize data loss some semantics, for example labels and measurements, can be integrated into the 3D data as geometry. But afterwards, depending on the format, it may be hardly possible to distinguish between the metadata and the geometry.

Some software companies offer solutions for their own product lines. For examples Dassault Systems introduced an XML based format called 3D XML. It is supported by all of their applications as an exchange format. There are many of those proprietary solutions. For sustainable semantics an open standard is needed.

3.3 Acquisition of Semantic Information

As described in Section 2.4 the creation of 3D content is not a standardized process. Semantic data can be added at creation time for construction and planning, using a modeling tool and also with generative modeling. A special case is the creation by automatic processes, for example acquisition of cultural heritage objects. The geometry is created from measured points. Semantic data of higher order however cannot be captured automatically. It can be added via analyzing the created shapes or by hand. The Metadata Generator by M. Schröttner et al. is a supporting tool for the manual creation of metadata. It is part of the scalable repository infrastructure by X. Pan et al. [PSH12] in the 3D-COFORM project.

Manual annotation is time consuming - an automatic process is preferable if possible. Such a process is shown in the next section with examples in the field of cultural heritage. It was proposed by T. Ullrich, V. Settgast and D. W. Fellner in *"Semantic Fitting and Reconstruction"* [USF08].

3.3.1 Automatic extraction of Semantic Data

The work flow from archaeological discovery to scientific preparation demands multidisciplinary cooperation and interaction at various levels. Computer graphics have contributed essential tools to this work flow especially to the model acquisition pipeline providing scanning and photogrammetry methods, reconstruction algorithms, and visualization techniques.

The model acquisition pipeline starts measuring an object using 3D scanning technologies based on laser scanners, computer tomography, or photogrammetry. The measured data, usually represented as a point cloud, has to be re-engineered to receive a high-level geometry description of the model appropriate for various uses.

In the context of cultural heritage the utilization encompasses raw data collection, archival storage, reconstruction, data analysis, addition of metadata, and dissemination, and visualization as summarized by Settgast et al. [SUF07]. In all these tasks the problem of shape description has to be tackled. Up to now the computer graphics community has pursued several approaches for model descriptions, which can be grouped into two classes.

The first, much larger, class follows the composition-of-primitives approach. It describes three-dimensional objects and whole scenes as an agglomeration of elementary geometric objects: Points, triangles, NURBS-Patches, spheres, cubes, blended blobs, and many others.

The second class is based on procedural shape representations. It comprises a variety of different methods. All of them have a lot in common: They describe objects by functions, operators, and algorithms using shape grammars, L-systems, functional composition or shape programming languages.

In the composition-of-primitives approach all elements are specified and listed individually. Without further information there is not much difference between a Greek temple and a statue. In the foreword of Snyder's Book [Sny92] on generative modeling, James Kajiya points out that, it's all "a matter of positioning the control points in the right places". But the inner logic of an object should be reflected in its construction, and its description of the result should reflect this process.

The importance of further information and semantic metadata becomes obvious in the context of electronic product data exchange and storage or, more general, of digital libraries

[Fel01], [FHH07]. For the usual low-level primitive-based shape representations it is hard – if not impossible – to realize the mandatory services required from a digital library: markup, indexing, and retrieval [HSLF07].

The promising approach of procedural and generative modeling is a key to solve this problem [HF03]. Due to the naming of functions and algorithms as well as possible markup techniques, procedural model libraries are the perfect basis for digital library tasks. The advantages of procedural modeling arise from the generative approach and the fact that generative models normally have perfect shapes, which do not suffer from wear and tear effects. Therefore they represent an ideal object rather than a real one. The enrichment of measured data with an ideal description enhances the range of potential applications – not only in the field of cultural heritage. A nominal/actual value comparison may indicate wear and tear effects as well as changes in style.

This connection is a great challenge as pointed out in "Procedural methods for 3D reconstruction" [Arn06]. A first step towards a solution and an outline of further research questions will be presented in the following.

3.3.1.1 Reconstruction in the Context of Cultural Heritage

The creation of consistent and accurate model descriptions is known as reverse engineering [VMJ97] and comprehends fitting, approximation and numerical optimization techniques [BBCS99], [BKV*02].

To outline the coherences between existing reconstruction approaches it makes sense to classify them into three groups (see Figure 3.2) according to the model description capabilities and the need of a preceding model segmentation.

If the underlying model description is able to describe every three dimensional object in a consistent and integrative way without the need of an additional superstructure, the fitting process can be called *complete* fitting. Otherwise an object is described by several small parts whose orientation to each other is stored in a superstructure. This approach is a *subpart* fitting process.

Complete Fitting

In 1992 Hugues Hoppe et al. presented an algorithm that fits a polyhedral surface to an unorganized cloud of points [HDD*92]. The result is a polygonal mesh which describes the complete object. Further development of polygonal reconstruction has lead to algorithms whose output is guaranteed to be topologically correct and convergent to the original surface as the sampling density increases [ABK98]. Besides polygonal reconstruction algorithms [GJ02], approaches based on radial basis functions [CBC*01], constructive solid geometry [RvdH04], or subdivision surfaces [CWQ*04], [MMTP04], [CWQ*07], are able to describe three-dimensional objects in a unified way and are representatives of the complete fitting class.

Subpart Fitting with Segmentation

The subpart fitting approaches can be divided into two categories depending on whether the input data has to be segmented and partitioned or not.

The preprocessing step of segmentation uses feature extraction algorithms, which determine feature lines such as crease loops and junctions, or border lines [GWM01]. Mark Pauly et al.

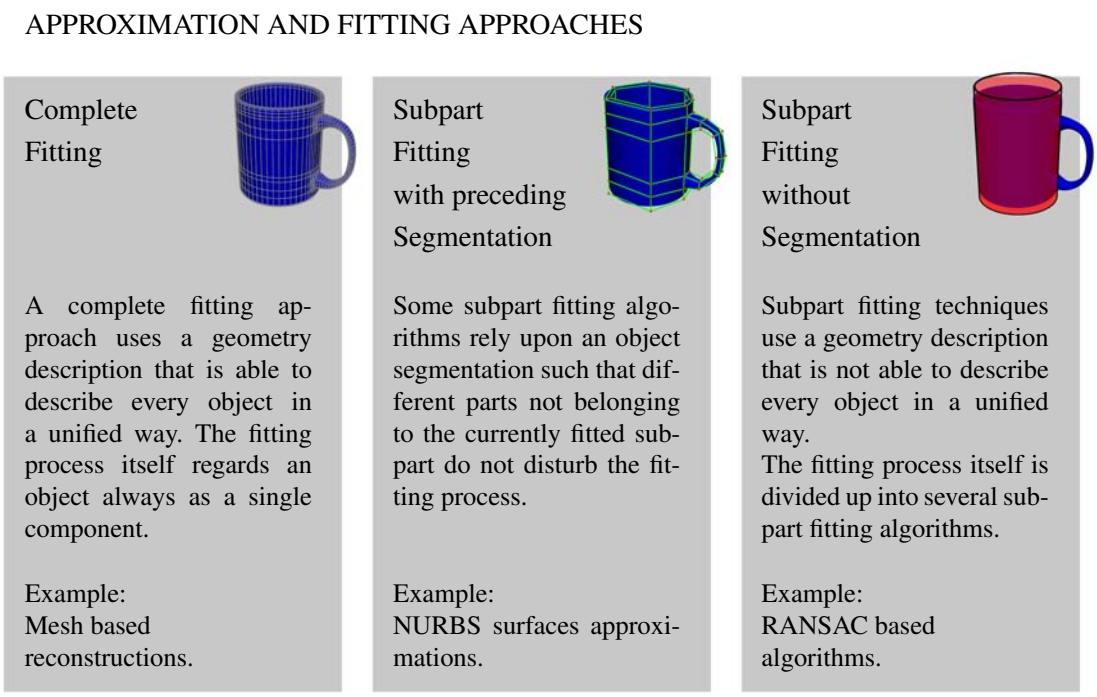


Fig. 3.2 The currently known solutions to reverse engineering can be grouped by the underlying geometric model description. The differences arise from the uniformity of a description and its need of a pre-segmentation.

used principal component analysis on local neighborhoods to classify points according to the likelihood that they belong to a feature [PKG03]. Using local feature vectors to segment and partition the input data into smaller regions, where each of it can be approximated by a single patch [GG04], is a common approach.

In combination with a special sequence of tests [PTK05] a large point cloud can be robustly split into smaller and smaller subregions until no further subdivision is sensible. The spectrum of partitioning approaches reaches from local feature extraction to global shape recognition [HOP*05], [VGSR04], and variational shape approximation [CSAD04], [WK05]. The detection of axes of reflection or intrinsic symmetries of a model [MSHS06], [MZWVG07] as well as geometrical classifications using line geometry [PWL01] offer further segmentation possibilities.

These analysis techniques are needed for example by reconstructions based on non-uniform rational b-splines (NURBS) [WPL04], developable surfaces [Pet04], or least squares techniques [Sha98].

Subpart Fitting without Segmentation

No preceding segmentation is needed among others by random sample consensus (RANSAC)-based algorithms [FB81], [DDSD03], [WGK05]. The basic idea of RANSAC methods is to compute the free parameters of a subpart from an adequate number of randomly selected samples. Then all sample points are tested against the proposed hypothesis; i. e. they are checked

whether they belong to the subpart or not. This process is repeated until a sufficiently broad consensus is achieved.

A method to create generative models from point clouds or range data of 3D objects has been presented by Ramamoorthi and Arvo [RA99]. The algorithm uses a hierarchy of generative model templates. The root node of the hierarchy is a very simple geometric shape. Each child node within the hierarchy is a refined version of its parent. For a given point cloud the algorithm starts to fit the root node to the input data. It determines the template's free parameters represented by spline curves which match the point cloud best. Afterwards it fits the node's children to the point cloud as well. The child node with the best results according to an error function is selected to be the new parent node whose children are fitted next. In this way the algorithm refines the generative model until the end of the hierarchy is reached or the fitting error is underneath a user-defined threshold.

This approach creates concise generative models from incomplete and sparse data, but it is not able to differentiate between objects on a semantic level. Two objects which have been fitted by the same hierarchy may differ in their free parameters which are spline curves. Therefore the hierarchy of rotating generalized cylinders may represent many different objects: a banana as well as a candle-holder or a coffee mug.

Cultural Heritage

In the field of computer-aided design and computer-aided engineering the problem of reconstruction is well studied. Unfortunately the context of cultural heritage distinguishes itself by model complexity, model size, and imperfection to such an extent most approaches cannot handle.

- Complexity: cultural heritage artifacts "represent a masterpiece of human creative genius"[4]. Hence, many cultural heritage artifacts have a high inherent complexity.
- Size: the UNESCO lists 745 cultural sites in over 130 states[4]. An archaeological excavation may have an extent on the scale of kilometers and miles with a richness of detail on the scale of millimeters or small fractions of an inch.
- Imperfection: cultural heritage artifacts are embedded into a context. Beside natural wear and tear effects many artifacts exhibit signs of preservation, restoration, and refurbishment.

These additional constraints have to be regarded by reconstruction algorithms in the context of cultural heritage.

Semantic Information for Digital Libraries

Without semantic information scanned objects are nothing more than a heap of primitives. As the scanning process itself is now available to a wide audience and as the amount of scanned data increases rapidly, the importance of semantic metadata becomes obvious; especially in the context of electronic data exchange and storage or, more general, of digital libraries. For a heap of primitives without valuable metadata, it is hard – if not impossible – to realize the mandatory services required by a digital library: markup, indexing, and retrieval.

In a very restricted and simple approach, this information may consist of attributes like title, creator/author, time of creation, and original place of the objects. But for many tasks this is

[4] The Criteria for selection to be included on the United Nations Educational, Scientific and Cultural Organization (UNESCO) World Heritage List – http://whc.unesco.org/en/criteria

not enough. In order to allow navigation through the data sets (in 3D as well as on a semantic level), it is required to have detailed information and semantic markup within the data set; i.e. a column of a temple has to be identified as being a column. Analyzing the column it might be necessary to find other artifacts, for example, of the same period of time. In general, relations between objects are an essential aspect when working with cultural heritage. The *Conceptual Reference Model* CIDOC-CRM meets these requirements.

The first step to support a user in defining these relations is the identification of artifacts respectively of shapes.

3.3.1.2 Shape Description

While the creation of geometric model descriptions is a problem which has been researched reasonably well, the shape and structure description problem is still an open-ended question.

Definitions and Examples

One solution to the shape description problem is based on dictionaries. A dictionary describes an object by its definition. Using the definition the object is embedded into a context and its main functions can be explained. Unfortunately, this approach is not suited for computers. Most definitions are based on previous knowledge. Looking up this previous knowledge leads to circular dependencies and recursions, which cannot be resolved by a computer.

A picture dictionary provides a non-recursive possibility to describe a shape. The main idea is to illustrate each word entry with photos or drawings. This *description by example* is widely-used e.g. in plant taxonomy and identification. The same approach is used in the field of computer vision to identify objects.

Feature Vectors and Metric Spaces

Chen et al. use a database of examples and calculate the similarity between a pair of 3D models by comparing heuristically chosen sets of 2D projections rendered from both models. Each projection is then described by image features such as silhouette. The similarity between two objects is then defined as the minimum of the sum of distances between all corresponding image pairs over all possible camera rotations [CTSO03]. Other feature-based similarity search techniques and an overview on “Content-based 3D Object Retrieval” can be found in an article by Bustos et al. [BKSS07].

Structural Descriptors

The majority of methods proposed for 3D description mainly focuses on geometric features. The main advantage of these methods arises from well-known techniques in image and geometry processing. A totally different concept uses structural descriptors. They agree with the general consensus that shapes are recognized and coded mentally in terms of relevant parts and their spatial configuration or structure [Bie87]. The main idea has been introduced by Hilaga et al. [HSKK01]. Structural descriptors describe an object using a graph that represents the shape’s structure, where nodes and edges represent geometric operations or attributes. A framework to extract structural descriptors is presented by Marini et al. [MSF07].

Procedural Descriptions

Structural descriptors are similar to shape grammars. For example Pascal Müller uses a hierarchy of successive refinements encoded in a shape grammar to model Roman housing architecture [MVUVG06]. Based on input data of geographical information systems (GIS) such as building footprints, street maps, etc. the shape and facades of buildings are generated by extruded footprints and a variety of successive, probabilistic derivation rules.

The presented system is capable to create an ancient city based on GIS data. The result is "statistically correct" with respect to population density maps and land usage maps and it shows a possible reconstruction, but it is not an exact reconstruction of some given geometry.

The good results of the system are based on extensive studies on Roman housing architecture. Applying knowledge to reverse engineering problems improves the recovery of object models [Fis02]: "computers are good at data analysis and fitting; humans are good at recognizing and classifying patterns." Robert B. Fisher demonstrated that general shape knowledge enables to recover an object even if the given input data is very noisy, sparse or incomplete.

Generative Modeling Language

Another possibility to encode procedural knowledge and generative shape descriptions is the *Generative Modeling Language* (GML) by Sven Havemann as described in Section 2.1.3 and Section 2.4.5.

The promising approach of procedural modeling can help to close the semantic gap due to the naming of functions and operator sequences. Therefore, procedural model libraries are the perfect basis for semantic enrichment.

The challenge of the generative modeling approach is its integration into the model acquisition work flow. The combination of procedural, semantic, regular descriptions used in generative modeling and multiresolution 3D data describing high-frequency, irregular, noisy parts of a model will offer a wide range of applications. The characterization of complete models as well as the indexing and identification of subparts using metadata is an essential requirement in many fields of applications.

The systematic composition of a model cannot be analyzed on a low level inspecting single triangles, but on its high-level structure. The possibility to have both descriptions always at hand is of vital importance in product data management, computer-aided manufacturing, computer-aided design, digital library services, etc.

A step towards an integrative model description is a reconstruction approach that can fit a generative modeling template to a given point cloud identifying a model's parameters.

3.3.1.3 Decreasing Exponential Fitting

The main idea of the *decreasing exponential fitting* approach is to introduce a weighting function ψ on top of the distance – similar to the least squares technique.

Least Squares Fitting

Squared distances to curves and surfaces do not only appear in geometric reconstruction problems, but also in registration tasks in computer vision and positioning problems in robotics. Due to the importance of the squared distance function, great effort has been made to understand

the geometry of the function, which associates to each point in space the square of the shortest distance to a given curve or surface [PH03]. In combination with an octree data structure, which stores in each of its cells a local quadratic approximant of the squared distance function of a geometric object, many geometric optimization problems – like registration or surface approximation, where the solution to the problem is found iteratively with a Newton-type method – can be solved very efficiently [LPZ03].

The metric based algorithm in Section 3.3.1.3 belongs to the category of subpart fitting algorithms without preceding segmentation. It is able to find the best fit of an arbitrary subpart within a point cloud. Regarding the used error functions and metrics involved the presented method is similar to least squares techniques. On this account, least squares techniques to fit a plane to a point cloud will be presented introductorily.

The sum of the squares of the distances between data points and a model is used to treat the residuals as a continuous differentiable quantity. In many cases a simplified version (e.g. using only a vertical offset) is used instead of the Euclidean distance allowing a much simpler, linear analytic form.

The nonlinear least squares fitting technique to fit a plane

$$\mathbf{E} : Ax + By + Cz + D = 0$$

to some points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ with $\mathbf{p}_i = (x_i, y_i, z_i)^T \in \mathbb{R}^3$ uses the unsigned Euclidean distance

$$d(\mathbf{p}_i, \mathbf{E}) = \left| \frac{A \cdot x_i + B \cdot y_i + C \cdot z_i + D}{\sqrt{A^2 + B^2 + C^2}} \right|. \tag{3.1}$$

The function to minimize consists of the squared distances

$$f_{LSQ}(\mathbf{E}) = \sum_{i=1}^{n} d^2(\mathbf{p}_i, \mathbf{E}) = \sum_{i=1}^{n} \frac{(A \cdot x_i + B \cdot y_i + C \cdot z_i + D)^2}{(A^2 + B^2 + C^2)} \stackrel{!}{=} \min_{A,B,C,D} \tag{3.2}$$

and takes the four model parameters with three degrees of freedom (the plane equation can be normalized).

The usage of squared distances is a mixed blessing. On the one hand it avoids square root operations. On the other hand outlying points may have a disproportionate effect on the fit. Even worse, if a model shall be fitted to a data set, least squares methods implicitly assume that the entire set of data can be interpreted by one parameter vector of the model. If this condition is not met (as illustrated in Figure 3.3), the data set has to be partitioned into regions fulfilling this condition.

Random Sample Consensus

The RANSAC approach mentioned above can handle multiple data sets and outliers very well. A basic prerequisite needed by any RANSAC approach is a solution for the inverse model description problem. A model's parameters have to be determined by a fixed number of points which define a unique model instance. This fixed number should be the minimum number of sample points needed to identify a unique solution, as the expected number of iterations needed by RANSAC is given by

$$N = \log(1-p)/\log(1-(1-\varepsilon)^s)$$

where p is the error probability, ε is the proportion of outliers and s is the number of needed samples [HZ04].

Revisiting the plane fitting example, an appropriate RANSAC algorithm has to calculate the plane parameters A, B, C, and D from three sample points, which is quite simple. This inverse problem becomes difficult or even unsolvable as soon as the model's complexity increases. To fit a cylinder, more points are needed and the calculation to get its position and radius directly is non-trivial. An overview of "Direct solutions for computing cylinders from minimal sets of 3D points" has been presented by [BF06].

Two major advantages of the RANSAC approach are its ability to ignore outliers without explicit handling and the fact that it can be extended to extract multiple instances in a data set [SWK07], [SWWK08].

Parameter Estimation

These advantages without the need to solve the inverse parameter problem also apply to the decreasing exponential fitting approach. It introduces a weighting function ψ on top of the distance.

A reasonable weighting function should preserve the properties of the metric used to compute the distance:

1. The weighting function ψ should be non-negative, so that outliers cannot compensate each other.
2. $\psi(0) = 0$ should be fulfilled, in order to identify the best-fit solution independently of the free parameters.

Considering the requirements for a numerical, iterative solver,

3. the weighting function should be monotonic increasing on the interval $(0, \infty)$ and monotonic decreasing on the interval $(-\infty, 0)$. Otherwise additional local minima complicate a numerical solution.
4. ψ should have two continuous derivatives ($\psi \in C^2$) to avoid an unnecessary reduction of the numerical tools at hand.

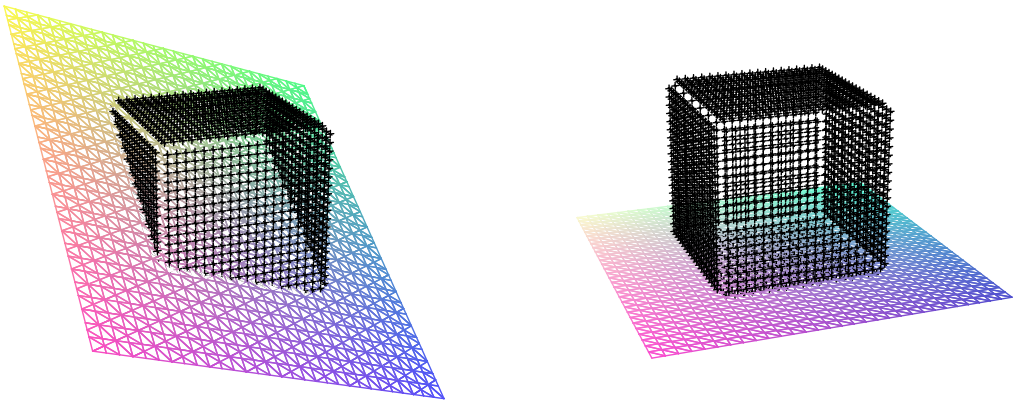


Fig. 3.3 Fitting a plane to a point cloud is a frequently needed task in reverse engineering. Using a least squares approach to approximate a point cloud of a regularly sampled cube by a plane leads to undesirable results (left). The solution to this problem either uses a segmentation algorithm to separate the six cube sides and to fit them separately or uses a fitting approach, which is robust towards outliers (right).

All these conditions apply to least squares fitting $\psi_{LSQ}(x) = x^2$. To overcome the sensitivity of the least squares approach, the area of statistics has developed a wide range of methods to circumvent the limitations of traditional parametric and non-parametric methods [MMY06]; in particular, generalized maximum likelihood estimators (M-estimators) [Zha97] and robust parameter estimation techniques [Ste99] are of big importance in this context.

The decreasing exponential approach uses a Gaussian weighting function

$$\psi_{EXP}(x) = 1 - e^{-x^2/\sigma^2}. \tag{3.3}$$

This function meets the conditions (1) to (4) mentioned previously. To overcome the disproportionate effect of outlying points the weighting function ψ_{EXP} has two important, additional properties.

5. The codomain of ψ_{EXP} is limited to the interval $[0,1)$. Due to the upper limitation $\psi_{EXP} < 1$, outlying points do not disturb the overall fitting. Outlying points have only limited influence (illustrated in Figure 3.4). But in contrast to segmentation and clustering techniques all points *have* influence and regard is paid to them.
6. While the idea behind least squares is based on regression analysis, the background of the decreasing exponential approach is related to combinatorial analysis. The combinatorial process to find a model containing the maximum number of points on its surface M can be formulated using the Kronecker delta:

$$\delta_{\mathbf{p}_i,\mathbf{M}} = \begin{cases} 0, \mathbf{p}_i \notin \mathbf{M}, \\ 1, \mathbf{p}_i \in \mathbf{M}. \end{cases}$$

The formulation of the discrete, combinatorial problem as a continuous maximization problem with the possibility of handling feasible noise leads to a weighting function

$$\psi_{COM}(x) := e^{-x^2/\sigma^2} = 1 - \psi_{EXP}(x).$$

Having transformed the maximization problem into a minimization one results in decreasing exponential fitting which maximizes the number of points on a model's surface.

Utilizing ψ_{EXP} the resulting objective function is

$$f_{EXP}(\mathbf{M}) = \sum_{i=1}^{n} \psi_{EXP}(d(\mathbf{p}_i,\mathbf{M})) = \sum_{i=1}^{n} 1 - e^{\frac{-1}{\sigma^2}\cdot d^2(\mathbf{p}_i,\mathbf{M})} \stackrel{!}{=} \min_{\mathbf{M}}. \tag{3.4}$$

The general, three-dimensional optimization problem to fit a model template to a point cloud using decreasing exponential fitting is given by

- some points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^3$,
- a model template $\mathbf{M}$ with d free, real valued parameters $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, (The parameters x_j may be restricted $x_j \in [x_j^{min}, x_j^{max}]$.),
- the Euclidean distance function $d(\mathbf{p}_i, \mathbf{M}(\mathbf{x}))$ measuring the distance from a point $\mathbf{p}_i$ to a specific instance $\mathbf{x}$ of the model $\mathbf{M}$, and
- the weight function $\psi = 1 - e^{-x^2/\sigma^2}$, $\sigma > 0$.

The optimization task is then formulated as a minimization problem

$$f(\mathbf{x}) = \sum_{i=1}^{n} \psi(d(\mathbf{p}_i, M(\mathbf{x}))) \stackrel{!}{=} \min_{\mathbf{x}}. \tag{3.5}$$

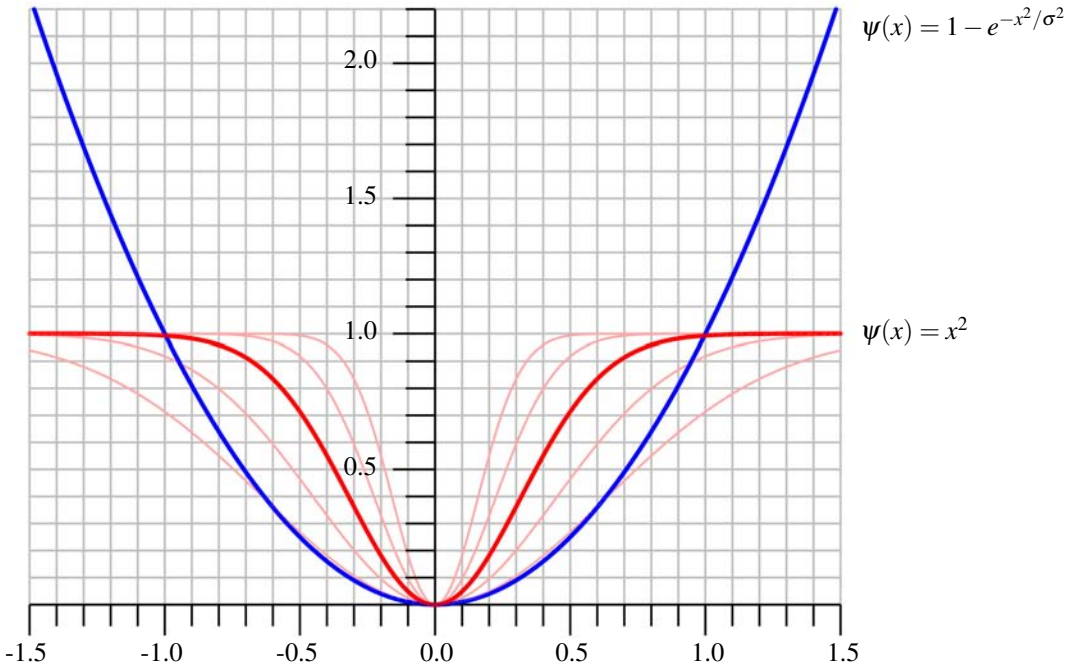


Fig. 3.4 The least squares approach minimizes the sum of squared distances, $\psi_{LSQ}(x) = x^2$, which weights outlying points (blue). The function $\psi_{EXP}(x) = 1 - e^{-x^2/\sigma^2}$ used by the decreasing exponential approach (red) reduces this effect significantly. The choice of σ depends on the noise level of the input data.

In practice σ should have a positive value. It should correspond to the noise level of the input data set. An adequate σ ensures that points whose distance to a model is $\pm\varepsilon$ only have a small contribution to the error function. The heuristic to select σ such that a point with distance d and noise ε will be weighted $\psi(d+\varepsilon) = 1/2$ yields good results.

Isometric Optimization and Model Optimization

In the context of point cloud processing two optimization tasks are of eminent importance: isometric optimization and model optimization. Both areas of application can profit from decreasing exponential techniques.

An isometric optimization determines the optimal isometric parameters (position and orientation to a point cloud) of a static 3D model. The free parameters, which have to be optimized, form a bijective map that preserves distances – an isometry. The parameters of the map form a six-dimensional vector and may be interpreted among other things as three Euler angles and a three-dimensional translation.

Beside static model fitting, decreasing exponential approaches can be used in the field of model optimization. In such a process it is not the position of a static model but its dynamic form, defined by free parameters, that is of interest.

Parametric objects can be fitted using only an Euclidean distance function d. If the overall fitting process uses a derivation free minimization algorithm, no even a derivation of the distance function is needed. Therefore, d does not have to be in closed form. Due to the pos-

sibility to evaluate the distance between a point and a specific instance of a parametric object algorithmically, arbitrary parametric object descriptions are feasible.

3.3.2 Examples

The presented fitting and shape recognition approach has been tested on three different point cloud data sets.

Theseus Temple

The first data set is a model of the Theseus Temple in Vienna, Austria. The temple has been built between 1820 and 1823 by Peter Nobile as a smaller imitation of the Theseion Temple in Athens, Greece.

The temple is a photogrammetrical reconstruction generated by the EPOCH 3D web service (http://www.arc3d.be) and MeshLab (http://meshlab.sourceforge.net). Approximately 100 photos were taken with a consumer camera in an arc around the temple. The resulting point cloud contains many holes as large parts of the surface were hardly visible in the photos (e.g. the floors). The data set consists of 2 million points.

To detect the columns, a row of vertical cylinders is used as the first generative model. The model's parameters are

- two center points (start and end) on the xy-plane,
- the cylinder radius, and
- the number of columns.

The second model template is a stairway built out of quadrangles. The free parameters are

- starting point,
- direction of the stairway,
- riser height, and tread length.

In order to reduce the complexity of the fitting process this shape template uses less parameters than the simple implementation presented in the GML sidebar.

This example demonstrates the ability to fit elementary geometric shapes like columns, approximated by cylinders, and generative descriptions such as stairs. The fitting works even for shapes which are only partly present in the data set: Parts of the columns surface and the stairways' treads are not in the reconstructed point cloud as they are not visible in enough photos.

The number of steps and the width of the treads are considered to be infinite. In Figure 3.5 these infinite structures have been cut manually along the bounding box of the relevant points near their surface.

This example reveals the problem of "over-fitting". A good configuration $\mathbf{x}_1$ for a cylinder of height h_1 with a small error value $f_1 := f_{EXP}(\mathbf{x}_1)$ can always be improved (according to the objective function) by a cylinder at the same position and with the same radius but with a height $h_2 > h_1$ using additional points from other parts of the model. The second cylinder will always have more points near its surface than $\mathbf{x}_1$, or at least the same number of points. Therefore, its error value will be smaller than or equal to f_1 which leads to over-fitted columns.

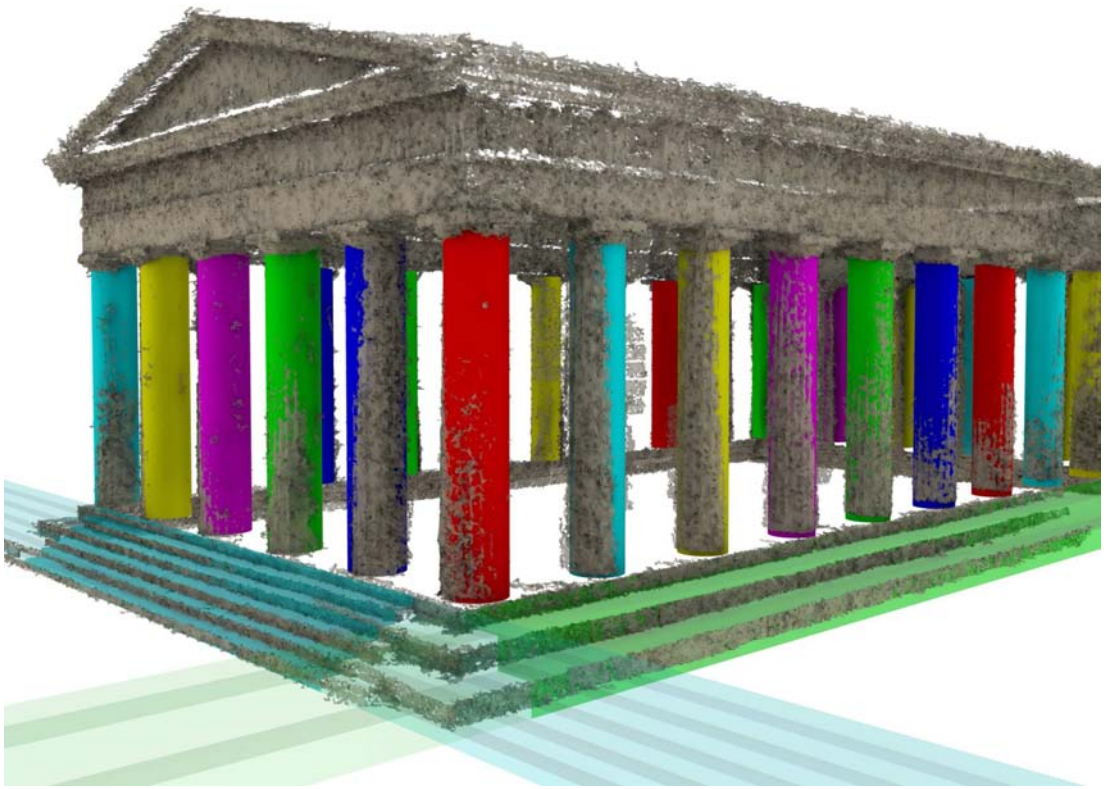


Fig. 3.5 The presented algorithm is able to fit generative descriptions such as columns and stairways to the Theseus Temple point cloud. Even structures, which are not present in the input data set (e.g. the treads), are realized as they are part of the generative description.

Landhaus

The second data set is a photogrammetrical reconstruction of an inner courtyard. It is located in Graz, Austria, and belongs to the Landhaus. This Renaissance building has been constructed in 1557 by Domenico dell'Allio.

In this examples we tried to compensate the over-fitting effect of columns by adding top and ground geometry. For example, if the shape description does not only contain a column, but also some parts of the ground the column stands on, then an over-fitted column does not have a better rating per se, as the ground points will not belong to the over-fitted template any more.

The generative model to fit describes an arcade. It consists of columns with a quadratic profile. Its parameters are

- the center point (x, y, z) of the first column,
- the angle α to define the arcade's orientation,
- the column width w,
- the column height h,
- the distance d between two columns, and
- the number of the columns n.

Figure 3.6 illustrates an instance of the procedural description in horizontal projection.

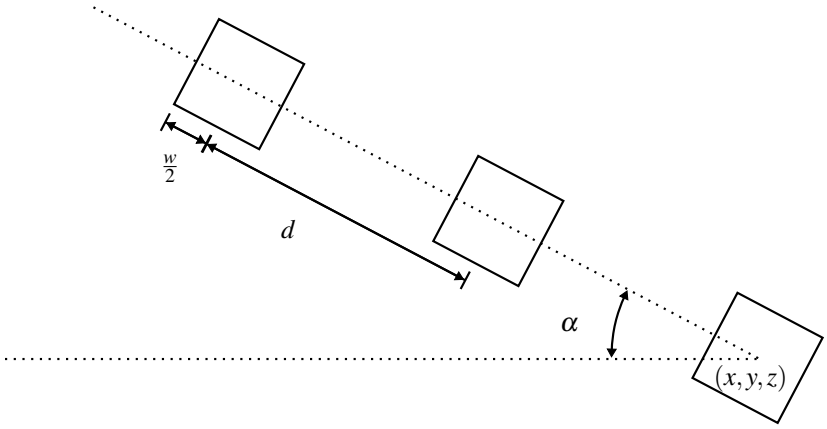


Fig. 3.6 The parametric description of the row-arcades model takes eight parameters: the three coordinates of the starting point (x,y,z), an offset angle α to define the arcade's orientation, a column's width w and height h, the distance d between two columns, and the number of columns n.

The results of the parameter extraction are shown in Figure 3.7. The floors and arcs should have compensated the over-fitting of the columns. While it worked good for the column's base, it failed for its height. The reason for this failure is mainly the poor quality of the data set. The point cloud shown in Figure 3.7 has been reconstructed with only 4 sequences of photos. The result has a very high level of noise. We started the fitting process with $\sigma^2 = 0.025\frac{1}{m^2}$; i.e. points up to $\pm 0.13m$ away from the surface are still regarded as part of the surface. As there are lots of noisy points from the wall above each arc, the algorithm determines the column height so that each arc crosses and intersects this wall. A data set with less noise could be processed with a smaller σ which would lead to better results.

Pisa Cathedral

The third example uses a point cloud with almost no noise. The data set of the Pisa Cathedral has been generated by the Visual Computing Laboratory at the Institute of Information Science and Technologies (ISTI) of the Italian National Research Council (CNR).

The Duomo is located on the Campo dei Miracoli in the center of Pisa, Italy. The architect Buscheto begun his masterpiece in 1064 and started the characteristic Pisan Romanesque style in architecture. Just like the whole building, the apse of the Duomo consists of many similar columns which are arranged in arcs and rows.

In this example the shape template describes an arcade that is arranged in an arc. It takes nine parameters and its horizontal and upright projection is shown in Figure 3.8.

Two fitting processes have been started to detect the two arcades in the data set. Their results are visualized in Figure 3.9. Due to the low level of noise within the data set, σ^2 has been set to a rather small value. In this case the strategy to avoid over-fitting works. If the columns were higher, all points of the arc would not belong to the arc template any more and only a few additional points (located at the wall) would then belong to the shape description.

A closer look at statistical details of a fitting process gives an impression of the algorithm's performance and exposes some technical aspects. The shape recognition of the circle-arcade model, which returned the parameters of the upper arcade in the laser scan data set (green visualization in Figure 3.9) has been started with 444.991 points.

The shape template has nine free parameters: the center point x, y, z, the main radius R, the column radius r, the offset angle α, the opening angle β, the number of columns n and

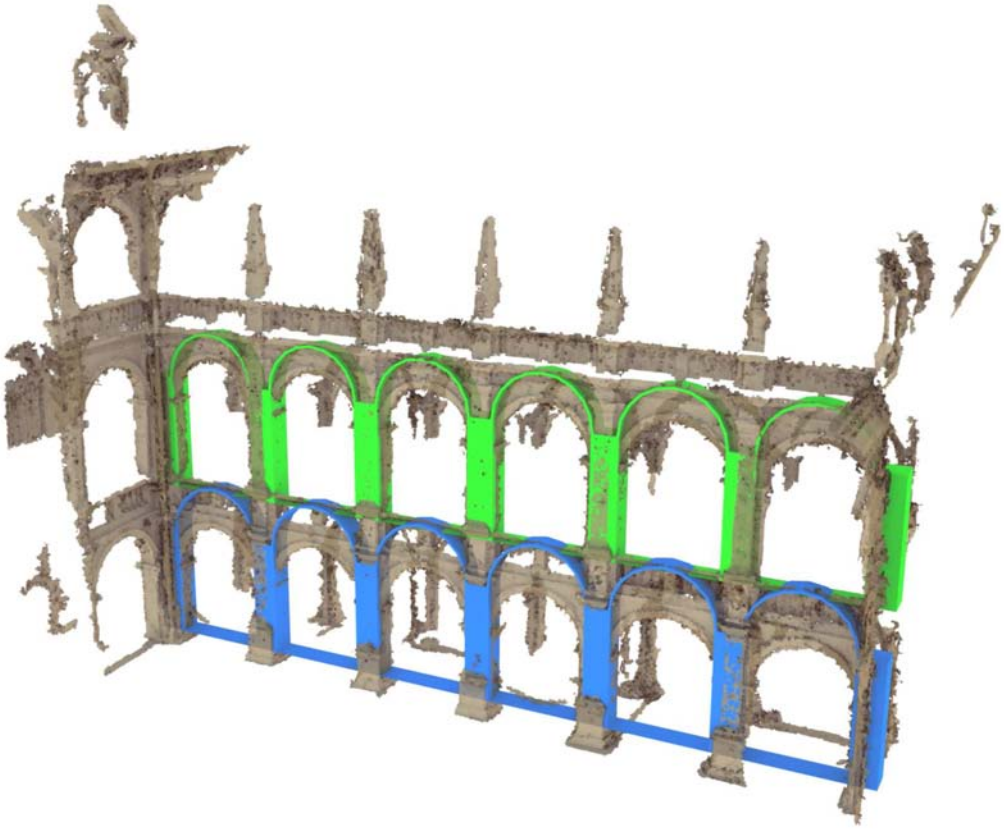


Fig. 3.7 The inner courtyard of the Landhaus in Graz consists of several arcades. Its point cloud model has a very high level of noise. A noisy data set requires a tolerant fitting configuration (large value of σ^2). As a result the algorithm interprets the points above an arc as points belonging to an arc.

the column height h. The fitting routine requires a bounded parameter domain. Therefore, the parameter intervals have roughly been estimated. The range δ of each parameter has been:

$$\begin{array}{rr|rr|rr} \Delta x = & 21.0m & \Delta y = & 21.0m & \Delta z = & 21.0m \\ \Delta R = & 1.5m & \Delta r = & 0.2m & \Delta n = & 0 \\ \Delta \alpha = & 14^\circ & \Delta \beta = & 14^\circ & \Delta h = & 2.0m \end{array}$$

Having set the parameter bounds/ranges the algorithm does not need any user interaction. It is able to handle an initial rough guess and converges to the global minimum with a precision in the scale of centimeters. Unfortunately, the number of columns had to be set manually. As the generative description does not check self-intersections or reasonability, a high number n of columns generates a "wall" of columns. In order to exclude such degenerated solutions, the number of columns has been fix. Consequently, the generative description had only eight free parameters.

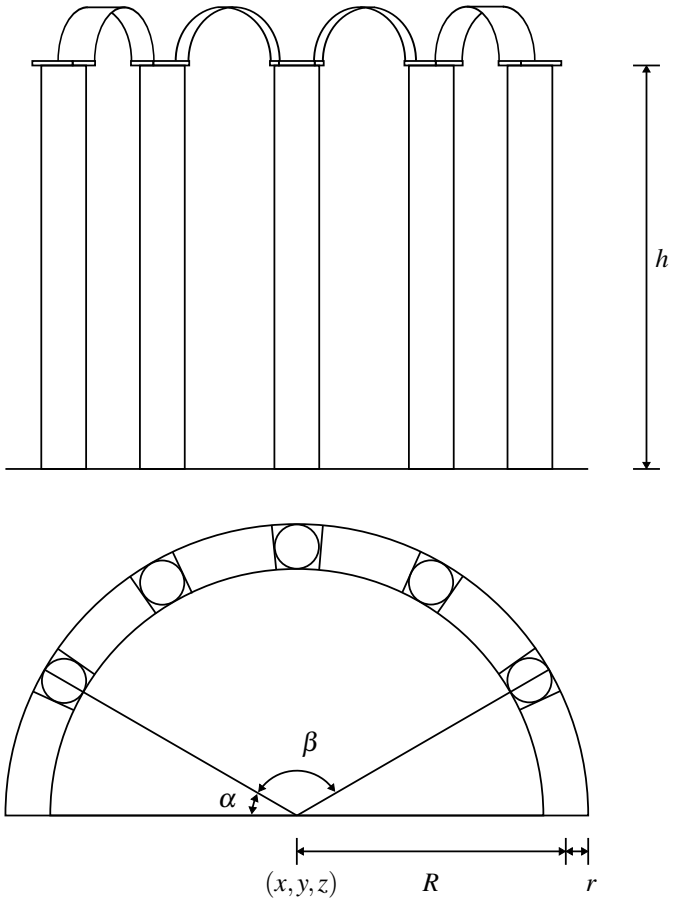


Fig. 3.8 The parametric description of the circle-arcades model takes nine parameters: the three coordinates of a center point (x, y, z), a main radius R, a column radius r, an offset angle α, an opening angle β, and the number of columns n. These values define the ground construction in the xy plane which contains the center point. The last parameter h defines the columns' height. The height of the Roman arcs are determined by the column distances.

3.3.3 Results

This section presents a numerical optimization technique for reverse engineering using generative model descriptions. The algorithm is able to fit model templates to point clouds. By fitting a shape description to a set of points it is possible to transfer semantic metadata from the model template. In this way the point cloud can be segmented and classified according to the meaning of its parts.

The examples show that the approach works. Unfortunately over-fitting is a serious problem and may lead to unwanted results. Compensating this effect using additional shapes has improved the results for the laser scanned data set. For point clouds with high level of noise this strategy is not sufficient. Further investigation of the over-fitting effect will lead to better results in the future.

Fig. 3.9 A generative model with nine parameters is fitted to the laser scan of the Duomo of Pisa. The algorithm detects two instances of the template and determines the parameters which describe the given geometry best. These characteristics (column height, etc.) are valuable information which are needed in the context of digital libraries in order to index a model repository.

The fitting process does not need a preceding point cloud segmentation and, once some parameter boundaries have been set, it is able to run automatically without any user interaction.

Our fitting algorithm only uses a small interface, namely a point-to-surface distance function for the shape template. Therefore various kinds of generative shape descriptions can be used as long as an Euclidean distance function can be provided. Bindings to other different scripting engines (e.g. Python bindings to Maya or Blender) may offer interesting possibilities and will be object of future research.

As only already existing model templates can be used, a library with "appropriate model descriptions" is needed. In some areas (e.g. architecture) the inner structure of a model is analyzed even though not described in a modeling language. Some artifacts (e.g. statues) elude a systematical description.

Although semantic information enables a simple search within 3D data, the problem of formulating an appropriate/advanced query is unsolved. Up to now it is only possible to search within textual semantic data. It is impossible to search for example for a temple just on a basis of some photographs.

More information about automatic fitting and ongoing research in the field of reconstructive geometry are described in the dissertation of Torsten Ullrich [Ull11].

3.4 Two Suggestions for the Data Linking

In this section I describe simple solutions to a seemingly simple, but in fact quite complex problem: To attach semantic information to 3D objects in a sustainable way. The complexity of the problem is a consequence of the generality of the required solution: There are numerous ways of representing 3D objects, and there are numerous ways to attach information. The sustainability requirement implies the use of well-established standards, and simplicity is a must in order to make it very easy for other applications to support and implement our proposed attachment technology.

Two possible solutions for attaching semantic information to 3D objects in a sustainable way are proposed. Their main usage is meant to be in the field of cultural heritage but is not restricted to that field. One solution is based on the Collada file format and the other one uses PDF documents.

3.4.1 Extended Collada

The first suggestion is published in the paper *"Sustainable Markup and Annotation of 3D Geometry"* written by R. Berndt, S. Havemann, V. Settgast and D. W. Fellner.

The original motivation for this approach came from a very well-made digital presentation of cultural 3D objects, the *Arrigo Showcase* from our colleagues around Paolo Cignoni and Roberto Scopigno from ISTI-CNR in Pisa [CPCS06],[CPCS08]. It represents a prototype case of cultural information visualization to a public audience today. It uses the *VirtualInspector* library for interactive display of massive multi-resolution meshes. The software is shown in a 3D-kiosk located next to the real statues, so that museum visitors can at the same time appreciate the beauty of the real, and enjoy additional information attached to the virtual statues (see Figure 3.10).

The truth, however, is that the Arrigo Showcase, how beautifully it may be made, is a dead end in terms of knowledge management: To really look good, the information pane in Figure 3.10 (right) is a bitmap created by a web designer. The point is, it is only stored as a bitmap on the Arrigo DVD. Fortunately, the original text was eventually found by our colleagues; but even the text is not very helpful in terms of sustainable knowledge management.

Sustainability can be achieved only by converting the facts in the text to some form of semantic format, amenable to automatic processing. One such format, developed for the cultural sector, is CIDOC-CRM (see Section 3.2).

Using the *Resource Description Framework* (RDF), CIDOC-CRM facts can be encoded as RDF-triplets (subject-predicate-object, formally entity-property-entity), in which case each entity has a type and refers unambiguously to an object. This unambiguous reference is simply encoded as *Uniform Resource Identifier* (URI), basically a uniform resource locator (URL). It is supposed to be a persistent identifier, a unique ID, that may, but does not have to, point to an existing web resource. In that sense the URI represents a sustainable piece of knowledge. This reduces the problem of attaching information to the problem of attaching a URI to a 3D model.

Fig. 3.10 The original Arrigo Showcase from ISTI-CNR: Museum visitors can interactively navigate (rotate, pan, zoom) a 3D model and obtain additional information by clicking on information links embedded with the 3D model.

3.4.1.1 Attaching URLs to 3D Objects

Attaching a URL to a (part of) a 3D file in general is a difficult problem, as was noted already by a number of researchers, just to mention Jarek Rossignac [Ros97]. One of the most annoying problems is the 3D file format problem: Hundreds of competing 3D formats exist, each with its pros and cons, and with its own community whose members usually do not care about other communities. This leads to fragmentation and incompatibility: Painful expierence teaches that almost any 3D format conversion inevitably implies **loss of information**. This leads to much friction and avoidable cost especially in the professional *digital content creation* (DCC) industry.

Since our concern is knowledge exchange, we chose Collada as the main format. It has a number of quite desirable properties for our purposes:

- **Clear, simple structure**: A Collada file has a <library> part, where the content nodes (models, materials etc) are defined, and a <scene> part with the scene hierarchy and transformations, which typically references library nodes by relative (intra-file) links
- **Relative anchors and links**: Library nodes can automatically be referenced using relative links and, thus, there is a way of addressing them from external resources using the #-syntax
- **No embedded BLOBs**: Collada files may not contain binary content, thus, there is a clear separation between structure and content; no massive datasets within Collada
- **Reference to external files**: Massive datasets can be referenced as external geometry; the file name is in fact a URL, which permits to include remote datasets transparently

- **Sustainable annotations**: Almost any type of node can have an <extra> child node attached that may contain arbitrary information encoded in XML. An application may claim itself Collada compliant only if saving a file leaves the <extra> information of a loaded object untouched.
- **Wide-spread use**: Collada is the 3D format of Google Earth and Google Warehouse, which offers masses of 3D content for free, and with Google Sketchup an authoring application for non-expert users is also freely available.

As pure ASCII format Collada is useful for encoding the low-poly models for games, or the clean CAD-like Sketchup models, but certainly not for massive scanned datasets with billions of triangles. Compressed triangle mesh formats typically need only 2-5 bytes per vertex (topology *and* geometry), much less than, e.g., compressed Collada ASCII. Furthermore, in cultural heritage certain custom shape representations are used which integrate, e.g., information for image-based rendering (for example Bidirectional Texturing Functions and Polynomial Texture Maps), for which special file formats exist. So the solution is to use Collada merely as a scene structuring format (we call it "*Collada light*") with references to external *geometry files* containing 3D data in one of several well-documented supported formats.

The great advantage is that since the annotations are stored in the Collada file, the original 3D datasets remain untouched. Furthermore, the same objects when stored, e.g., on a web server, can appear in different Collada files; and possibly also with different annotations. This is consistent with the view that semantic information is, in some way, always an interpretation. It is even possible that researchers exchange such lightweight Collada files via e-mail, for instance to discuss different interpretations of an artifact. When the file is opened, the scanned dataset is retrieved from a central server, if requested even in a reduced resolution.

3.4.1.2 An easy-to-use practical solution

Now the question is: Which types of annotations are possible, and how can researchers or curators create these annotations easily?

Shape markup does not always have to be carried out manually. One could also employ automatic feature detection algorithms that infer, by statistical evidence, certain types of semantic information. Algorithms exist for wall detection, for detection of symmetries, for detecting part-of relations (instantiation) and similarities between shapes, for primitive decomposition (sphere, box, cylinder etc.), as well as for surface properties such as curvature, roughness, and the like. All these algorithms were published in scientific papers (e.g., by Schnabel et al. [SWWK08]), but no method exists to reliably and sustainably store, combine, and explore their results. Our proposition is to create easy-to-use interfaces for these algorithms, and to store their results as semantic attachments in Collada.

This peek into the future already indicates that a flexible type of software infrastructure is needed, that is very versatile and proves useful in a whole range of different applications. We propose to use a component architecture, in our case Microsoft's ActiveX, to create a very simple widget that merely consists of a rectangular OpenGL drawing canvas. This ActiveX control, called *ActiveEpochViewer*, may be embedded into applications that can be created using easy-to-learn languages like C# or VisualBasic that are useful for GUI design.

The ActiveX control contains the scene graph engine OpenSG (see Section 4.1.2.1), that fosters sustainability by being well supported and extensible: When a new 3D format is to be included, it is sufficient to create a loader and new type of OpenSG geometry node for rendering. On the Collada level, the 3D format is completely transparent, since it is just a URL.

The last missing ingredients are the communication between ActiveX control and application, and a method for rich 3D interaction (behaviour and events). In our case, both are handled by the *Generative Modeling Language* (GML) [Hav05]: When a GUI button is pressed, VisualBasic (for example) sends a pre-defined string with GML commands to the ActiveEpochViewer, to reposition the camera, to switch to attachment mode, or to load a file etc. It can, however, also attach a GML-callback to a 3D object. This callback is to be executed, e.g., when the object is clicked. Since GML is a full scripting language that has access to the whole scene graph, arbitrarily complex actions can be issued this way, leading to rich 3D interaction.

We have realized two example applications based on this method, one is an **authoring application** for creating and editing shape markup, the other is a **presentation environment** to explore the resulting semantically enriched shapes.

3.4.1.3 Related Work

To the best of our knowledge, our work is novel in that no previous work exists which focused on solving the general problem of sustainable markup and annotation of 3D geometry. However, we are of course not the first ones to attach information to 3D objects. Numerous projects exist where, like in the Arrigo Showcase, custom and ad-hoc solutions for 3D-annotation were invented. And numerous proprietary solutions also exist in the context of *product lifecycle management* in the automotive, aerospace, and building industries, where 3D objects can also be versioned on a sub-part level, and every change is documented pedantically for a long time with author and clearing date.

But also in these domains, the awareness increases that a sustainable knowledge management is still missing. With liability periods of 30 years and more, manufacturers need reliable standards for vendor-independent markup of 3D geometry, to become robust against software version changes. On the European level for instance, the MACE project with its upcoming conference look into sustainable metadata standards for architecture. Our focus, however, is to devise a mechanism how to attach information to 3D objects in general, independent of a particular domain.

One inspiration for our work is the *Digital Photo Library* of the *Kunsthistorisches Institut in Florenz*, freely accessible under (khi.fotothek.org), where researchers can browse historic images. When they mark a spot on an image, the result is a URL that they can send to a fellow researcher, for instance to make him look at a statue on the facade of the dome in Florence:
http://digilib2.gwdg.de/...&mk=0.1179/0.5021&pt=0

Links can be attached to 3D objects in the VRML/X3D file formats since a long time. However, links are attached only on a per-object basis, and there is no standard way of attaching mark-up geometry as an annotation to any node. And we found Collada easier to use for our purposes than X3D, especially in the "light" version with external content formats. – One attempt to attach semantic information to X3D files was made by Franco Niccolucci and Andrea D'Andrea [ND06]. However, the linking was not bi-directional, and focus was the object ontology.

We would like to note that there is, of course, a standard XML technology for bi-directional linking between 3D objects and semantic information, namely XPointer / XPath / XLink. However, simplicity is an asset, and since our Collada light-files only have the duty of storing the scene hierarchy and semantic annotations, we did not see the point in treating each link as a separate entity of its own. Furthermore, to address objects on a sub-object level, we simply need a geometric target for a link in the file, which is exactly what our markup geometry is.

Our work can maybe best be understood in the context of an article about seven open research problems with 3D objects today [HF07]:

- Classify all shape representations in CG
- A sustainable 3D file format
- Generic, stable, and detailed 3D markup
- Generic 3D query operations
- Paradata: Processing history, provenance
- Close the semantic gap: Meaning of shape
- Maintain consistent relation between shape and meaning

This section can be understood as a first solution to the third problem.

3.4.1.4 Information Linking

This section will describe the first technical ingredient of our system, the annotation format used with the Collada <extra> tag, that is then evaluated in OpenSG. OpenSG is a powerful and extensible realtime graphics system and scene graph engine, capable of displaying many kinds of 3D object representations. It also offers the possibility of attaching arbitrary data to scene graph elements. We use this technique to store additional information directly within the scene. The annotations are attached as text strings that contain unmodified XML code.

The additional information is represented as a combination of URLs, a markup geometry and a camera definition. The concept of URLs allows to stick any data to an annotation as long as a URL exists to find it. A single markup can contain any number of URLs. For the first version of this annotation concept a simple sphere shape is used as markup geometry. Other shapes are possible, but the simple sphere definition is one of the fundamental ways to mark a spot in 3D. More complex annotation geometry will be introduced after we have made some experience with spherical annotations.

Note that annotations are very often view dependent. Automatic ways of setting the camera transformation exist, but in general it is more reliable to let a human set up the point of view. Therefore at least one camera transformation is added to the data set, per default, this is the camera set up at the time of annotation.

Format Specification

The following format specification for <annotation> can be part of any <extra> element, which belongs to a <node> within a Collada file. Each <node> element can contain multiple <annotation> tags in the <extra> part.

The <annotation> element represents one annotation attached to the <extra> element of a node within the scene. The annotation defines a geometry, one or more camera definitions, and one or more information links.

Attributes

id	xs:ID	A text string value containing the subidentifier of this element. This value must be unique within the scope of the parent element. Required.
title	xs:string	A text string containing the title of the annotation. Optional.

Child Elements

Name	Description (Occurrences)
<annotation_geometry>	Geometry defining the bounding of the annotation. (1)
<annotation_camera>	Camera position (0 or more)
<annotation_url>	Unique identifier to the additional data (1 or more)

The <annotation_geometry> defines the position of the annotation within the scene. At the current implementation level the geometry can consist of one or more spheres, but arbitrary other geometry (like oriented boxes or free-form shapes) would be possible in the same way.

Child Elements

Name	Description (Occurrences)
<annotation_sphere>	Geometry defining the bounding of the annotation. (1 or more)

The <annotation_camera> element defines a camera for the parent <annotation>. It is used to move the camera to a reasonable position when viewing the annotation.

Child Elements

Name	Description (Occurrences)
<from>	A point in space to look from. (1)
<at>	The camera target position (1)
<up>	The up-vector of the camera. (1)

Unique identifier (URL) to the additional data are stored in <annotation_url>.

title	xs:string	A text string containing the title of the information source. Optional.

<annotation_sphere> describes a bounding sphere, which is used to define the geometric reference of the annotation within the current node.

Child Elements

Name	Description (Occurrences)
	The center of the sphere. (1)
<radius>	The radius of the sphere (1)

An example of a Collada attachment where these nodes are used is shown in Figure 3.11.

Epoch-Viewer Framework

The core of the Epoch-Viewer framework is a reusable component, which combines OpenSG with the Generative Modeling Language (GML) [GBHF05]. Most of the functionality of the OpenSG API was exposed to the GML language. Since the first version the list of available operators has been continuously extended, e.g., support for reading the Collada scene description and scripted XML code manipulation was added.

```
...
  <annotation id="AnchorId" title="Additional Information">
    <annotation_geometry>
        <annotation_sphere>
          <center>12.9 33.94 3.12</center>
          <radius>3.84</radius>
        </annotation_sphere>
    </annotation_geometry>
    <annotation_camera>
        <from>44.8 34.8 28.58</from>
        <at>7.5 35.6 -1.64</at>
        <up>0.02 0.99 0.05</up>
    </annotation_camera>
    <annotation_url title="Collada">
       http://www.khronos.org/collada/
    </annotation_url>
    <annotation_url title="OpenSG">
       http://opensg.vrsource.org/trac
    </annotation_url>
  </annotation
...
```

Fig. 3.11 Example <annotation>: The annotation "AnchorId" with one sphere as the annotations markup geometry, a camera definition and links to two external information sources.

In the current version this component is realized as an ActiveX control, which is compatible with the interface of the ActiveGML plugins [BFH05]. The following C# snippet shows the typical usage of the control:

```
...
EpochViewer.LoadModel("mark1.xgml");
EpochViewer.Call("mark1.main");
...
```

One feature of the new ActiveEpochViewer control is its concise API with only four methods, two of which were available in the original ActiveGML methods (marked *italic*):

- *LoadLibrary(string)*
- *Call(string)*
- GetStackSize()
- GetStackElementAsString(index)

The two new methods replace the depreciated getStackTopAsString. The limitation of the existing interface was that only communication from the container application to the ActiveX control was possible. With the new introduced event called GMLEvent a bi-directional communication can be established. So the container application can receive notification from the scene graph engine (e.g. a node has been clicked). A complete technical overview of the EpochViewer framework would go beyond the scope of this section and will be described in Chapter 5.

This component can be used by all ActiveX-compliant software (like Microsoft Office, Internet Explorer, .NET Framework), which makes it the ideal platform for generating tailor-made graphical applications. Figure 3.12 shows the control embedded in a web page.

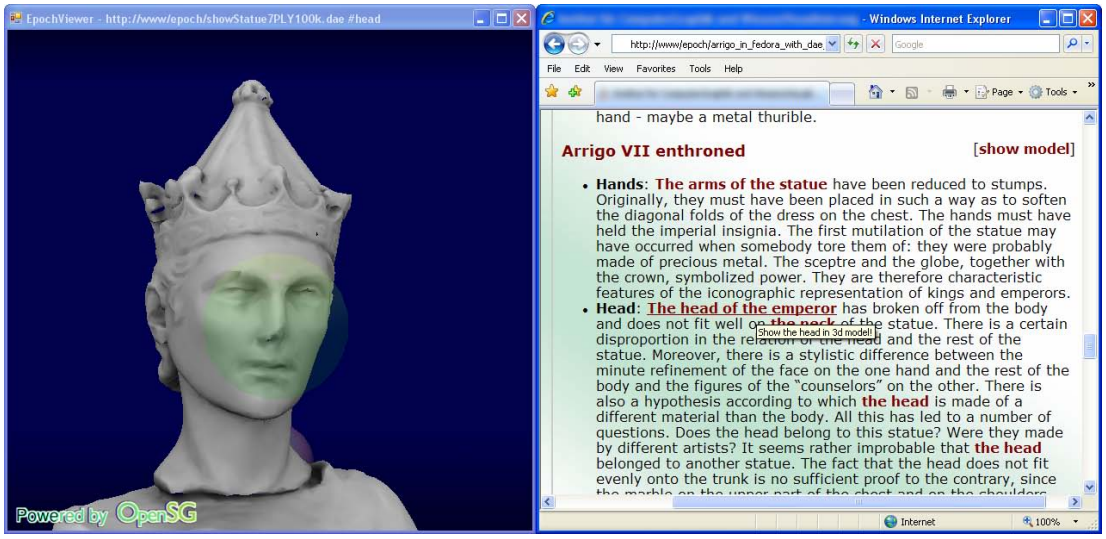


Fig. 3.12 This is the Epoch-Viewer ActiveX control inserted into an HTML page using the object tag. GML functions for loading and highlighting the annotations are triggered with JavaScript.

Sample Application

The Epoch-Viewer Framework is used by two showcase applications, an exemplary authoring tool and a viewer for the created semantically enriched 3D objects. The annotation procedure should be as easy as possible. People who have special knowledge of an object, e.g. a historian or an archaeologist, will not always have a lot of experience in 3D applications. So we tried to support non-3D-expert users by minimizing the functionality to what is absolutely necessary.

For the authoring application a standard mouse navigation is used. The user orbits around the scene and is able to zoom and pan to find good observation points for the annotations. In presentation mode, other forms of navigation are also supported including joystick input and tracking devices. Navigation is fully customizable by scripting.

A GML script defines internal functionality and user interaction. The GUI elements are capsulated into the enclosing .NET application. Smaller changes to the functionality can be achieved by modifying the GML script alone. For the showcase applications there are four functions in the script that are triggered by the container form.

- load-file and save-file
- show-anchor
- update-data

For quick processing the anchors of a scene are stored in a list. In GML, anchors are represented as dictionaries.

load-file

The load-file function expects a filename or URL of the selected scene on the interpreter stack. A standard open file dialog can be used in the container form to get a valid filename. The filename is pushed on the stack and load-file is called.

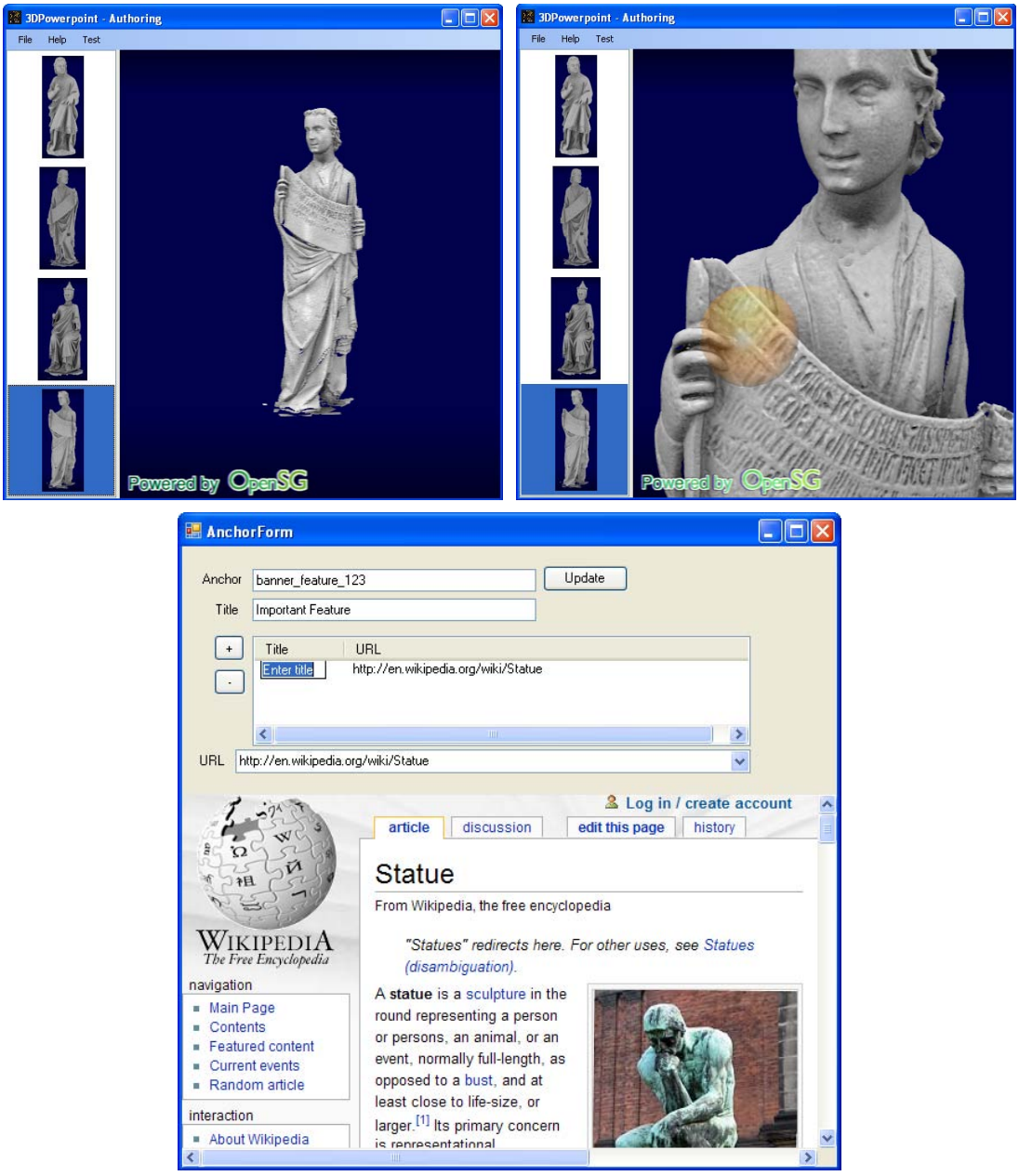


Fig. 3.13 The showcase for authoring: The user navigates to a position of interest. Annotations are added by clicking on the model and dragging the radius of the markup sphere. In the anchor form multiple URLs can be attached by typing or dragging the address from a browser. The dialog also includes a browser for preview and web navigation. (web browser: www.wikipedia.org)

Asynchronous loading is done by the loading thread in the OpenSG framework. A callback function is defined and connected to the load finish event. After the loading process is done, this call back function is triggered. A scene may consist of a single node or a tree of nodes.

With the new node as entry point, the subscene is traversed. If there are geometry nodes containing XML attachments, they may already contain annotations. The XML code is analyzed in another GML function. If there is an <extra> element, parse-extra is called to extract annotation information.

For the definition of new annotations every geometry node gets a mouse call back function. If there is a click on the geometry, a new markup sphere is generated. By dragging the mouse, the radius of the sphere is defined. The mouse button release event triggers an event back to the editor application for the annotation enter dialog. After the user has finished editing the annotation, the editor calls update-data.

The container form has a save as button that opens a standard save dialog and a save button. For the save as function, the form pushes the filename onto the stack and calls save-file-as, similar to the load function. The save-file function is called for the save button and triggers a write procedure in the control to save the scene back into the same Collada file.

show-anchor

The show-anchor function is called from the container form by the time the user requests the annotation data. This is possible by clicking on one of the anchors in the anchor list. The function expects the name of the anchor on the stack. If the anchor is found in the list of anchors, the markup geometry is highlighted and the camera animation is started to move to the defined location.

update-data

This function is called by the container form every time the user has modified the annotation in the anchor dialog. It expects the new XML code and the old anchor name on the stack. The old data is deleted from the list of annotation dictionaries and the new data is added. The old anchor name can be replaced by a new one defined in the dialog. The last step is to attach the new XML data to the OpenSG node.

parse-extra

The parse-extra function seeks the XML code of the <extra> element for all of the occurring <annotation> elements. For every annotation element, the annotation geometry is created. In the example code, this is always add-sphere.

add-sphere

This function gets the position and the radius of a new markup sphere and calls the primitive creation provided by the OpenSG control. The geometry is added to the scene graph in a special node that is excluded from exporting. For the markup geometry, again call back functions have to be defined to react on user mouse clicks. If the sphere is clicked, a camera animation moves the point of view to the predefined location and a special FinishPoint event is fired. The container application reacts on this event by adding a new anchor element to the internal anchor list.

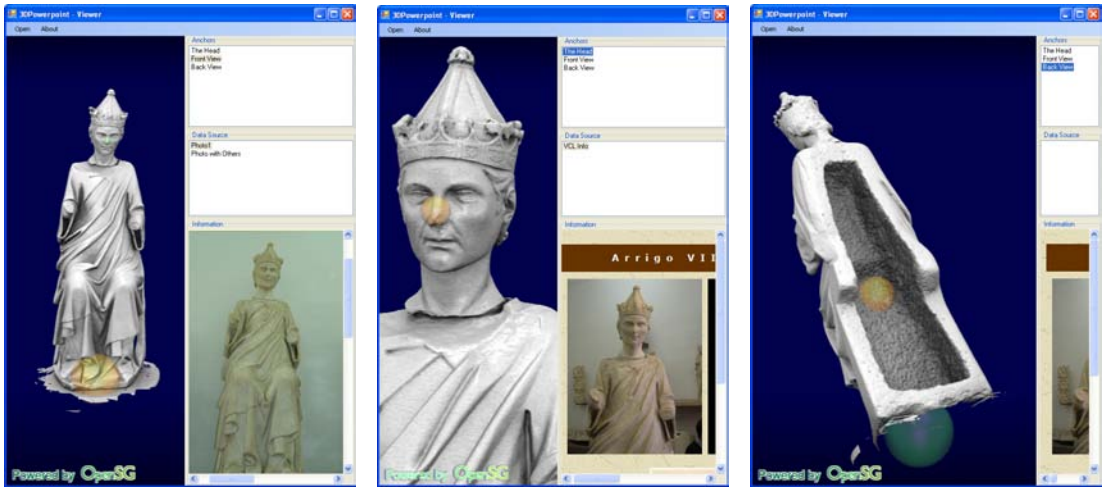


Fig. 3.14 A simple viewer showcase: The user can examine annotations by clicking on the markup sphere in the 3D scene or by selecting the anchor name in the list at the top right of the application window. The camera automatically moves to the predefined observing position. A list of attached URLs is displayed for the currently selected anchor. A simple browser control can display the content of the URL's target. (images in web browser: www.scultura-italiana.com (left), vcg.isti.cnr.it)

The Viewer

For the viewer application the same GML code can be used for highlighting annotations and animating the camera. Depending on the purpose of the presentation, GML scripting is a flexible way to generate all kinds of showcases as introduced in [HSLF07]. Besides the Epoch-Viewer control the container window includes a browser control to display the URL targets and a list of anchors (see Figure 3.14). Annotations can either be examined by clicking on the markup geometry in the 3D scene or by selecting an anchor from the list. By choosing an annotation, the point of view is automatically moved to the predefined position. Manual navigation within the 3D space is also possible.

3.4.1.5 Results

As promised we have presented a simple solution to a complex problem: 3D objects, even massive scanned datasets, can be provided with semantic annotations to provide arbitrary types of additional information on a sub-object level. The head of a statue, even its nose, can be linked bi-directionally to web resources like web pages, PDF documents, images and videos, and even Flash presentations – anything a standard Internet browser can display.

The format specification is also simple enough that automatic feature detection algorithms could be easily designed so that they produce Collada light files to let users browse through the detected features. The format is also suitable for long-time archival, for sustainable knowledge management, as long as it is made sure that the targets of the annotation links still exist; e.g., by using links that point into the same database. Consequently, one of our future research directions is to set up an extensive database of annotated Collada files, that point to web resources containing primary data (scanned datasets) of our colleagues in the EPOCH network of excellence on processing cultural heritage.

Another road for future research is to create more easy-to-use applications for creating, managing, browsing, and presenting semantic relations in 3D scenes. This has a huge potential

since in our three-dimensional reality we are surrounded by man-made objects with a long-standing tradition, which are embedded in a dense network of semantic relations. Consequently, another research direction is to make these semantics explicit, by coupling the 3D exploration tightly with a semantic database. RDF-triplets encoding CIDOC-CRM knowledge could then be embedded live in the 3D scene.

Speaking of larger scenes, another research direction is the question of scalability. We envisage that when providing our authoring toolkit with the functionality for interactive scene assembly, things will soon get very complicated. Collaborative 3D markup is already technically possible now (even in a time-synchronized manner), but we are aware that consistency issues and resolving conflicts will then become a serious issue. With denser semantics, also creating different views on demand by database-like selection queries for particular semantics will be needed.

3.4.2 PDF 3D for Annotations

The following work flow is presented in *"Publishing 3D Content as PDF in Cultural Heritage"* by M. Strobl, V. Settgast and R. Berndt, S. Havemann and D. W. Fellner.

Three-dimensional models can be incredibly rich and useful sources of historical information. 3D models may be obtained by measurements from (typically incomplete) physical remains of historical artifacts, or by synthetic reconstruction, hypothesizing about the original shape and appearance of things in the ancient past. In either case, a 3D model as such is not very useful without semantics, i.e., without interpretation, mark-up, annotation, or highlighting, that help to discern the important from the unimportant. Accurate interpretations rely on accurate raw data; accurate raw data mean nothing without interpretation.

CH professionals such as historians, archeologists, curators, but also conservators and restorers, have a hard time today making use of 3D models as information sources. It is by far more easy to create, annotate, and exchange drawings or photographs than 3D models. However, many questions can *not* be answered from photographs or 2D drawings. Complex spatial configurations, shape details, or distance measurements are barely possible with photographs. Whenever a thing needs to be looked at from all possible sides, and for serious questions beyond aesthetic imaging, e.g., sections or occludedness, 3D models are the medium of choice.

Research requires communication. Assuming 3D models are available and two cultural heritage professionals A and B wish to develop and share hypotheses and insights based on these models, which options do they have? In essence there are two:

- **Common Model Repository:** Both researchers refer to the same web-based information resource, a common 3D infrastructure, e.g., by sending each other links (URLs) that refer to positions in 3D models in the data base.
- **General Purpose Modeling Tool:** Researcher A loads a model into 3D Studio Max, Maya, Blender etc., inserts annotations (typically by creating additional geometry or text embedded in 3D), saves it, and makes it available to researcher B, e.g., via e-mail or FTP.

The repository solution is clearly more sustainable. It makes a sharp distinction between the model and the annotation, and it permits multiple interpretations; but it relies on an online connection, and on the long-term availability of the 3D model in the repository. The second option has the advantage that it is self-contained, but the annotation follows no standard, and it "pollutes" the 3D model. Technical issues are that import/export filters of 3D tools greatly vary

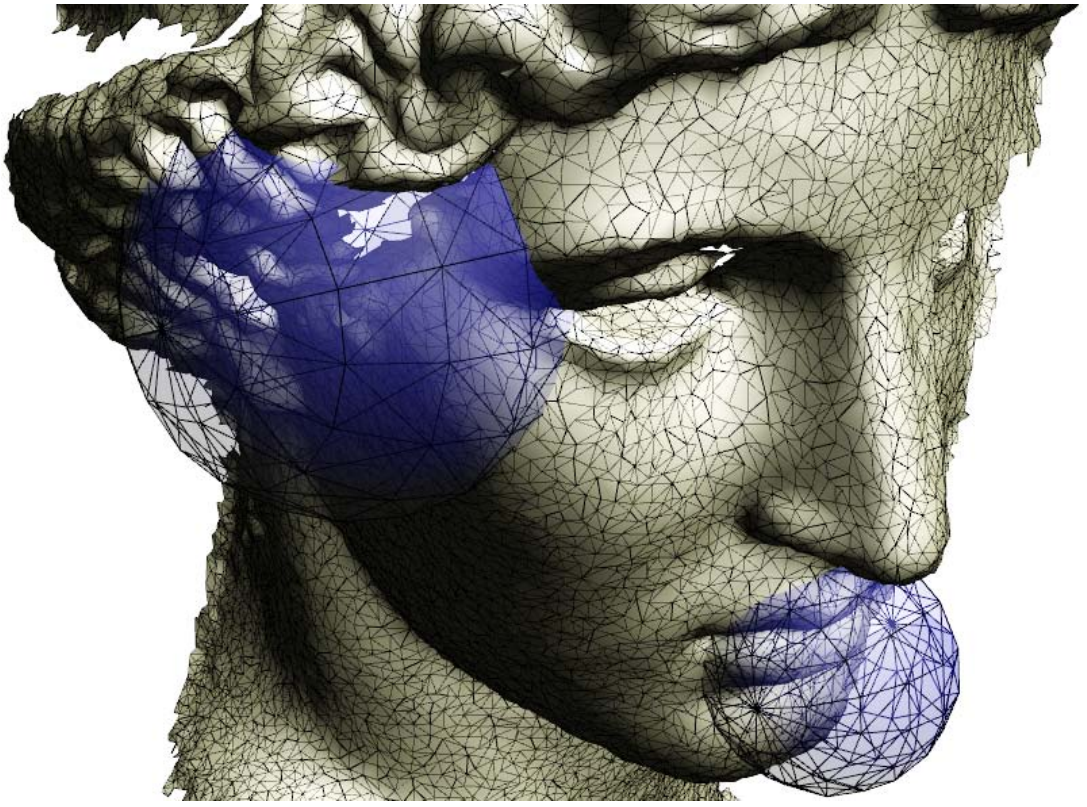


Fig. 3.15 A 3D model file with embedded annotations. Blue spheres have URLs attached.

in quality, and there is always the danger of loss of information (fidelity). So A and B should better agree on using the same 3D software.

Another problem is using 3D models in scientific publications. Today, 3D models appear in publications only as 2D images, e.g., as nice rendering, essentially a digitally born photograph. Nice renderings are difficult to obtain and require being acquainted with rendering methods and their parameters; using, e.g., MentalRay in Maya or 3D Studio Max requires expert knowledge. Furthermore, it seems odd replacing the richer three-dimensional information by its projection to 2D. Why not include the 3D model in the paper, all the more since all scientific articles that are published today are available in electronic form – typically in PDF format.

3.4.2.1 Contribution and Benefit

We present a PDF-based solution that combines the advantages and avoids the disadvantages of the two approaches from the previous section. Since PDF is the de-facto standard for electronic documents, it is only natural to examine its usefulness for reasoning about historical 3D models. Since version 1.6 (from 2005) PDF supports 3D. PDF viewers are available basically everywhere, so no special software installation is required for viewing 3D models embedded in PDF documents. However, currently only the free *Acrobat Reader* software from Adobe Inc. fully supports viewing embedded 3D; but since 3D is now part of the PDF standard, other PDF readers will follow sooner or later.

Previous work has shown how sustainable 3D markup can be achieved by a strict separation of 3D model (stored in binary format, e.g., PLY) and annotation, stored as a Collada XML file,

referring to the 3D model as external resource [HSB*08]. The solution builds upon this work and extends the markup facility, so that no special software except a 3D capable PDF viewer is necessary to inspect the annotated 3D models. The contributions of the solution are:

- Explaining the export of annotated 3D models to PDF
- Explanation of the manipulation of the PDF structure
- Distinction between model and annotation is still possible
- Three use cases: Sending annotated 3D models as PDF via e-mail, including them in scientific publications using Microsoft Word, and using LaTeX.

This solution allows easy inclusion of annotated 3D models into scientific publications, allows archiving them, and allows taking annotated 3D models to places without network connection (e.g., on a remote excavation campaign).

It is important to maintain the difference between model and annotation, which is not possible reliably if the model is annotated using general-purpose 3D software. Our approach is to exploit the built-in annotation capabilities of PDF, i.e., to integrate the markup into the PDF document structure, rather than into the 3D model. The model can be discerned from its interpretation also in the long run, and both can be extracted separately from the PDF. This is an important prerequisite of any 3D file format suitable for archival. Furthermore, in the common infrastructure scenario mentioned above, where a large 3D model repository is available as online resource, the annotated PDF can be regarded as a snapshot that is available even if the online resource vanishes, or if the link location changes. A central cultural heritage infrastructure clearly is an extremely desirable (and ambitious) goal, and it is incredibly valuable as research tool; however, the ability to achieve independence from it is desirable as well.

Adobe provides the *Acrobat Pro Extended* software for inserting 3D models into PDF documents, which (in 2009) costs more than 900 Euro. Our current solution depends on this software for exporting (but not for viewing) annotated 3D models as PDF. Clearly, our goal is to become independent from any proprietary software; ways for doing so are sketched in Section 3.4.2.4.

Fig. 3.16 The 3D model is annotated using the *EpochStudio*. The software is composed of a menu (left), a 3D view for the detailed inspection of the model (middle), and a browser for web-based research (right). Annotations can be placed on spots on the surface of the 3D model, shown as transparent balls. Annotations can be connected to one or multiple URLs.

3.4.2.2 Previous and Related Work

PDF is used intensively in the manufacturing industry as exchange format for annotated 3D models. Sections and 2D drawings are insufficient for describing free-form surfaces. So PDF export functionality is available in almost all major CAD applications, just to mention Dassault Catia V5, Bentley MicroStation and Graphisoft ArchiCAD. Main reasons for using it are the advantage of 3D models over 2D plans, the wide spread availability of PDF viewers, and security considerations. Commercial solutions like PDF3D from Visual Technology Services [Vis09] allow integrating PDF export also with existing production work flows and applications. PDF is used for product presentation and explanation, e.g., for 3D user manuals that explain unambiguously the assembly and disassembly of complex machine parts.

In the manufacturing industry the standard today for organizing huge masses of 3D content with rich structure are extremely complex product data management (PDM) systems. They help developing industrial plants with huge machines composed of various parts and sub-parts, each produced by different companies. PDM systems manage versioning information and help keeping up with change requests and maintaining geometric consistency. Once a facility is built, digital plans and built reality are kept in sync using similarly complex product lifecycle management solutions. PDM/PLM systems, for example *Windchill* from PTC, allow creating and exchanging annotations to 3D models, and importing / exporting them via PDF.

It may be surprising that in fact the same requirements apply to 3D in cultural heritage: Large excavation sites exhibit extremely complex geometric configurations (strata), on multiple scales (from single artifact to landscape), and both physical state and knowledge/interpretation change over time. But unlike in the industry setting, not just clean CAD data need to be managed, but huge masses of measured 3D data that affected by noise, have outliers, etc. Unfortunately, PDM/PLM solutions cost several 100K Euro.

But also free and open source software exists for adding 3D content to a PDF document. The *Meshlab* software by Paolo Cignoni et al. from ISTI-CNR in Pisa, Italy [CCC*08], allows editing 3D meshes and converting many 3D file formats to U3D. The *media9* package (replacement of the obsolete *movie15* package) for LaTeX by Alexander Grahn provides means for embedding various multimedia objects into PDF files generated by LaTeX. U3D and PRC files can be embedded, and JavaScript can also be attached to the PDFs' 3D data streams. The annotation of a 3D model must be done using an authoring tool like Adobe Acrobat.

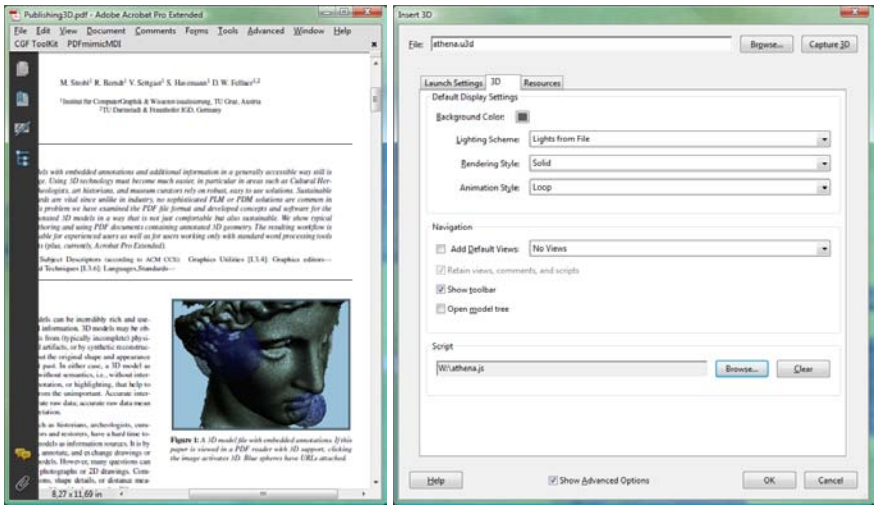


Fig. 3.17 Using the *3D Tool* of Acrobat for inserting 3D content.

Concerning the PDF standard, PDF/A-1 has been approved in 2005 as the standard for long-term archival of textual PDF documents [ISO05]. The most recent version PDF/A-2 [ISO08] (approved in 2011) is based on PDF 1.7 and includes new features added with PDF version 1.5, 1.6 and 1.7, like JPEG2000 image compression etc. Unfortunately no standard for 3D content was added. The PDF/A-3 version of the standard will hopefully also include 3D data. This would make PDF indeed the preferential format for the sustainable publishing of 3D content mixed with other data like text, sounds or videos.

Using 3D in PDF has apparently not received much research attention, and is also not common in cultural heritage. Rieko Kadobayashi developed a system for automatic 3D blogging in cultural heritage that uses annotations to 3D models [Kad05]. Visitors of virtual museum artifacts can make textual annotations and attach them to spots on a 3D model of the artifact. Other users can subscribe the annotations of a particular model as a blog. This system could use our approach to send out the annotations as PDF, or to let users make a snapshot of the annotations of a particular object.

3D in PDF was used as research tool by Barnes and Fluke in the area of astronomy [BF08]. Their S2PLOT library for visualization, e.g., of astronomical data whose VRML export was complemented by PDF3D export, because the authors noted that 2D illustrations are insufficient to understand the complex 3D data in astronomy. However, they use it only as export filter and make no particular use of annotations.

3.4.2.3 Use Cases

This section describes different options for the workflow in order to publish annotated 3D content:

- Annotating 3D content with EpochStudio
- Adding 3D content using Acrobat 9
- Adding 3D content using Word
- Adding 3D content using LaTeX

The workflow for two of the most popular text authoring tools, Microsoft Word and LaTeX, is described in the following. Note that 3D can still be used even for other text authoring tools: The fallback solution is simply always to generate the document with a placeholder, and to replace this placeholder using *Acrobat Professional* (Extended not needed).

Authoring 3D annotations with EpochStudio

The annotation of 3D models is done using the *EpochStudio* software that was presented in [HSB*08]. Figure 3.16 shows a typical authoring session with the 3D view for displaying the model in the middle and a web browser for investigation on the right.

The model is loaded from the local hard drive or from a remote data source. By clicking on the authoring tab on the left, the user can select *"Add Annotation"* from the menu. Annotations are then added by clicking on the surface of the 3D object. A dialog lets the user choose an anchor name. This name can be used later to create a reference to the annotation. Anchors are visualized as spheres in the 3D view and they also appear on the left in the list of anchors.

An arbitrary number of URLs can be attached to each annotation. With the included browser it is possible to research on suitable material without leaving the application. After all annotations are added the user can export a simple Collada file by clicking on *"Save DAE"*. This Collada file holds the scene graph hierarchy and the annotations. The 3D models, however, are

not contained in the Collada file but only referenced using a link. To generate a PDF, the user selects another export option, *"Publish PDF"*. A PDF file containing the 3D model as well as the annotations is then written to disk. For the manual integration we also export the 3D scene as VRML and a separate table in simple CSV (*comma separated value*) format to describe the mesh annotations, viewpoints, and textual annotations. Using a CSV file means that the interface for the creation of annotated 3D PDF models is very robust, clear, and simple to use.

Fallback solution: Acrobat Professional

With Acrobat Professional, 3D content can be integrated into an existing PDF document using the *3D Tool* (from Tools / Multimedia). The user specifies the area where the 3D content shall be placed on the page by drawing a rectangle. A file dialog asks for the 3D model, and advanced options include an optional JavaScript file (e.g. for animations), various render styles, background color, and many more (see Fig. 3.17).

PDF with 3D from Word (with PDFMaker)

The installation of Adobe Acrobat includes the Word add-in PDFMaker, which enables advanced PDF features like embedding videos and 3D content. Figure 3.18 shows the toolbar of the PDFMaker in Microsoft Word 2007. The *"Embed 3D"* button allows drawing a rectangle on the page and asks for the 3D content file to insert. The context menu of the inserted content allows changing the options concerning the JavaScript file (used, e.g., for animations) and other options like render style, background color, etc.

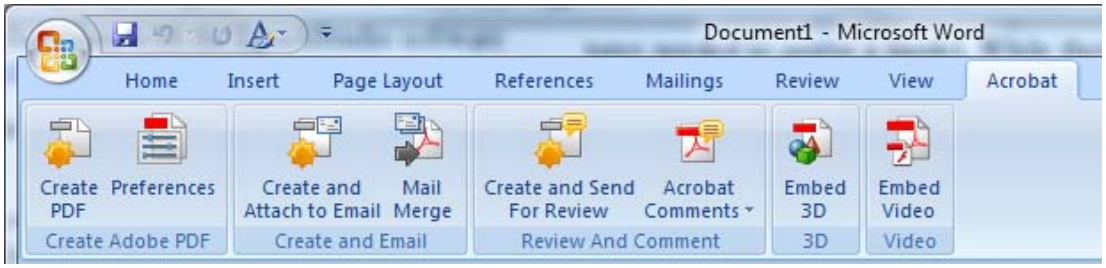


Fig. 3.18 The *PDFMaker* add-in in Microsoft Word.

Without installation of the PDFMaker add-in the fallback solution is to insert a placeholder (as explained above).

PDF from LaTeX

It is also possible to include the 3D content into a LaTeX document. The straightforward solution, to use pdflatex to include the generated pdf file directly, leads to an error, since pdflatex cannot include PDF files of version greater 1.4 directly. Hopefully this limitation will vanish soon. There are basically two alternative options. The first one uses the *media9* package to include external media. The second is to include a PDF generated by the Epoch PDF Publisher at a placeholder position. The LaTeX document is generated as usual, but at a specific place, a

\vspace or a \newpage directive generates empty space. The generated PDF can be included using Acrobat Professional.

3.4.2.4 Results

Manipulating 3D annotations in PDF is certainly not a straightforward practice. One needle's eye of the current implementation is the need for a controller application to contact an Acrobat plugin if anything inside a PDF 3D data stream needs to be modified. The SDK is quite cumbersome indeed, and errors in the mesh manipulation using the SDK tend to produce rather error prone 3D data streams that are difficult to "debug".

The annotated 3D PDFs we have produced nicely perform the task of archiving annotations. However, Acrobat Reader yields a rendering quality of the annotations that is not up to par with current technology. The balls we inserted into the 3D annotations, for example, show ugly artifacts.

The EpochStudio permits attaching multiple URLs to the same spot, because in the cultural heritage context these URLs are considered interpretations; and there will typically be more than one valid interpretation. We have not found a good solution for attaching multiple URLs to the same location using the JavaScripts included with the 3D model. It would, for instance, be convenient to open a dialog window that lets the user select one of the attached URLs. This is not possible with the JavaScript code in the 3D annotation, but there is a chance that it might be achievable using the JavaScript embedded on *document level*.

For highly detailed models the conversion to the PRC format can take a significant amount of time, e.g., two minutes for a PLY file of 20 MB.

To satisfy intellectual property protection requirements, and to reduce the file size, the 3D model can be embedded in simplified form. In addition Acrobat offers the option to either "allow" or "prevent" extracting the 3D data from the PDF document, which is not really secure because 3D data can always be read on the graphics driver level. The standard security features of PDF, such as or digital signatures, can also be applied to PDF documents containing 3D.

Collaborative review and knowledge exchange can be easily enabled by activating the option *"Enable For Commenting and Analysis in Adobe Reader"*. Then any user can add comments and save them using the free Acrobat Reader (see Fig.3.19).

We have identified a whole range of different possible ways of using the 3D capabilities of PDF in cultural heritage beneficially. The ubiquitous availability of Adobe's PDF Reader and Acrobat provides for a nice, portable format to supply end-users with annotated 3D models, i.e., models containing not only shape but also interpretation. Embedded JavaScript code allows for great interactive illustrations that can be defined by any 3D content authoring software generating PDFs. We have exploited the JavaScript capabilities of 3D models for attaching information to locations on the surface of 3D models. Other uses would be camera-flights or animating 3D models to show alternative spatial arrangements of historical artifacts.

An area to explore further is interactive PDF (and 3D PDF) manipulation controlled by a webservice. PDF seems to be versatile at communicating with web servers, although this is apparently not possible using JavaScript on the level of 3D annotations. Document-level JavaScript has a so-called Net.HTTP object that might be usable for giving control over the PDF to a webservice.

The current solution makes use of the Acrobat SDK for manipulating the PDF document. The drawback is that this requires the commercial software *Adobe Acrobat Pro Extended* to be installed. This makes it impossible to use it in a web service solution due to license issues. An alternative would be to use the PDFLibrary SDK from Adobe, which provides the same API

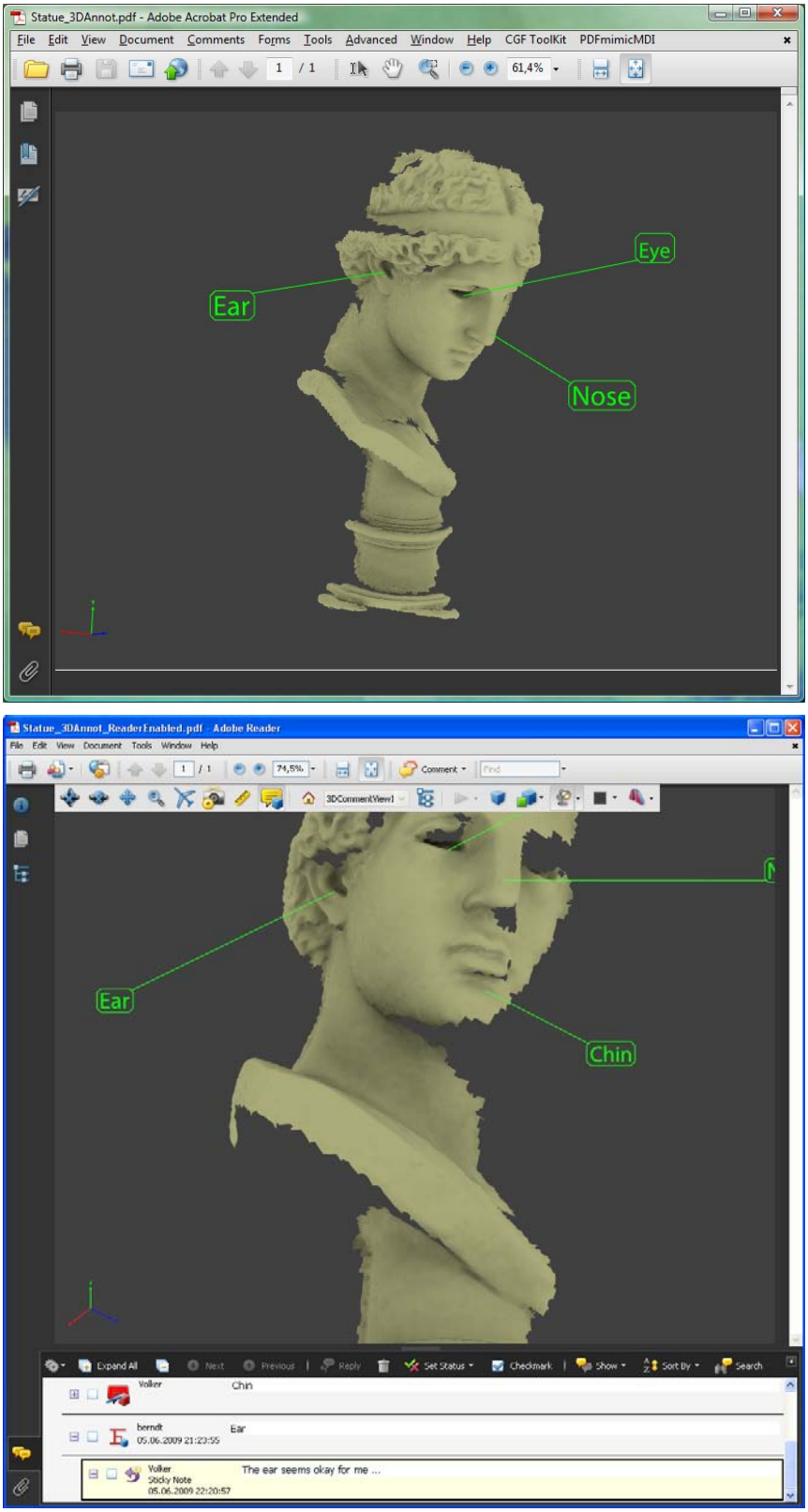


Fig. 3.19 Collaborative commenting of 3D content. *Top:* Researcher A adds 3D comments to objects and enables commenting for the Acrobat Reader. *Bottom:* Researcher B uses the Acrobat Reader to add new comments and can also reply to the comments of A.

interface as the Acrobat SDK, but can be used to develop stand-alone application. Pricing is royalty based and depends on the Adobe PDF Library functionality used by the application. Another alternative is the commercial PDF3D SDK from Visual Technology Services, a C++ library that permits direct publication of the 3D content. This is very useful when adding PDF export capabilities to existing applications.

3.5 Insight

Semantic information adds additional value to the data and enhances the presentation of 3D content. In this chapter I have presented a classification scheme for semantic markup. Further more a method to transfer semantic knowledge of higher order to unstructured, automatically created geometry, e.g., point clouds from a laser scanner is presented. I have discussed the requirements for the integration of 3D models into cultural heritage libraries. Two solutions are proposed for a tight connection of 3D content and semantic information. The first one uses attachments to scene graph nodes in OpenSG and the Collada file format for storage. The other solution uses the 3D capabilities of Adobe's PDF format instead. They both use the OpenSG-based Epoch-Viewer Framework for arranging and annoating 3D content.

The following chapter will compare suitable graphics systems for interactive 3D applications. Furthermore methods to prepare arbitrary 3D content for interactive rendering are described and user interaction methods for immersive environments are presented.

Chapter 4
Interactive Presentation

Abstract The final step of the process is the presentation of the data. In this chapter diverse forms of interactive presentations are described. First different graphics systems for interactive 3D applications are explained. Then various environments like table displays and immersive environments are listed and their special needs are described. Furthermore methods to prepare arbitrary 3D content for interactive rendering are described. User interaction methods for immersive environments are presented. Finally a method is proposed to integrate legacy software into virtual reality environments.

4.1 Graphics Systems

Nowadays even standard office computers are capable of rendering hardware accelerated three dimensional graphics. To bring graphics onto the screen, low level application programming interfaces (API) are the common way to talk to the graphics hardware. It safes a lot of development time to build visualizations upon existing higher level abstraction libraries or frameworks. There are various general scene graph systems. One which is used in many of the applications examples of this thesis is OpenSG[1] which will be described in Section 4.1.2.1. The most powerful graphics systems are developed for games. More and more also those game engines are used for serious visualizations. As many of the professional game engines are freely available for non commercial use, it is worth to give them a try.

4.1.1 Low level APIs

The two most commonly used APIs for 3D graphics are the Open Graphics Library (OpenGL) by Silicon Graphics [SB04] and DirectX by Microsoft [Mic10]. DirectX is restricted to the Microsoft Windows operating system, Windows Phone and of course the Microsoft game console XBOX. It was initially released 1995 with Windows 95. Since then many new hardware features have been added and the API was improved. The most recent version 11 of Direct3D was released in combination with Windows 7 in 2010. A software development kit is available from the Microsoft web page.

[1] http://www.opensg.org/

OpenGL is available for many platforms including Windows, Linux, Mac OS. Its cross platform capabilities and its openness make OpenGL the more popular API in the field of academic research. OpenGL was released in 1992 as an open standard. Over the years, many new features have been added to OpenGL including the OpenGL Shading Language (GLSL) in OpenGL 2.0. At the time of this writing the latest version of OpenGL is version 4.3[2].

A subset of the OpenGL API called OpenGL for embedded systems (OpenGL ES) is designed for embedded systems like smart phones, tablets and game consoles. The current version 2 of OpenGL ES is the basis for WebGL, a JavaScript API for 3D graphics rendered directly on an HTML5 canvas element. The hardware-accelated rendering works in compatible web browsers without any additional plugins.

4.1.2 Scene Graph Systems

A scene graph is an object oriented structure to describe the logical and spatial arrangement of 3D elements in a scene (compare with Section 2.2). Many implementations have been developed to introduce suitable data structures for scene graphs. Starting from a simple linked list structures have been optimized for typical graph tasks and also for more flexibility and extensibility.

Scene graph systems are not restricted to the management of 3D models. They offer much more functionality to form a high-level graphics API. Such systems feature data structures for material and shader organization, lighting, shadows and animations. To have convenient access to 3D data they offer various import and export possibilities and they also cover user interaction. Therefore the difference of a scene graph system to a game engine is not that big.

On the basis of HTML5 and WebGL many frameworks have been developed for web-based scene graph systems. One of them is X3DOM[3] (Fraunhofer Institut für Graphische Datenverarbeitung (IGD), Darmstadt) which is an experimental integration of parts of the X3D ISO-standard to HTML5. Another, very similar project is XML3D[4], developed by "Deutsches Forschungszentrum für Künstliche Intelligenz" (DFKI) at Saarland University. The XML3D project defines an integration of 3D graphics on the basis of the HTML5 standard.

4.1.2.1 OpenSG

OpenSG is an open source C++ scene graph system for realtime 3D graphics. After the Fahrenheit scene graph project of Microsoft and SGI was not continued in 1999 among others OpenSG was born. It was initially designed and implemented by Dirk Reiners, Gerrit Voss and Johannes Behr at Fraunhofer IGD in Darmstadt [RVB02]. OpenSG is based on OpenGL and it is freely available under the Lesser GNU Public License (LGPL). At `http://www.opensg.org/` sources and libraries can be downloaded for Windows, Linux and Mac OS.

The plan was to develop a scene graph system that serves as a general basis for a wide variety of applications. OpenSG 2.0 is currently the latest version with many improvements in usability and supported features.

From the beginning it was designed to make optimal use of the graphics hardware. To obtain optimal performance OpenSG provides optimization algorithms. For example it provides

[2] `http://www.opengl.org/`
[3] `http://www.x3dom.org/`
[4] `http://www.xml3d.org/`

functions to transform a model consisting of triangles to connected triangle stripes and a material sorting optimizer which minimizes the number of state changes for rendering the scene. Also many high level optimizations are included in the OpenSG renderer like visibility culling. The processing of scene parts which are behind the viewer or occluded by other parts can be skipped by OpenSG to speed up the rendering of large scenes.

OpenSG allows to have multiple threads accessing and manipulating the scene graph in an efficient way [VBRR02]. Data structures are organized in multi-thread safe buffers called aspects. Each thread can be assigned to its own aspect meaning it can have its own copy of the buffer. To reduce memory overhead the copy is created only on demand. Remote aspects are copies of the buffers which are send over the network. They are used to synchronize the scene graph within a cluster of multiple machines. In this way the rendering of complex geometry can be handled interactively. Also virtual environments consisting of many screens and machines like tiled displays and CAVEs™ are implemented with remote aspects.

Not only by making the source code available but also by using highly dynamic and flexible structures OpenSG can easily be extended or adapted to specific needs. Custom render nodes can be added to extend the render capabilities. Reflective interfaces allow to integrate arbitrary types of custom data to the whole system. New application-specific data which is not known by the system at compile time can be used in the internal loaders and writers and synchronized to a cluster.

4.1.2.2 GML as a Scripting Language for OpenSG

John K. Ousterhout makes the distinction of system languages on the one hand and scripting languages on the other hand [Ous98]. He describes a change of application development towards scripting languages. Interactive graphics systems are time critical. Core functionality will still rely on system languages, typically C or C++. But for user interaction, animations and the overall logic of the application scripting languages have advantages: The development of applications is much faster. Scripting languages are easier to learn so the programming does not require an graphics expert. Modifications and adjustments are fast and easy done without recompiling the core parts of the application.

A scripting language for OpenSG based graphics applications was needed. In Instant Reality it is JavaScript which is added to the X3D scene. But the extensions of Instant Reality are not open source and a JavaScript interpreter is quite heavy. As an alternative we already had the integration of GML in OpenSG, mainly for geometry creation but also for user interaction. The first version of this integration was developed for OpenSG 1 by Björn Gerth [GBHF05].

Martin Hecher re-engineered the connection of GML to OpenSG 2. To support procedural geometry in other OpenGL-based applications he added a general interface for the mesh library *Meshlib* of GML. For the integration of GML into OpenSG development was needed on both sides. In GML the OpenSG FieldContainer structure is mapped to a fake dictionary. Data types are converted from GML to OpenSG and vice versa. In this way GML programs can access and modify the entire content of the OpenSG scene graph. In OpenSG the rendering of meshes created by GML was added. The cluster support of OpenSG had to be added to the GML core library.

A special material handling was necessary. The material management was part of Meshlib from the beginning. The render method is able to choose from a set of OpenGL-based materials on a per-triangle level. For performance reasons the primitives are sorted by material - a strategy that also OpenSG uses to minimize state changes of the graphics hardware. In OpenSG the material management is more complex than that supporting not only standard OpenGL mate-

rials but a very flexible interface for modern shader materials. To support materials defined in OpenSG for the rendering of GML generated geometry, the core render methods of the Meshlib had to be modified. GML programs address materials using names (strings). Internally, those names are connected to the set of Meshlib materials. With the modification of the Meshlib it is now possible to extract all the used material names and connect them to OpenSG materials.

OpenSG can use the same geometry in many scene graph locations. This is called instancing. To allow the same concept with GML generated geometry, instancing had to be added to the Meshlib. The base mesh can be shared, but the adaptive subdivision levels have to be calculated separately for each instance. Instances with different parameters are separated completely, because the base mesh will of course be different as well.

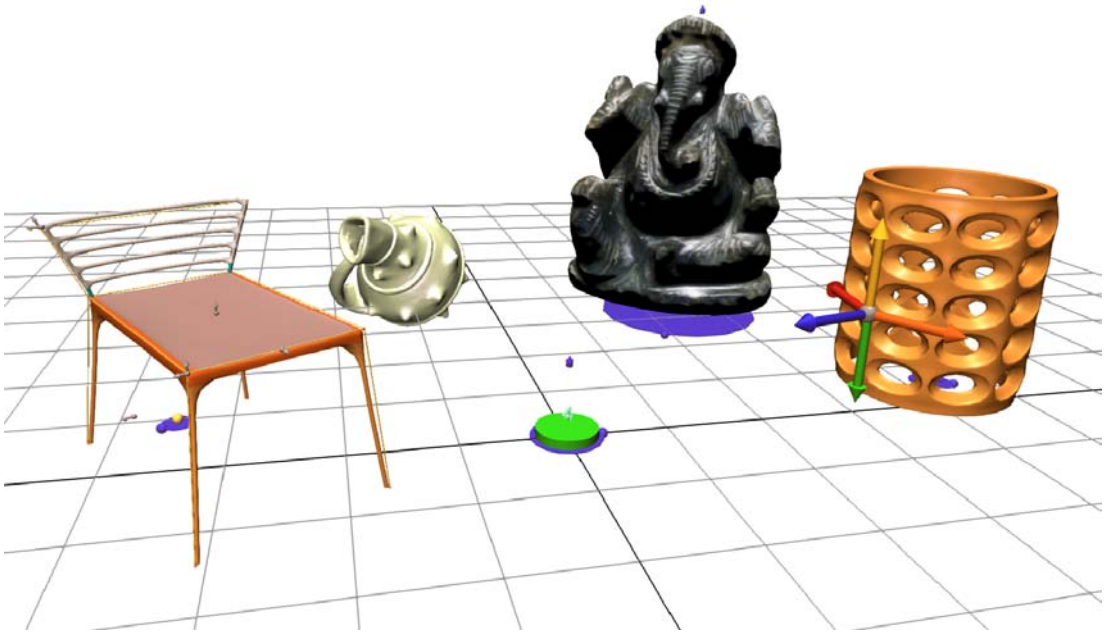


Fig. 4.1 This is an example scene showing some interactive GML objects integrated into an OpenSG scene. The scene is created with the 3D-COFORM scene assembler. (image courtesy of Martin Hecher, static content: 3D-COFORM).

In OpenSG the GML cannot only be used for geometry creation but also as a scripting language for creating complex scene graphs and control user interactions. The idea is to be able to create a large variety of OpenSG-based applications without recompiling mostly similar C++ code.

A set of OpenSG specific operators was added to the GML to make modification of the scene and scripting of user interaction more convenient. The operators are divided into five groups:

OpenSG Core The core functionality covers operators for accessing the fields of the field containers like osg-getfield, osg-getfcid, osg-insertfield, osg-subfield, osg-replacebyfield, osg-replacefield, osg-numfields and operator for creating new nodes, adding children to existing nodes and removing them like osg-newnode, osg-corednode, osg-addchild, osg-subchild, osg-clear With osg-clonetree a subgraph is copied to a different location in the scene graph. The osg-parent operator accepts both nodes and node cores. For nodes it returns the parent node and for cores it returns the nodes which are using the core.

Creating nodes can also be done with the osg-async-load operator which takes a file name string parameter and creates a subgraph with the file content. The loading of large geometry will not block the application. It is asynchronously handled in a separate thread. When finished the operator can trigger a call-back function. It may be necessary to react on the successful loading of geometry for example to place it in the scene.
After loading a complete scene the application may need to access a specific part of the graph. This is done by finding nodes by name with osg-find. For picking nodes interactively the operator osg-rayintersect triggers a ray intersection test and reports the hit point and the node core. The transformation cores are modified with osg-translate, osg-rotate, osg-scale.

Scene Manager With the scene manager it is possible to access the window, the camera and the scene graph. The operator osg-getroot returns the root node and osg-showall sets the camera to a position from where the whole scene is visible.

Primitives For prototypes of applications it is helpful to create simple geometric primitives like boxes and spheres. The creation parameters can be updated interactively at any time.

Application control and User Events Animations of transformation nodes can be defined with osg-anim-rotate, osg-anim-scale and osg-anim-translate. The duration of the animation is given as parameter. Then the animation can be started with osg-anim-start. Camera animations can be defined which are then started within a separate thread by executing osg-anim-camera. Additional callback functions can be registered for drag and drop events, for joystick events, for touch events and tracker events. Some specialized input devices have been added for the EPOCH project.

Scriptable Materials To modify materials within GML a custom material definition is added. It is accessible like a dictionary in GML and creates shader materials in OpenSG. This is used in the Autovista project to control the video texture shader (see Section5.2.1). For the visualization of the crowd analysis results an animated shader material was added as well.

The GML scripting for OpenSG is successfully used in the EPOCH project and in the Autovista project. GML models of train tracks and escalators are combined with an OpenSG scene graph in the mPed+ project (see Figure 4.2). The projects are described later on in Chapter 5. It is flexible and can be used for any kind of OpenSG-based visualization. For computationally intensive tasks it can be extended with operators written in C++. But the experience with students has shown that the GML is not beginner-friendly. Especially when applications become larger the structure of GML code is too complicated for students who are typically not familiar with the post-script syntax. As an alternative to GML scripting there is now (since 2012) the Python language interface freely available for OpenSG 2.

A different way to modify the scene graph and to control applications is the use of web services. The next section describes an implementation of a web service manipulation system for OpenSG 2.

4.1.3 Service-Oriented Scene Graph Manipulation

In this section a software architecture is presented for the integration of a RESTful web service interface in OpenSG applications. The proposed architecture can be integrated into any OpenSG application with minimal changes to the sources. Extending a scene graph application with a web service interface offers many new possibilities. Without much effort it is possible to review and control the scene and its components using a web browser. New ways of (browser

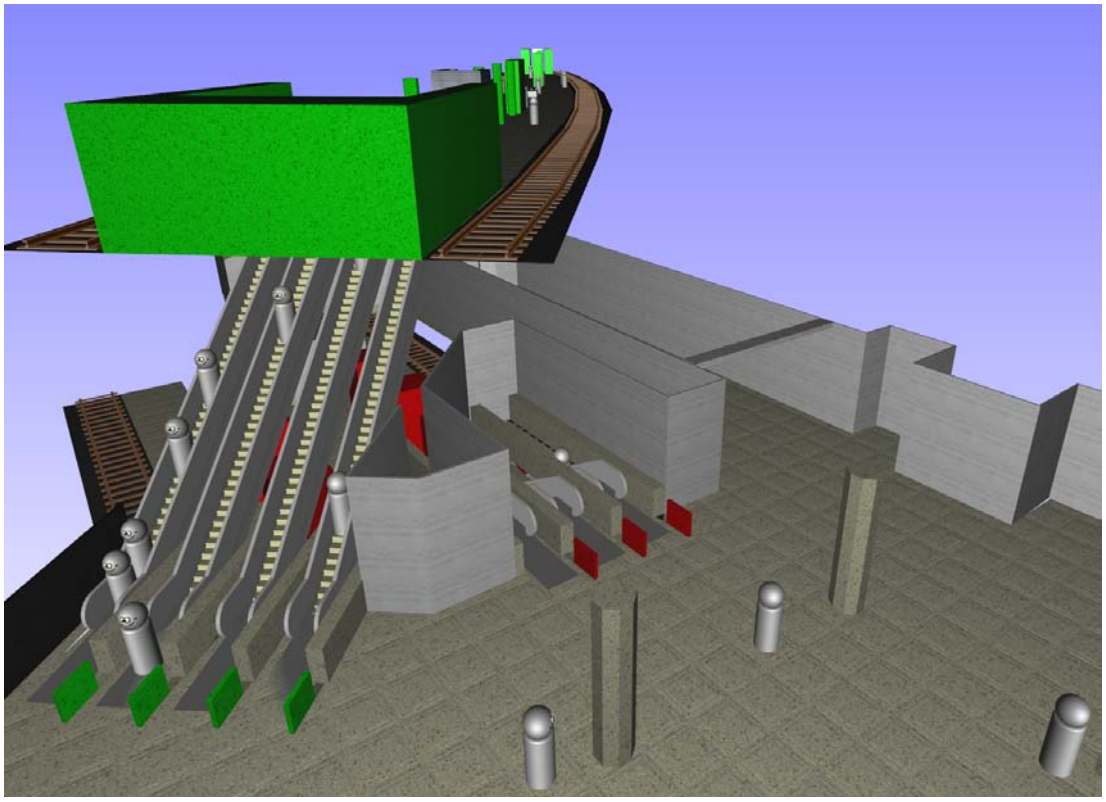


Fig. 4.2 Generative models of escalators and train tracks are used in the OpenSG-based visualization module of the mPed+ project.

based) user interactions can be added on all kinds of web enabled devices. As an example we present the integration of "SweetHome3D" into an existing virtual reality setup.

4.1.3.1 Introduction

The intention to combine a web service with a scene graph system is to address the following problems:

Modeling Visualization Cycle Most virtual environments offer only limited modeling capabilities. As "standard" modeling software (Maya, 3D Max, etc.) is seldom adapted to virtual reality (VR), many VR systems are just viewers. In this case, the modeling-visualization-cycle is interrupted; i.e. a human modeler has to switch frequently between modeling environment and VR visualization. Especially during presentations where customers inspect 3D models in VR and want to apply model changes (customization), an interrupted modeling visualization cycle is not feasible.

VR Correlated UX Due to differences in VR systems (High-resolution projection walls, CAVE environments, etc.), the user interaction and user experience (UX) implement different paradigms. The interaction capabilities may vary from desktop-based mouse and keyboard interaction to multi-touch or 3D gestures. These shifting hardware situations result in a substantial adaption and implementation effort.

The proposed solution to the previously described problems is a flexible software architecture. It strictly realizes the model-view-controller pattern. As a consequence the software develop-

ment process can be parallelized. Furthermore it is easily adaptable to new user interfaces and requirements.

4.1.3.2 Web Services

Web services provide an implementation for a service-oriented architecture (SOA) [Jos07]. The main characteristic of this software design principle is a loose coupling of services. These services communicate by exchanging messages. This provides a great flexibility in term that two applications only need to agree on the message format and are independent from the actual application code.

The following definition of a web service was introduced by the World Wide Web Consortium (W3C) [BHM*04]:

> *A web service is a software system designed to support interoperable machine-to-machine interaction over a network. . . .*

Web services offer a number of advantages over other machine-to-machine middleware systems like Open Network Computing Remote Procedure Call (ONC RPC) [Sri95], Distributed Component Object Model (DCOM) [BK98], Common Object Request Broker Architecture (known as CORBA) [Obj], Remote Method Invocation (RMI) [Sun04], etc:

Platform support Web services use the world wide web for communication. Every platform got already built-in support for the web. Web services can be consumed by web browsers. Since most platforms already include a web server even hosting web services can be done without additional efforts. Other module communication systems are only supported by a small number of platforms (e.g. DCOM is available only for the Windows platforms) or do not ship with the platform and need to be installed separately (e.g. CORBA).

Language support Almost every programming language provides support for internet protocols and XML processing (C++, Java, C#, JavaScript, Python, Ruby, etc.). Although also other middleware systems (e.g. CORBA) claim to be language independent one will find only support for C, C++, and Java.

Internet The major drawback of systems like DCOM and CORBA is that communication across the internet is nearly impossible. These middleware system use binary formats which are blocked by most firewalls. Even in the same department it can be a painful task to configure the network to successfully communicate using DCOM/CORBA.

Currently, two principles to implement web services exist:

SOAP The Simple Object Access Protocol (SOAP) [GHM*03] is a protocol for exchanging structured messages with a web service. The structure of these messages is usually described by the Web Services Description Language (WSDL) [CCMW01]. Both SOAP and WSDL are XML-based language.

REST Roy Fielding was the first to introduce the Representational State Transfer (REST) in his Ph.D. thesis [Fie00]. The basic principles behind REST are

1. identification of resources using URI
2. manipulation of resources using standard HTTP operations (GET, POST, PUT, DELETE)
3. self-descriptive messages, e.g., Plain Old XML (POX) and the JavaScript Object Notation (JSON) [Cro06], etc.

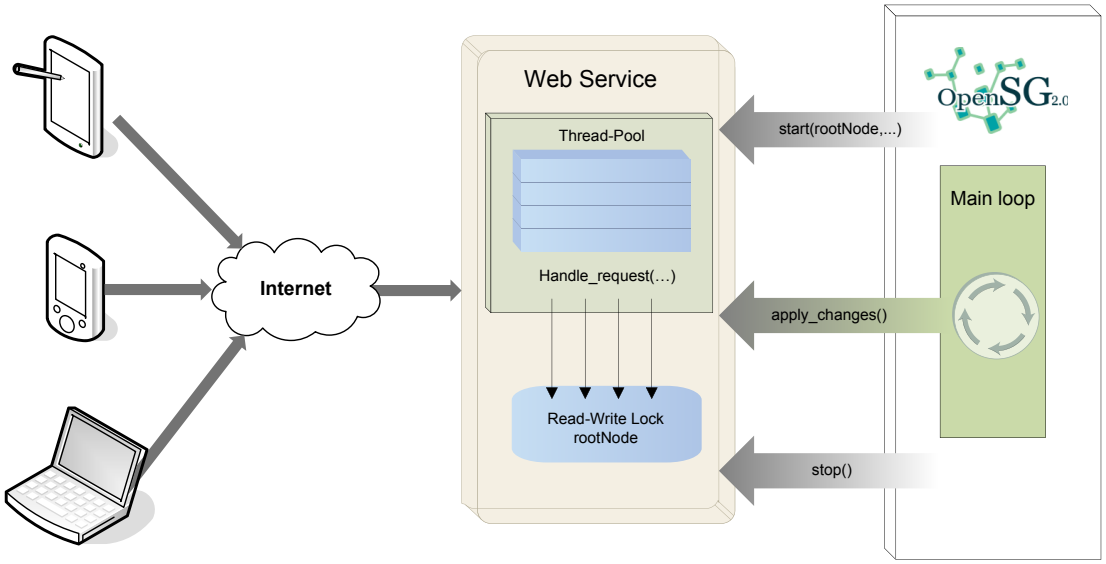


Fig. 4.3 The integration of a service-oriented architecture via a web service in the scene graph system OpenSG offers many possibilities. Three method calls are sufficient to export a scene graph of an arbitrary OpenSG application. This minimally invasive change of the application transforms closed visualization into an open service-oriented platform.

4.1.3.3 Web Services and Scene Graphs

Integrating a web service into an OpenSG application allows to access and modify the contents of the scene graph during runtime without recompilation of the application. These features offer many new possibilities in interacting with the OpenSG application and the scene graph, some of them are discussed in detail later in this section. A system overview is sketched in Figure 4.3.

The main features of our web service are querying the contents of the scene graph, adding and deleting of nodes and changing properties of nodes. Referencing of node data – to use it in multiple nodes – is also supported by the web service API. It is also possible to download the whole scene or sub graphs of it as a single file from the web service in various file formats.

RESTful web service API

The API to access the web service is designed in terms of the Representational State Transfer (REST) architectural style as described in Fielding's thesis [Fie00]. Reasons for choosing the REST style for the web service API include the simplicity of REST and that the hierarchical nature of a scene graph fits nicely to the resource oriented URL style of RESTful web services. Also, the four HTTP methods GET, POST, PUT, DELETE used to access the RESTful web service map very well to the most common operations on the scene graph: reading, creating, updating and deleting of nodes and values.

For implementing the web service, the GNU library libmicrohttpd[5] is used. As the data interchange and resource representation format the JavaScript Object Notation (JSON)[6] is used

[5] http://www.gnu.org/software/libmicrohttpd/
[6] http://www.json.org

by our web service. It is a lightweight, human readable and easy to parse data interchange format based on a subset of the JavaScript programming language.

There are three types of representations that are returned by the web service: FieldContainer, List and Data.

Data Data resources contain the value of primitive OpenSG data-types like integers, floats, vectors or matrices as a string. Additionally the data-type itself is also specified as the name of its OpenSG class. In the following example the field travMask from the FieldContainer example below is represented:

```
{
  "content": "Data",
  "type": "UInt32",
  "value": "4294967295"
}
```

List Lists are ordered collections of resources that are accessible by index. The List representation contains the type of its items and a list of locations to the item resources. The following example shows the JSON representation of the children field, again from the FieldContainer example below:

```
{
  "content": "List",
  "type": "NodePtr",
  "items": [
    "/scene/root/children/0",
    "/scene/root/children/1"
  ]
}
```

FieldContainer FieldContainers are resources that itself contain locations of other resources which are accessible by name. Every FieldContainer has an unique id and a type, which is the name of its OpenSG class. All sub-resources of the FieldContainer are accessible through the fields attribute of the JSON object, which contains a mapping from the field name to the location of the corresponding resource. Here is an example of a FieldContainer that represents an OpenSG Node object located at the root of the scene graph – it contains locations to both the travMask and the children resources of the above examples:

```
{
  "content": "FieldContainer",
  "id": 292,
  "type": "Node",
  "fields": {
    "attachments": "/scene/root/attachments",
    "travMask": "/scene/root/travMask",
    "core": "/scene/root/core",
    "children": "/scene/root/children"
  }
}
```

All OpenSG data-structures accessible in the scene graph can be represented with these three basic types. For accessing the resources, the four HTTP methods GET, POST, PUT and DELETE are used. To support AJAX web applications on different domains to access the web service, the method OPTIONS is also partially supported. Web browsers use this method to au-

thenticate cross site requests and web services must respond to OPTIONS requests as described by the W3C Cross-Origin Resource Sharing working draft[7] to allow the cross site requests.

The semantics of the different HTTP methods when accessing resources of the web service are now described in detail.

GET

GET requests are used to get the representation of a resource. All valid URLs of the web service can be called with the GET method and return one of the three representation types presented above. A GET request does not change any data in the scene graph.

If a GET request is made with just the URL to the resource, one of the JSON representations is returned depending on the type of the specified resource. Additionally, a filetype URL query with a file extension can be specified to download the resource - including sub-resources - in the specified file format. This applies only to FieldContainer resources with the type Node, for all other resources the query is ignored.

If the called URL is not found in the scene graph, an error message with the HTTP status code 404 - Location not found is returned by the web service, otherwise the response has the status code 200 - OK.

Examples for GET requests:

GET /scene/root/core

Get the JSON representation of the core of the root node

GET /scene/root?filetype=osb

Download the root node (and all its children) as OpenSG-Binary file

POST

The POST method is used to either appending new nodes to resources of type List or to update the value of a Data resource. According to the HTTP/1.1 specification [FGM*99], the POST method is the only non-idempotent method. In other words, only for the POST method it is allowed that multiple identical requests lead to different results (as when appending to a list).

Updating of Data resources is also handled with the POST method because POST can handle big amounts of upload data which may occur when updating textures or vertex arrays.

For appending a new node to a List resource, the type of the new node must be specified with the type URL query. If the specified type is a OpenSG node, a Group core is automatically added to prevent nodes without a core. On the other hand if the specified type is a OpenSG node-core, then an empty node is created and a core of the specified type is added. The response to such a request contains a JSON string with the location of the new node and additionally a HTTP header field Location with the full URL to the new node.

When updating a Data resource, the new value of the resource must be specified in the entity body of the HTTP request as a JSON string. After a successful update, the JSON representation of the updated resource is returned - the same as a GET to the resource, but with the updated value.

In case a new node is successfully added to a List resource, the response has the status code 201 - Created. If a Data resource is successfully changed, the status code of the response is 200 - OK. There are multiple possible errors that can occur using this method. A 400 - Bad request status code is returned if the type URL query for appending a new node is missing or the given type is unknown, or if the entity body of the request is not a valid JSON string when updating

[7] http://www.w3.org/TR/cors/

a Data resource. For unknown resources – when the URL is not found in the scene graph – the status code 404 - Location not found is returned. Finally, if the called URL does not point to a List or Data resource, the response has the status code 405 - Method not allowed.

Examples for POST requests:

POST /scene/root/children?type=Geometry
Appends a new node with a Geometry core to the children of the root node.

POST /scene/root/children/1/core/matrix
With "1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1" as the entity body, sets the matrix of the second child's core to the identity matrix.

PUT

Similar to POST, a PUT request can be used to create new resources. But instead of appending a resource to a list PUT replaces an already existing resource or inserts the new resource at a specific index in the list.

To replace an existing FieldContainer resource with a new one, the PUT request either needs a type URL query with the type of the new FieldContainer to create - similar to POST - or a location URL query with the location of another FieldContainer. In the latter case, the specified location gets referenced from the location of the PUT request. This allows for example to instance geometry or to share a single material between multiple geometries.

If the request is successfully processed, the response has the status code 200 - OK. As with the POST method, there are multiple error cases for which different error codes are returned. The status code 400 - Bad request is used if the required type respectively location URL query is either not existing or the specified value is not valid. Also the 400 - Bad request status code is returned if the index specified in the URL is not valid when inserting a new resource in a List resource. Like the other methods, a PUT request returns the 404 - Location not found status code if the called URL is not found in the scene graph. If the called URL does not point to a FieldContainer resource or an item of a List resource, the status code 405 - Method not allowed is returned by the web service. Some OpenSG containers do not support inserting items at a specific index, in that case a response with the status code 500 - Internal Server Error is returned.

Examples for PUT requests:

PUT /scene/root/core?type=Geometry
Replaces the core of the root node with a new Geometry core.

PUT /scene/root/children/0/core?location=/scene/root/core
Sets the core of the first child to the root's core. Both nodes share the same core after this request.

PUT /scene/root/core/properties/8?type=GeoVec2fProperty
Creates a new GeoVec2fProperty and assigns it to the ninth item in the root's core properties.

DELETE

The last method used by our web service, DELETE, is used to delete items from a List resource. If the DELETE request is made to the URL of the List resource itself, all items of that List are deleted. But if the request is made to one of the items in a List resource, only that item is removed from the List.

If the item is successfully deleted the response has the status code 200 - OK. In case the called URL is not found in the scene graph a response with the 404 - Location not found status

code is returned. Because DELETE is only allowed for List resources or for items of a List resource, the status code 405 - Method not allowed is returned if the URL points to any other type of resource.

Examples for DELETE requests:

DELETE /scene/root/children/0
Removes the first child of the root node.

DELETE /scene/root/children
Deletes all children from the root node of the scene graph.

Integration overview

Our web service is designed to be easily integrable into every OpenSG application with only a few changes to the source code. The public interface of the OSGWebservice class consists only of three methods, which is everything needed to get the web service running in an existing OpenSG application. The three methods are now presented in detail.

```
bool start(FieldContainerMTRecPtr root,
           UInt32 aspect,
           UInt32 port);
```

This method is used to start the web service. The parameter root specifies which resources are exported through the web service. The FieldContainer specified as the root parameter is accessible through the web service at the location /scene. Typically this is the root node of the scene graph, but it may also be an OpenSG Viewport or any other FieldContainer. In case it is an OpenSG Viewport, additional properties of the OpenSG application are accessible through the web service besides the scene graph, like the camera of the specified Viewport.

OpenSG provides so called aspects for multi-threaded operations on the scene graph. Threads can operate independently on different aspects of the scene graph without interfering each other. At some point, the aspects can be synchronized and every thread then sees the changes of the other threads. The aspect parameter defines the aspect on which the web service operates. This parameter should be set to some unused aspect which is then exclusively used for the changes through the web service.

Finally the port parameter specifies the port on which the web service listens for incoming HTTP connections. The default value for this parameter is 8080.

```
void apply_changes();
```

This method should be called at regular intervals to sync the changes made through the web service into the application's main thread. Changes are also synced in the other direction, from the main thread into the web service, by this method. So it is important to call this method regularly to keep the application and the web service in sync. Typically this method will be called during the render function of the application every frame.

```
void stop();
```

Finally, this method can be used to stop the web service.

Threading issues

Depending on an user defined setting, the web service may use multiple threads to handle incoming HTTP requests. All of the threads work on a single OpenSG aspect (specified when starting the web service), so they need to synchronize their access to this aspect. To synchronize the threads, access to the aspect of the web service is protected by a Read-Write lock. Multiple threads can simultaneous read, but if one thread writes to the aspect no other thread can access the aspect. Every subsequent access to the aspect waits until the current write operation finishes. But even with this synchronization in place it is possible that threads overwrite each others changes to the aspect before the changes get synchronized to the main application. Therefore, to ensure reliability, currently only one writing change is allowed for every call to apply_changes(). Requests that only read data are not affected by this limitation.

Extensibility

Currently the web service provides access to an OpenSG application's scene graph. But users of the web service, the application programmers, may also want to provide additional functionality for their application through the web service. To satisfy their needs, an additional API is designed, which allows application programmers to map function calls of their application to user defined URLs accessible through the web service. The application programmer just needs to register a callback function at the web service with a specific user defined URL. The web service stores the function pointer to the provided callback function and the URL, and every time the URL gets accessed through the web service, the stored function gets called.

For example the application programmer can map a function with the signature string buildHouse(string input) to an URL.Whenever its URL /actions/build_house is accessed through the web service, the corresponding function gets called by the web service. All input from the HTTP request is passed to the function as a string and the return value from the function is used as the response to the HTTP request.

Using this mechanism application programmers can easily extend the functionality of the web service specifically for their application's needs.

4.1.3.4 Advantages of Web-Based Linking

The presented approach offers a variety of possibilities. First of all, the approach realizes a simple integration of the scene graph system OpenSG into already existing applications. While it is still possible to realize an integration via dynamic or static linking, a web-based linking via web services enables a simple integration across different languages. As HTTP requests are the common denominator of web-services, they are understood by almost all programming languages. Therefore, the integration of an OpenSG-based visualization into e.g. a Java application is no problem anymore. It offers the possibility to use the right tool respectively programming language of the job.

Furthermore, the web-based integration of OpenSG offers new and simple interaction possibilities. As the camera, a part of the via web-service exported scene graph, can be edited and modified via web browsers new human-computer interactions are possible. For example, a CAVE environment can be controlled and explored via a smart-phone. With adopted, specialized applications this has been possible before, but with our approach it can be realized much

easier: a few lines of code start the web-server, the web server takes the root node of the scene graph and automatically exports it.

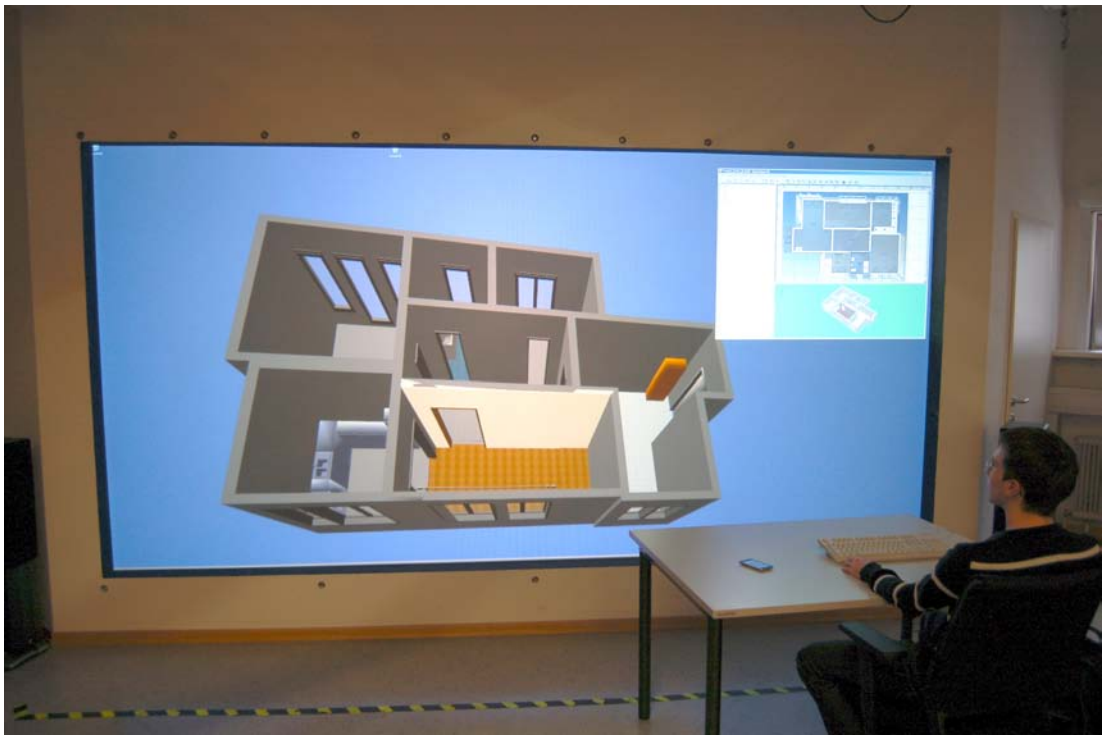


Fig. 4.4 A plug-in to the interior design application SweetHome3D transfers all changes made to the 2D house plan through our web service to an OpenSG application which displays a 3D representation of the house plan on a big tiled display.

4.1.3.5 An Example Application

To demonstrate and test our web service we have implemented a plug-in to the open-source interior design application SweetHome3D[8] (see Figure 4.5). The Java application SweetHome3D runs on a variety of platforms. It's main features are the ability to draw a 2D house plan and to place furniture on that plan. 3D geometry is generated from those plans and there is a 3D preview of the house plan included in the user interface. A plug-in API is available to access all data that defines the house plan like position and size of walls, materials, details of placed furniture, etc. From this data the geometry for a 3D representation is calculated by the plug-in – which is very similar to what the renderer included in SweetHome3D does for displaying the 3D preview.

With a menu entry our plug-in can be connected to an OpenSG application through our web service. Once connected, our plug-in transfers all changes made in SweetHome3D in real time to the OpenSG application, where it adds the geometry of the 3D representation of the 2D house plan to the scene graph (see Figure 4.4 and Figure 4.6).

[8] http://www.sweethome3d.eu

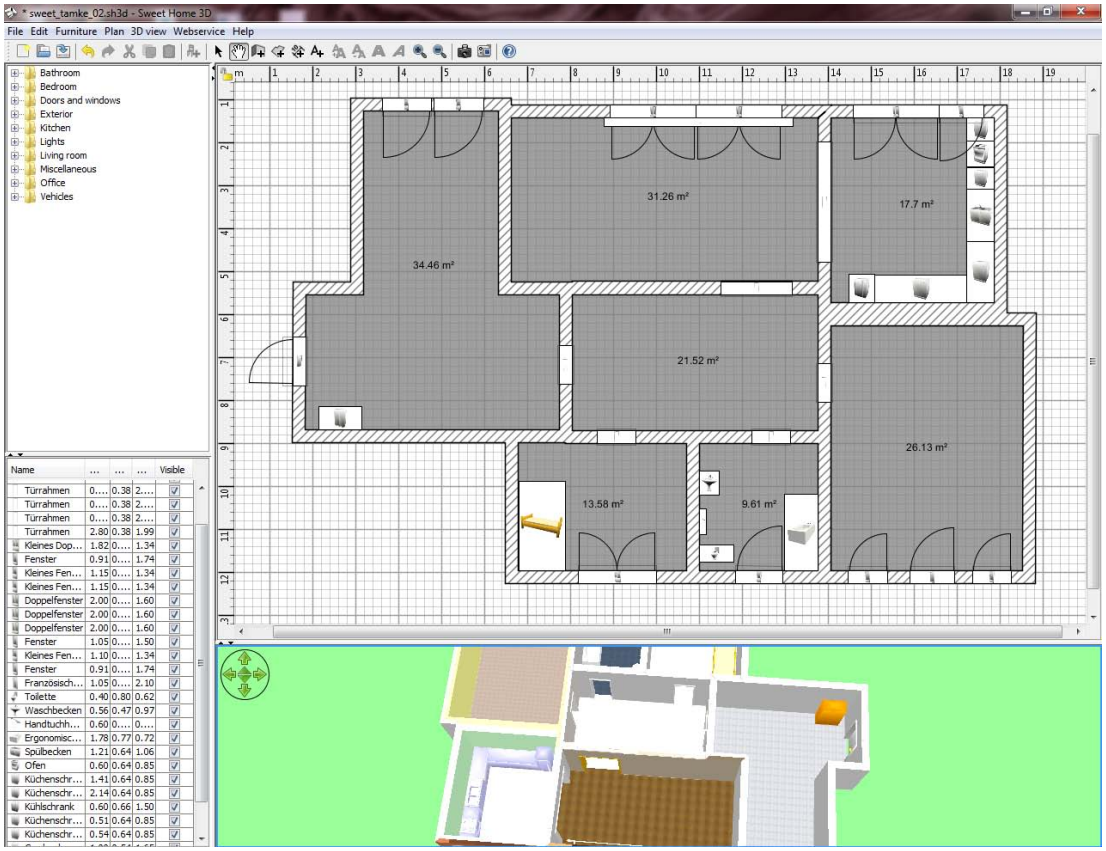


Fig. 4.5 The Sweet Home 3D software connected to the OpenSG application via the web service plugin.

Except for starting the web service and exporting a root node in which the SweetHome3D plug-in creates the house geometry through the web service the OpenSG application does not need any additional code for the integration to work.

All of the features of SweetHome3D are transferred correctly: The geometry, the colors and the materials with textures as well as the imported geometry representing the interior are displayed in the same way as they are planned in SweetHome3D. Additionally the graphical representation can be much improved over the simple 3D preview of SweetHome3D. The 3D preview does not have advanced features like shadows, etc. which can easily be included in the OpenSG application displaying the 3D representation of the house plan.

The advantage over the included 3D preview is the flexibility of OpenSG. The OpenSG application can display the 3D representation of the house plan in a virtual environment like a CAVE or on a big tiled display. At the same time the model can still be modified in its semantically enriched form of the SweetHome3D editor.

4.1.3.6 Comparison

Web-based Architectures

Currently there is an ongoing discussion "REST vs. SOAP" in the SOA world. A comprehensive comparison between RESTful vs. "Big" Web services can be found in an article by Pautasso et al. [PZL08].

As described by Weerawarana et al. [WCL*05], REST services and SOAP services should not be treated as different implementations alternatives for the same solution:

> *...As a rule of thumb, REST is preferable in problem domains that are query intense or that require exchange of large grain chunks of data. SOA in general and Web service technology ... in particular is preferable in areas that require asynchrony and various qualities of services ...*

Distributed Graphics

Bacu et al. describe a render cluster using the Chromium[9] software [BMG08]. A scene graph is rendered on multiple clients by transferring the low level OpenGL commands over the network. For transferring the render results back to the server a video streaming format is used. Using the Chromium software it is possible to distribute OpenGL based applications without code modifications but in general the amount of data transfered over the network is much greater than using OpenSG. And especially for a CAVE setup a customized application is essential.

Integrated Approaches

Zhang and Gračanin propose a framework using web services to combine multiple input providers into one 3D portal application [ZG08].

Web services in combination with a 3D city visualization are described by Hagedorn and Döllner [HD07]. The service provides high quality rendered images independent of the client devices graphics capacities. Their framework is specialized on using high-level geoinformation services without cluster support.

A similar combination of web service and scene graph is described by Behr et al. [BDR04] for the Instant Reality framework[10]. It also uses OpenSG as rendering back end and X3D for the scene description. A difference to our approach is the use of SOAP instead of REST. The flexible Instant Reality framework offers a great render performance and can also be used in a CAVE environment. But it is not possible to integrate it into existing applications on the C++ level.

4.1.3.7 Insight

In this section I present an integrated approach of a web-based scene graph application interface. In contrast to previous work our solution is

multi-threaded All components of our system are multi-threaded. The scene graph system OpenSG supports multiple threads and the web service can handle requests in parallel.

[9] http://chromium.sourceforge.net
[10] http://www.instant-reality.org

multi-user capable The implemented synchronization mechanism ensures a consistent scene graph, even if multiple users send several HTTP requests at once.

platform independent OpenSG as well as the web service implementation (libmicrohttpd) are platform independent; i.e. all major platforms (MS Windows, Linux, Mac OS) are supported.

minimally invasive Any existing OpenSG application can be upgraded to a web-based service-oriented application by adding a few lines of code. Due to the clean design of OpenSG only the web server's start-up method and its synchronization routine have to be inserted into the existing application. Therefore, the number of changes to an existing application normally involves less than ten lines of code. This minimally invasive modification transforms an OpenSG application into a web server.

adaptable / accessible On the client side only minimal adaptions are needed. The effort required to build a client to a RESTful service – the technique used to transform OpenSG into a service-oriented architecture – is very small as developers can begin testing such services from an ordinary web browser. Due to web support on all platforms and in all languages (Java, C/C++, ...), the integration of an OpenSG-based visualization is no problem anymore. Wrapper and interface code to overcome differences in language and communication is not needed.

All things considered, the presented solution enriches web-based interaction and visualization. The combination of OpenSG and a web server has a beneficial effect on service-oriented scene graph systems, on the integration of immersive visualization environments into existing applications, and on (browser-based or web-based) user interfaces in virtual/mixed reality.

Fig. 4.6 The SweetHome3D output is transferred with our web service to an OpenSG CAVE application which lets the user walk through a 3D representation of the house plan.

4.1.4 More Visualization Engines

4.1.4.1 Instant Reality

Instant Reality[11] is a graphics framework developed and maintained by the Fraunhofer IGD, Darmstadt. It is a high-performance framework that supports classic Virtual Reality (VR) as well as advanced Augmented Reality (AR). The framework provides a very simple application interface but includes the latest research results in photo-realistic rendering, user interaction and immersive display technology. It is available for a wide number of hardware and software platforms (Windows, Linux, Mac OS X). The Instant Reality player is freely accessible for non commercial use.

The rendering module of Instant Reality is based on OpenSG. Therefore all the benefits described above can be used with the Instant Reality framework as well. The other modules are the dynamic scene management and manipulation system Avalon, the VisionLib, a flexible pipeline processor for computer vision tasks and a device and data-stream management system.

The system design includes various industry standards, like VRML and X3D (see Section 2.3), to ease application development and deployment. Program logic is scripted using JavaScript. Data-flow graphs, an extension to the X3D scene, can be used to define an application simply by connecting components. With a drag and drop interface even complex applications can be created by non-expert users. The final result can be deployed to many platforms from mobile devices to immersive environments.

Instant Reality supports most of the X3D components and also adds features which are not (yet) part of the standard. For example the definition of shader materials on an abstract level, the CommonSurfaceShader, implements the Blinn-Phong BRDF with normal mapping and a perfect specular component.

Network-interfaces are provided to incorporate application data at runtime. With HTTP and SOAP interfaces the running application can be controlled in a very flexible way. Many input devices are supported directly. For unsupported devices a C++ plugin API is also available to connect it to the framework.

3D content can be added from the most common tools via VRML or X3D exporters. The IGD provides a specialized exporter for Autodesk 3ds Max. Also many CAD formats are supported via data importers. A powerful conversion tool helps to optimize the 3D content for interactive visualization.

4.1.4.2 Game Engines

Game engines are a motivational project for many students in the field of computer graphics and beyond. Wikipedia has a list of over a hundred game engines[12]. Many of them are open source. The most powerful graphics engines are found where big money is involved: the games industry. And the creators of some successful game engines have made their software freely available for non commercial use, for education and training.

Here is a short description of three game engines which are currently popular also for serious visualizations:

Unity Unity Technologies development studio head quarters in San Francisco, developers spread around the world in North America, Europe and Asia. Unity is a game development

[11] http://www.instant-reality.org/

[12] http://en.wikipedia.org/wiki/List_of_game_engines

tool with state of the art graphics, animations, artificial intelligence etc. Its main focus is on optimized workflows and support for a large amount of target platforms.
The editor can be downloaded at http://unity3d.com/ for Windows and MacOS. Unity is available in two variants: the Pro version and the free version that has some limitations in the range of features. With the Pro version the resulting games can be compiled and packaged to run on many platforms: Windows, MacOS, gaming consoles (Wii, XBox 360, Play Station 3), mobile devices (iOS and Android), Flash and Web (via plugin) and Linux (since version 4). The free version only supports Windows, Linux and MacOS as target platform. As usual an extra license is needed for game console support of the Pro version.
Included in the Unity editor is a tool called BEAST[13] to automate the process of light map creation. The creation of optimized texture coordinates for light maps is done automatically. The tool creates true global illumination lighting and bakes it into texture maps. It also supports the creation of light probes for dynamic light effects. To support shaders on multiple platforms Unity uses a dialect of the GLSL which is transcoded automatically to the target system.
Unity is open for modifications via scripting: JavaScript, C#, and a dialect of Python named Boo are supported. The game logic runs on the .NET platform Mono. The .NET libraries are available within all of the supported scripting languages. As the projection matrix of the camera can be freely controlled by scripts, a modification to support unusual environments like CAVEs™ is easily realized.

CryENGINE The CryENGINE is developed by CryTek, Frankfurt. The most recent free version is the CryENGINE 3 SDK which was released in August 2011. It is available at http://www.crytek.com/cryengine. With the commercial version games can be deployed for Windows, XBOX 360, PlayStation3 and the Nintendo Wii U.
With the so called Sandbox Editor the developer can modify all aspects of the engine. This includes creation of graphics elements, a particle system, terrain editing, character animations, artificial intelligence, the game logic and the sound system. Included with the engine is a multithreaded physics engine. Shaders are abstracted to be platform independent. CryTek calls it "Uber Shader" technology. A special graphics feature of the CryENGINE is the global illumination method described by Kaplanyan [Kap10]. The engine uses LUA for scripting. Sandbox offers the Flow Graph Editor, a visual editor for creating scripts without typing text.
Juarez et al. proposed an implementation for using the CryENGINE 2 in a CAVE™ environment [JSB10]. But they could only use a symmetric view frustum, so the game did not react correctly on head movement. Version three of the CryENGINE offers a stereo view frustum and also code for the calculation of an asymmetric frustum is included. However, there is no direct access to projection matrix from the LUA scripts.

Unreal Engine The Unreal Development Kit (UDK) is developed and maintained by the North American publisher Epic Games. Like the other engines mentioned earlier it is freely available and can be downloaded at http://udk.com/. Support for Windows, MacOS, Linux, XBOX 360, PlayStation3 and iOS mobile devices (iPhone, iPad, iPod).
The engine offers all that is necessary to build a professional computer game including state of the art graphics with shaders, realistic lighting, particle effects etc. There is a powerful character animation engine that supports realistic facial animations. An artificial intelligence system is integrated to generate vivid scenes with elements that move autonomously. For physical animations the NVIDIA PhysX engine (formerly Ageia) is used. UDK uses its own

[13] http://gameware.autodesk.com/beast

scripting language UnrealScript. Scripts can be programmed also in a visual editor called Kismet.

The engine supports a stereo view frustum, but no direct access to the projection matrix via scripting. Some work has been proposed by Jeffrey Jacobson et al. to make the engine work in a CAVE™ environment using VRGL [JLRLC05]. VRGL was developed by Willem de Jonge and it modifies the projection matrix of the game (and any OpenGL application) by exchanging the OpenGL driver library. This work is not supported any more and will not work with the current version of the UDK.

4.2 Setups/Environments

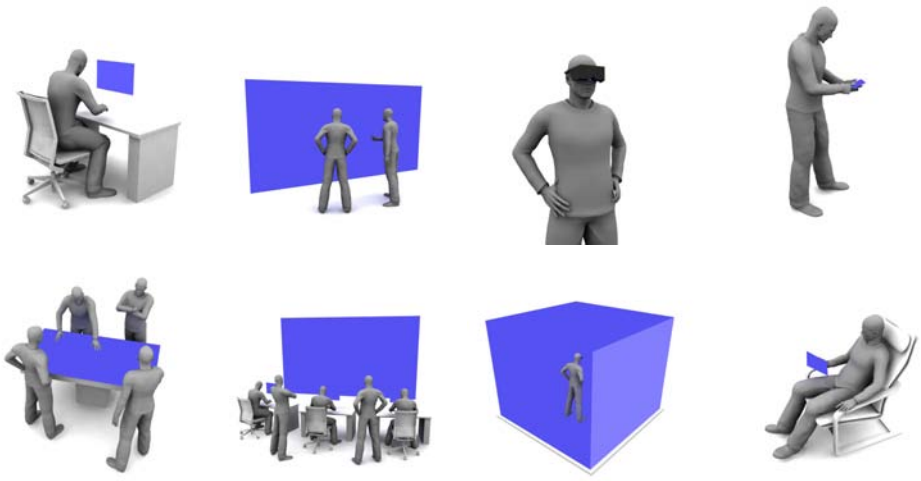


Fig. 4.7 Different environments for interactive visualizations: Desktop PC, table display, large screens, head mounted display, immersive projection and mobile devices.

Interactive visualizations can be presented in many different environments. Environments range from the classical desktop setup over table displays and multiple screens to mobile devices. Some environments are more suitable for a large group of users working together on a project. Some of them enable an unique experience by letting the user immerse in the virtual world.

It is necessary to adapt the visualization software for the different environments. Also the input devices for user interactions may have to be adjusted.

4.2.1 Desktop Setup

Although the trend is moving towards tablets and smart phones as the standard device the majority of the computers in the world is still used in the traditional way: a 2D display is placed on the desktop controlled by a keyboard and a mouse (Figure 4.8). For laptops, the mouse may be substituted by a touch pad but the principle setup stays the same. The user is typically sitting on a chair in front of the screen.

This setup is mostly used by a single user. No more than two persons can gather around a standard sized screen (nowadays 19”-23”) with a resolution of 1920×1080 pixel. (Source: product range of displays at amazon.com and radioshack.com, May 2012) For presentations in front of a group of people a computer is often used with a projection system to get a larger screen size.

Since a few years also a secondary display is becoming more and more common for the office work space. Since common graphics hardware supports two output connections, the technical obstacle is minimal. But the use of both screens for one large presentation is quite new.

Fig. 4.8 Desktop: A single user is typically sitting on a chair in front of the screen.

Developing for the desktop setup requires no extra effort. Most of the developer machines will also be desktop setups. Applications do not require special input devices or unusual hardware.

4.2.2 Large Multi Screen Setup

A multi screen setup is the combination of several screens to compose one large screen. Not only the physical size of the screen is enlarged but also the amount of pixel per area. In this way it is possible to create screens with a very large resolution. The displays can be LCD monitors. But because of the construction of LCD panels it is not possible to compose a screen without visible seams. To create a seamless single display, projectors are used for the image parts (tiles). Tiles can be projected from the front (front projection) or from behind (back projection). By using a back projection the observer can stand directly in front of the screen without casting a shadow onto the projected image.

The projection screen can be a flat plane or it can also be a curved surface. Curved screen are often used for simulators to give the impression of a wide range image. The projected image

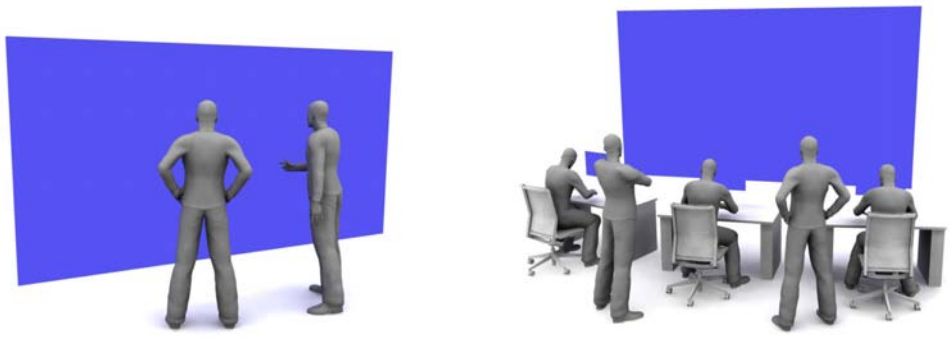


Fig. 4.9 Large screens with a high resolution can be observed by multiple persons.

covers the entire viewing range of the observer without showing distracting corners. This kind of projection is also used in large cinemas (IMAX) and planetariums (dome projection). Special lenses are needed to produce a focused image on the curved surface.

Specialized hardware solutions offer a multi screen setup controlled by a single computer. For example the Power Wall by Schneider[14] is composed of four or six projectors but can be used as if it was a single display. For a larger number of displays and higher screen resolutions the video signal has to be provided by multiple machines to distribute the rendering.

Since 2003 the Fraunhofer IGD in Darmstadt maintains a multiscreen project called HEyeWall. The HEyeWall now consists of 24 stereo projectors creating a 5 by 2.5 meter screen. With roughly 35 mega pixels (8400×4200) the resolution is very high even if the user is standing near to the screen. The projectors are able to show a 3D Stereo image in combination with shutter glasses.

A second HEyeWall prototype is located in Graz since 2007. It consists of 6 mono projectors with a combined resolution of approximately 4000×2000 pixel. In contrast to the HEyewall in Darmstadt, the screen is touch sensitive. The large plexiglass screen was coated with a thin silicone layer and infrared emitters send light from the borders to the inside of the plexiglass. The technique called frustrated total internal reflection (FTIR) [Han05] is used for multi touch detection. The FTIR technique is described in more detail in Section 4.5. More technical details to the Graz HEyeWall can be found in the PhD thesis of Marcel Lancelle [Lan11].

For tiles to create one big screen, the display size and position has to be measured and taken into account. To show the correct part of the image, a modified projection matrix is used. A seamless match from one tile to the other can be created in different ways. In a very time consuming process it is possible to calibrate the projections exactly so that the edge of one screen touches the edge of another screen. More often a small seam of one image part overlaps with the adjacent tile. If not handled, this seam would be identified by the doubled brightness. To avoid this effect, the tiles can be projected onto a grid of hardware blends. This requires a well constructed projection screen in order to hide the edges between tiles. A more common way is the soft edge blending. The tiles are arranged to overlap and the images are rendered with an overlaying blend mask. The blend mask reduces the brightness of the seam to match the brightness of the tiles. An advanced method was developed by Marcel Lancelle and

[14] http://www.schneider-digital.com/

Dieter Fellner in 2011 [LF11]. They achieve smoother results for corners in soft edge blend masks compared to commonly used blend masks by using first order continuous weighting functions.

One of the problems of creating a big screen out of many smaller displays is the incoherence in brightness and color. Although identical hardware is used it is not possible to get the exact same brightness for many projectors. And the effect gets worse the longer the projectors are in use. Also the color temperature changes slightly over time. This effect was reduced to a great amount by using DLP projectors instead of LCD projectors, because the liquid crystal displays were more prone to be color inhomogeneous. But also the color variation in different DLP projectors can be recognized by the human eye.

To solve these problems, additional calibration of brightness and color has to be applied. For example a brightness sensor could measure the lowest maximum brightness among all tiles and the render software would limit the displayed brightness to this value. The calibration has to be performed in relatively short intervals because the change of brightness is a dynamic process. It is connected to the lifetime of the projectors light source but also to the power supply. A similar calibration step can be done for the color adjustment. This leads to a reduction of the color range and the dynamic range of the composed image. Unfortunately the brightness of a tile is not constant for all view directions. A calibration will work for one position but fail for another. Tracking the user in front of the screen can compensate this effect but only for a single user.

The user may stand in front of the big screen (for example this is the typical usage of the HEyeWall). But multi screen setups can also be used in a surveillance center or similar scenarios (see Figure 4.9).

4.2.3 Immersive Environments

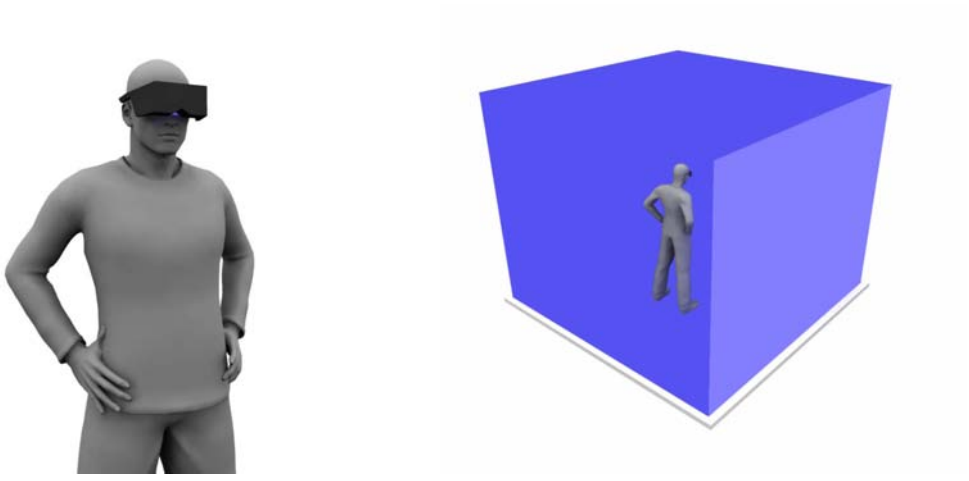


Fig. 4.10 In immersive environments the user feels himself being part of the virtual reality. This effect is achieved with a head mounted display worn directly in front of the users eyes (left) or large surrounding screens (right).

Immersive environments allow to enter virtual worlds. The user feels himself being part of the computer simulated reality. This is achieved by stimulating as many senses as possible. Obviously the sight is our main target in this work. But to make the illusion complete also hearing, touch, balance, acceleration and smell or taste should be addressed.

To give the user the feeling of being surrounded by the virtual world there are basically two options which are shown in Figure 4.10. The first is covering the eyes with a head mounted display. If the user moves his head the virtual camera is rotated accordingly. This is done with motion sensors or a tracking system. The user can experience a full 360 degree view or even walk around in the virtual world. The other option is to enlarge the screens and surround the user with them. Stereoscopic projection and head tracking are used to display the correct perspective per eye. This setup was developed at the University of Illinois, Chicago in 1992 [CNSD*92]. It is called CAVE which is a registered trademark of the University of Illinois Board of Regents. A typical CAVE^{TM} consists of four to six screens arranged like a cube. The idea was adapted to many variations including curved screens [AKMHP09] and tiled displays [DAA*11].

4.2.3.1 The DAVE in Graz

In 2005 the Institute of Computer Graphics and Knowledge Visualization at the University of Technology in Graz, Austria, built a new version of the *DAVE*. DAVE stands for definitely affordable virtual environment. Affordable means that by mostly using standard hardware components we can greatly reduce costs compared to other commercial systems. This includes especially the costs of updating to new rendering hardware. Room restrictions motivated a new compact design to optimally use the available space. The back projection material with a custom shape is stretched to the wooden frame to provide a flat surface without ripples [FHH07], [FHH03b].

The DAVE consists of three rear projected active stereo screens (left, right and front wall on that the images are projected from outside) and a front-projected screen on the floor (image for the floor is projected from above). Large mirrors are used to fold the light paths from the projectors to the screens in order to minimize the necessary room size. In Figure 4.11 the schematic layout is shown. At the right four of the eight render PCs are connected to two of the stereo projectors. The DVI cables are drawn in blue. At the top of the DAVE frame the four cameras for the tracking and their connections to the tracking PC are shown in turquoise. The synchronization of the projectors is hinted in red. In the background and to the right the master PC is visible connected via network cable to the other PCs.

The installation itself is a cubicle with 3.30 m wide walls. The floor is a front projection on a gray surface matching the brightness of the back projection walls. For each of the projection screens double DLP projectors (Cube3D^2 projectors with resolution of 1400 x 1050 from *digital IMAGE*[15]) use time sequential stereo displayed with 60 Hz for each eye. They are synchronized with custom hardware. Using the projector image buffers we can avoid a genlock in the graphics hardware. The synchronization signal is transmitted to the shutter glasses via infrared pulses.

Eight PCs are used for rendering the images inside the DAVE. A master machine controls the whole setup via network. Most of the input devices and a sound system are directly connected to the master PC. The one controlling master PC and the eight render PCs are normal off-the-shelf machines. Updating PCs or just graphics boards is inexpensive and keeps the performance up-to-date. The PCs and also a file server are connected via a gigabit network.

[15] http://www.digital-image.de

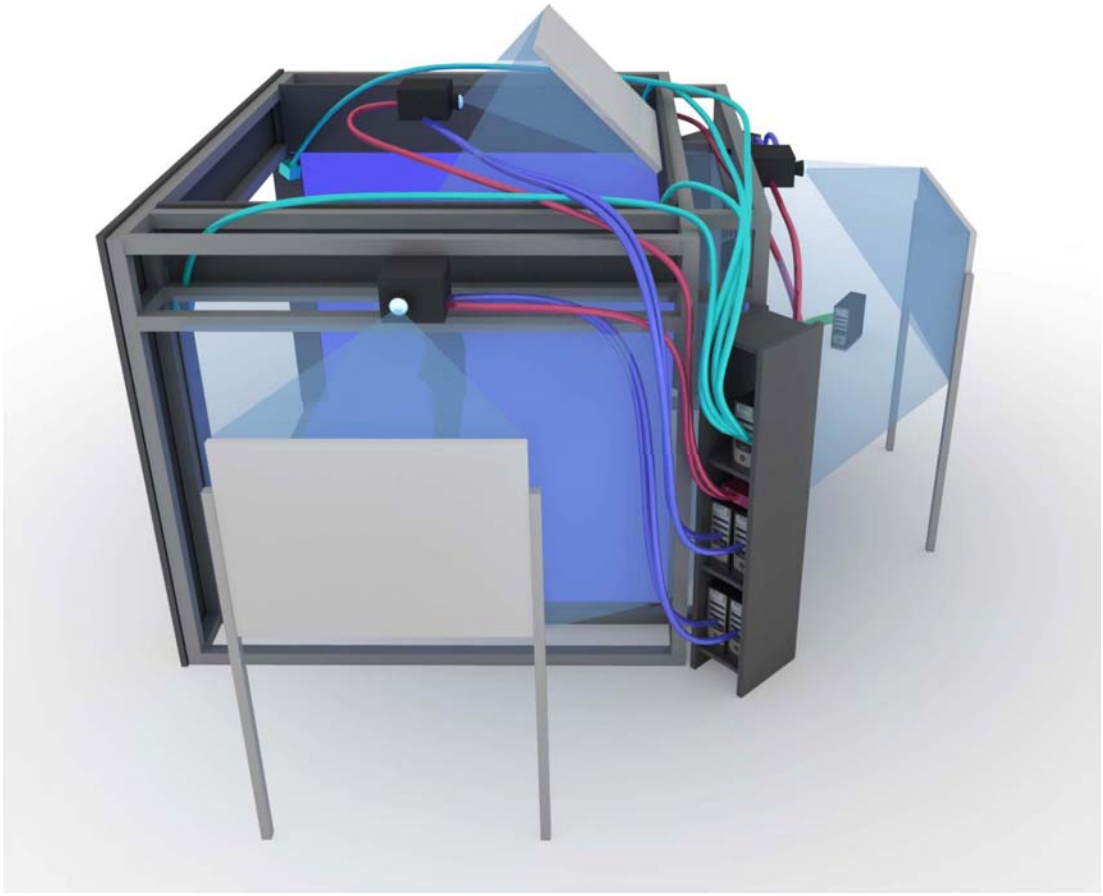


Fig. 4.11 This is the schematic layout of the DAVE in Graz. The video connections are drawn in blue, the four cameras for the tracking are shown in turquoise and the synchronization of the stereo projectors is hinted in red. Large mirrors are used to fold the light paths from the projectors to the screens in order to minimize the necessary room size.

We use a selfmade low-cost optical tracking system consisting of four cameras with attached infrared emitters. The cameras are connected to a dedicated tracking server. Passive reflecting markers are detected and triangulated. Since all markers look the same an identification is only possible with heuristic estimations or a fixed constellation of markers that we call *target*. For one target at least three markers are necessary for 6 degrees of freedom. Typically we use four markers for one target. In our current setup the cameras attached above the screen cannot see markers that are low and close to the screens. As an example the tracked volume is quite limited for foot tracking and the constant low power lighting setup prevents marker detection during fast motions. However, the most common task of tracking targets works well.

One pair of the shutter glasses is equipped with a target built out of four retro reflective markers for the tracking system. With about 45 Hz the users eye positions are computed to calculate a correct perspective view.

Beside the glasses, the input devices are also located by the optical tracking system. Electronic components from commercial gamepads (currently a Nintendo Wii controller) are used to realize wireless buttons. Since 2011 a Microsoft Kinect is installed as additional sensor.

Being totally immersed requires to travel through the virtual world in an intuitive way. This means that interactions, which would not have to be explained in the real world, should not be in need of an explanation in virtual worlds. When disregarding this basic principle the user interfaces create a barrier that compromises the immersion's impact. Also, the user's inhibition threshold towards immersive interaction can be lowered remarkably. This is especially important for virtual reality (VR) installations that are often used for demonstration purposes or in exhibitions [FLL*93]. Visitors are almost always first time users who should quickly be able to try out the system. A good example in a non-immersive environment is *Nintendo's Wii Sports Tennis*. As no user tracking is available, only arm and hand movements are interpreted but not the user's movements and position. The computer sets the player's position automatically. In this way, the input device as well as the user interaction can be kept simple and user-friendly. In our DAVE we realized these basic principles in immersive environment applications that require special attention due to their unconventional needs (also see Section 4.5).

Fig. 4.12 A photo of the DAVE installation in Graz, Austria.

Projection Matrix

To render the correct perspective over all screens we modify the projection matrix. An off-center projection is used. For the calculation of the projection matrix we need the position of each screen and the eye positions of the user. All coordinates have to be in the same coordinate system. In the DAVE the tracking system uses a different coordinate system than most of the applications. Therefore the head transformation matrix is modified to match the coordinate system of the applications.

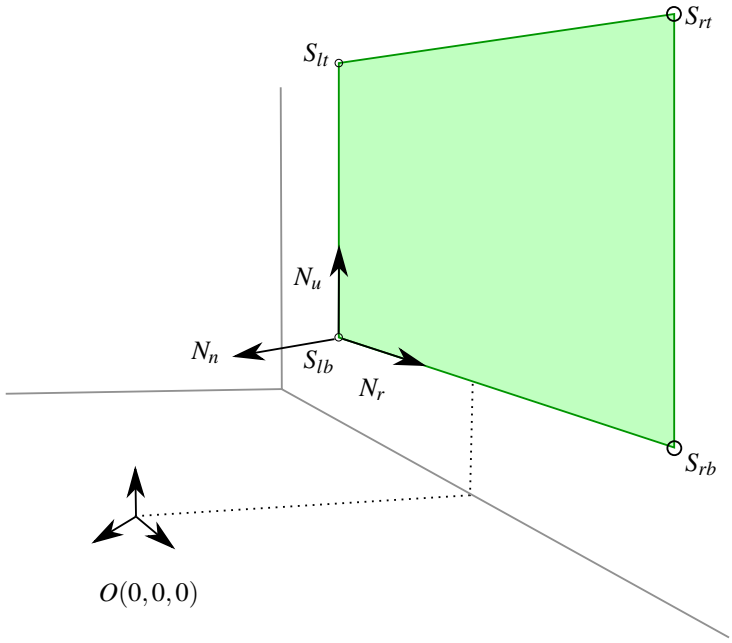


Fig. 4.13 The coordinate systems and the position of the example screen.

To demonstrate the calculation of the projection matrix we will concentrate on one single screen and one eye as shown in Figure 4.13. In the DAVE setup we would calculate eight matrices, two for each screen. The origin of the coordinate system O can be defined at an arbitrary position. We chose the center point at the floor of the DAVE to be the origin. The border of the screen is defined by the four points S_{lb} (left, bottom), S_{rb} (right, bottom),S_{rt} (right, top),S_{lt} (left, top). The four points have to be coplanar and they have to describe a rectangle.

In a preprocessing step we calculate the normal of the screen N_n as well as the normalized vectors to the right N_r and up N_u.

The necessary parameters for calculating the projection matrix for one screen are described in Figure 4.14. The eye position of the user Eye and the position of the screen S_l und S_r are described in 2d for simplicity. The shortest distance d of eye and screen is calculated by the scalar product of the vector $Eye - S_l$ and N_n:

$$d = (Eye - S_l) \cdot N_n \tag{4.1}$$

By using N_n for the calculation of d it is possible to define any screen orientation. Analogue to the calculation of d the distances l and r are computed with the normalized vector N_r and the difference vector $S_l - Eye$ and $S_r - Eye$ respectively:

$$l = (S_l - Eye) \cdot N_r \tag{4.2}$$

$$r = (S_r - Eye) \cdot N_r \tag{4.3}$$

As we are actually calculating the projection matrix for 3D we have to compute also the distance of upper and lower border in the same way. In the following we will call those parameters t and b. Furthermore the distances for the *near plane* n and the *far plane* f have to be defined. These distances describe the start and end plane of the camera frustum. They should

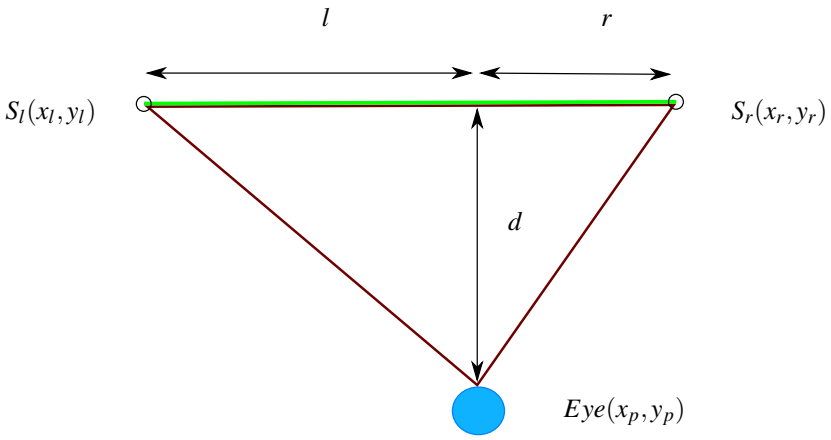


Fig. 4.14 Parameters needed for the calculation of the projection matrix.

be set depending on the scene. Especially the far plane should not be too far away because the larger the camera frustum the less accurate is the depth value calculation. Inaccurate depth values may lead to z-fighting (flickering).

The parameters can be added to the common calculation of a projection matrix like this:

$$M' = \begin{pmatrix} \frac{2d}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2d}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{4.4}$$

As the point of reference is the origin of the coordinate system, the projection matrix has to be corrected as follows: We need a translation and a rotation. The matrix for the rotation can be defined with the vectors N_r, N_u and N_n. The position of the user Eye is subtracted.

$$M = M' \cdot \begin{pmatrix} x_{N_r} & y_{N_r} & z_{N_r} & 0 \\ x_{N_u} & y_{N_u} & z_{N_u} & 0 \\ x_{N_n} & y_{N_n} & z_{N_n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -x_{Eye} \\ 0 & 1 & 0 & -y_{Eye} \\ 0 & 0 & 1 & -z_{Eye} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.5}$$

The projection matrix M has to be calculated per frame as the user constantly moves his head. In the DAVE eight projection matrices are calculated, two for each screen. The left and right eye positions are approximated by the head position coming from the tracking system. Additionally a calibration matrix is used to adjust the projection to the screen borders. This is described in detail by Lancelle et al. [LSF08].

Existing OpenGL based applications can be modified fairly easy to run in the DAVE if the source code is available. Some game engines allow the modification of the projection matrix as well. We added the described modification to the Unity engine. It is possible to modify the camera matrices using JavaScript. But since Unity uses a left handed coordinate system some

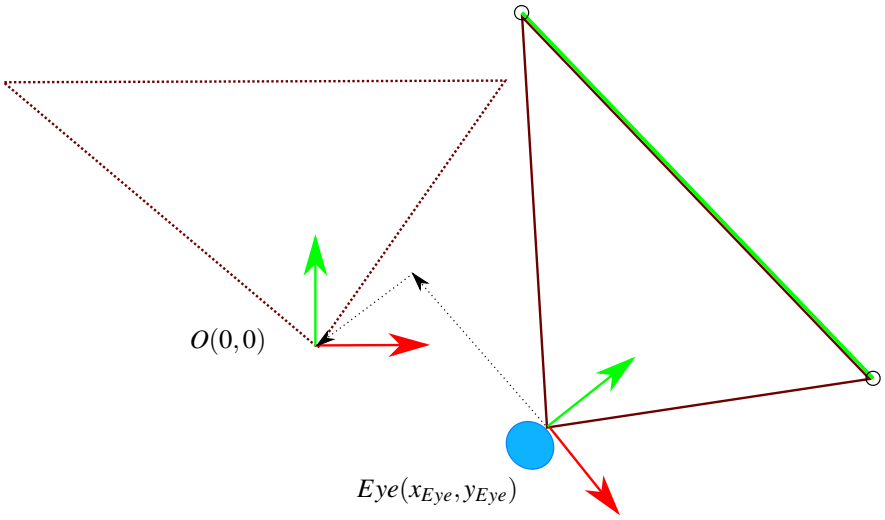


Fig. 4.15 Correction of the projection matrix: rotation and translation.

modifications were necessary. Basically the third column of the matrix is multiplied with -1 and the whole matrix is mirrored along the z axis.

$$M'_{LH} = \begin{pmatrix} \frac{2d}{r-l} & 0 & \frac{r+l}{l-r} & 0 \\ 0 & \frac{2d}{t-b} & \frac{t+b}{b-t} & 0 \\ 0 & 0 & \frac{(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.6}$$

Similar attempts with the Unreal Engine 3 and the CryENGINE 3 were not successful. Unfortunately it is not possible by scripting to arbitrary modify the projection matrix of those engines. Therefore it is not directly possible to use those engines in a CAVE™ environment.

Applications

Most of the applications in the DAVE use the scene graph API OpenSG or our self developed C++ library DAVELib. These libraries provide functions for program synchronization, perspectively correct rendering and input devices. The most suited applications for a CAVE™ are virtual walk- and fly-throughs. Scenes with precomputed radiosity are especially appealing. As the users movements are restricted to the size of the DAVE, an additional navigation concept is needed for large virtual worlds. For fly-through scenarios we use a Wii controller to fly in the direction the controller is pointing to. The controller is equipped with retro reflective markers. Since there is no back wall the scene automatically rotates to the front wall when pointing to the side while moving.

The game "PPRacer" has been partially ported to the DAVE using the DAVELib. While the 2D GUI elements could not be exchanged quickly, the main 3D content required little source code modifications to run in the DAVE. The penguin is directed by the users head position.

For an architectural scene based on OpenSG we added the Nvidia PhysX engine for placing furniture. The PhysX engine is also used for collision detection of the user with the virtual world.

The DAVE is used as a controlled test environment in the national funded projects "IMITATE" and "MOVING". In both projects guidance systems for pedestrians are evaluated. The DAVE allows to control environmental conditions, to measure the interaction of a given test person with the infrastructure and other passengers and to adjust static, dynamic and even mobile guidance systems as needed. This test environment makes it possible for the first time to measure the effects of situation-specific personalized guidance systems, investigate the acceptance of components of the guidance system and consequently adapt these in a cost-efficient way (see also Chapter 5).

Example implementations exist for Instant Reality and for the Unity game engine. Two finished undergraduate student projects are an underwater world with animated fish and a glider simulation where the user holds a marker in each hand to control the plane. In cooperation with the Laboratory of Brain-Computer Interfaces in Graz the DAVE was used for a simple navigation task, purely by thoughts (see Section 4.5.4).

An iPod is used to start applications and for frequently needed system controls. The iPod is used to access a web page on the controlling machine. Scripts are used to start the applications on each DAVE-PC. But also common tasks like projector control and setting the volume of the sound system can be accessed by the web interface.

4.2.4 Table Setup

The table setup is composed of a screen which is shown on top of a table surface. The screen can be a projected screen from a projector located above (front projection) or beneath the table (back projection). Back projection avoids shadows of the user but also means that the projector is build into the inside of the table. Most likely this will lead to a box-shaped table with hardly any space for the legs and feet of the users. Front projection with the projector located at the ceiling is easier to setup up. The other possibility is an LCD monitor that is let into the table surface. In that case the construction is thin and leaves space for the users legs. It is to mention that for the horizontal usage additional cooling may be needed for the LCD panel. Many monitors are passively cooled with air slots at the top of the casing.

The table setup is often combined with a touch interface. That makes user interaction much more convenient than using a mouse and keyboard on the table. With specific touch techniques also the projection technique has to be adapted. For example optical tracking and marker recognition is very complicated when using an LCD monitor. The first commercial available touch screen that combined optical tracking and image recognition with an LCD is Microsofts PixelSense technology. It was produced by Samsung and introduced as the Surface 2[16] display.

Some table setups combine the horizontal screen with a second display. The Bene IdeaWall for example has a second screen attached on one side of the table (see Figure 4.17). The HEyeWall in Darmstadt can be controlled by a touch table standing in front of the large tiled screen. Located in the center of the immersive laser projection system Elbe Dom[17] in

[16] http://www.microsoft.com/surface

[17] http://www.iff.fraunhofer.de/de/labore/elbe-dom.html

Fig. 4.16 Table: A screen which is shown on top of a table surface. Multiple users sharing one screen.

Magdeburg a touch table is used for navigation tasks and other user interactions [SMM*08]. For example a virtual factory is displayed on the table and on the dome walls. With the table interface it is possible to choose and place factory equipment.

Table setups have one important difference compared to other setups: the screen is not oriented to one user. At a regular table, at least four persons can share the display. Each one is looking from a difference direction to the screen. Today's operating systems are not able to handle this flexibility in orientation. Therefore, multi-touch user inputs demand some special reinterpretation and applications have to be modified in order to support this setup. With multiple users sharing one screen it is also necessary to support parallel user input. This can be tricky as for most of the multi touch screens it is not possible to distinguish between different users.

In 2007 Chia Shen et al. [SFWV07] authored an open letter to operating system (OS) designers in which they outlined the major requirements of the new user interface paradigm. A multi-user tabletop OS needs:

Fig. 4.17 This is the multi touch table of Fraunhofer Austria and Bene Office Furniture. A second screen is located at the back wall of the table. It is used to enrich applications run on the table display.

tabletop-friendly controls Multi-touch-friendly controls need to be larger as the touch-area of a fingertip is much larger than the single pixel defined by a mouse pointer.

tabletop-friendly graphics With several people around a multi-touch table, windows should be placeable at arbitrary positions and orientations. Furthermore, pop-ups, menus, dialogs and other child windows should be positioned according to their parent windows.

a multi-user, multi-input event architecture A multi-user and multi-input setting demands operating systems to handle the following situations:

- A single user may be touching multiple user interface (UI) controls simultaneously.
- Multiple users may be touching the same UI controls simultaneously.
- Multiple users may be touching different UI controls simultaneously.
- Multiple users may be entering text from different keyboards simultaneously.

4.2.5 Mobile Devices

Mobile devices are very popular not only since the introduction of the Apple iPad. Also smart phones become more and more a substitute for small laptops. In the last years the render performance of mobile devices has increased to a level which is not far from desktop PCs. The critical limitation for mobile devices like smart phones and tablets is the power consumption. Nevertheless all of the new products of the major smart phone manufacturers are equipped with a multicore processor and a powerful 3D graphics chip. More and more game engine developers are trying to port their software to phones and tablets. Unity projects for example can be

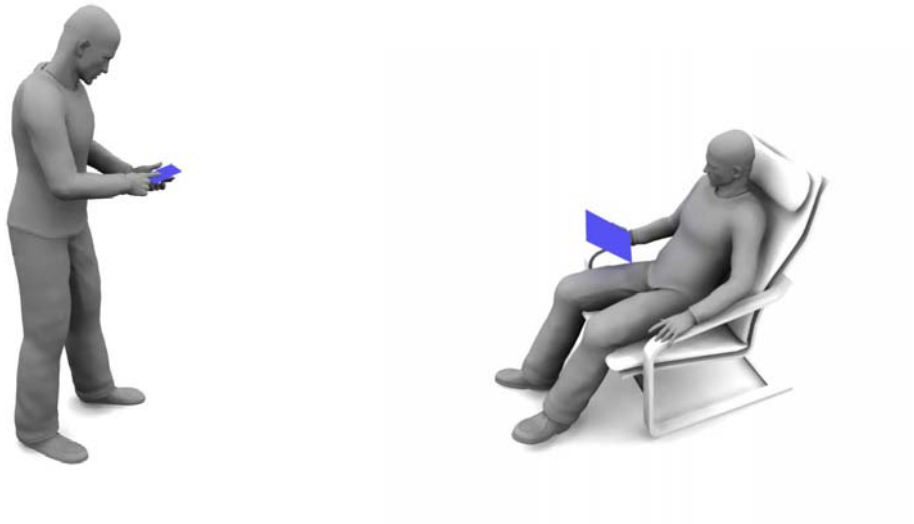


Fig. 4.18 Mobile devices are typically operated by a single user. Tablet devices become very popular as alternative to the desktop PC at home.

compiled for mobile devices with very few manual adjustments. Others use the HTML5 capabilities of modern browsers and develop mobile applications for frameworks like X3DOM and XML3D (see above).

A different option for visualizations on mobile devices is server side rendering. A render cluster produces the image and streams it to the device. As the network infrastructure is developed further and fast Internet is available almost everywhere the graphics can be delivered as a video stream in realtime. For fast user interactions this method can be combined with a proxy object for navigation. As long as the user interacts with the 3D scene it is rendered in a simplified way. In pauses when the user watches the image it can be updated by the server side rendering.

Mobile devices are typically operated by a single user. Their place of use is practically everywhere. And at home the tablet device becomes a comfortable alternative to the desktop PC (see Figure 4.18).

4.3 Preparation of 3D Content

The presentation of 3D content is an essential part of many industrial and scientific projects. Interactive visualizations are much more useful than images and pre-rendered videos. Virtual Reality with arbitrary view points and head tracking helps to speed up production and therefore saves money otherwise spend in physical moc-ups and test installations. But the creation process can be an important cost factor. Furthermore, the outcome of such visualizations has to compete with state of the art computer games.

3D models are created in many different ways and there is not the one and only single format. For construction of machines and architecture the planning is performed with tools like CATIA or AutoCAD. Freeform surfaces are defined for example with NURBS patches. For entertainment and marketing the main goal is to create appealing 3D scenes. In that context 3D content is created for a few viewpoints or a defined camera path. Typical primitives are polygons and Subdivision Surfaces but also other primitives and shader effects can occur. 3D models can be created using a laser scanner or with photogrammetry. For example, in the field of cultural heritage it is important to capture real objects in their current state. Scanned data can also be presented using point clouds or as a volume representation which is quite common for medical data.

However, it is not sufficient for interactive presentations to own the 3D content and the rendering software. The content has to be modified. In the best case the data only has to be converted to be understood by the presentation application. This task can be automated by conversion software, for example Deep Exploration[18]. But in most cases the content has to be modified beyond that. Optimized scenes for interactive rendering are hardly created automatically and the modification is a time and cost-intensive procedure. Suitable measures to reduce the time and cost effort are described in this section.

Software tools for 3D modeling are available in many variations and for all purposes. They all want to enable the creation of 3D shapes in a convenient way. Tool providers come up with good arguments to stand out from their competitors. Important software providers like AUTODESK[19] or DASSAULT[20] offer integrated solutions for modeling, animation, simulation and rendering. Their advantage is that 3D content can easily be exchanged between the editor and the renderer or the simulator for example. Other providers of creation tools are specialized in only one or a few aspects. For example MCNEEL offers the excellent NURBS modeling tool Rhinoceros[21]. And the sculpting tools Sculptris and ZBrush by PIXOLOGIC[22] are very popular among artists to create sculptures and virtual creatures. For applications like these it is important to create data which can easily be integrated into an existing tool chain.

4.3.1 Common Problems

3D models which are not initially created for interactive rendering have to be adapted. Otherwise they may be too complex for the rendering hardware in terms of memory consumption or interactive frame rates. Also the visual quality can be improved if the 3D model is prepared properly for real time rendering.

[18] http://www.righthemisphere.com
[19] http://www.autodesk.com
[20] http://www.3ds.com
[21] http://www.rhino3d.com
[22] http://www.pixologic.com

4.3.1.1 Conversion Problems

To use existing content in a rendering engine one has to start with choosing a file format. The format has to be exported by the creation tool and imported by the engine. Conversion software like Deep Exploration helps to overcome part of this task but some special features like shader effects may have to be imported separately.

4.3.1.2 Problems related to Construction and Planning Software

Computer-aided design (CAD) tools aim to create an exact representation of a 3D object. The main goal is to define all the details to actually construct the object. This goal may lead to the following problems for real time rendering:

- Unnecessary geometry, curves and labels are included in the 3D content. For example each single construction part is defined in detail, down to the last nut and bolt. Figure 4.19 shows such an object with unnecessary geometry.
- The geometry gets corrupted by the conversion process. Vertex positions change because of transformations, faces are duplicated. Mirrored faces suddenly appear black because their normal is flipped.
- Freeform surfaces with trim curves etc. may have to be tessellated depending on the rendering engine. The approximation of curves and surfaces can lead to numerical problems and gaps in the geometry.
- The conversion of freeform surfaces has to be done in an intelligent way. Automatically generated tessellations may lead to a lot of irrelevant polygons filling the rendering pipeline with unnecessary data.
- The material definitions of CAD tools may only consist of references to real materials without defining the visual properties. For the construction of an object it is sufficient to know the material. To render the object, either offline or online, a detailed material definition is necessary.

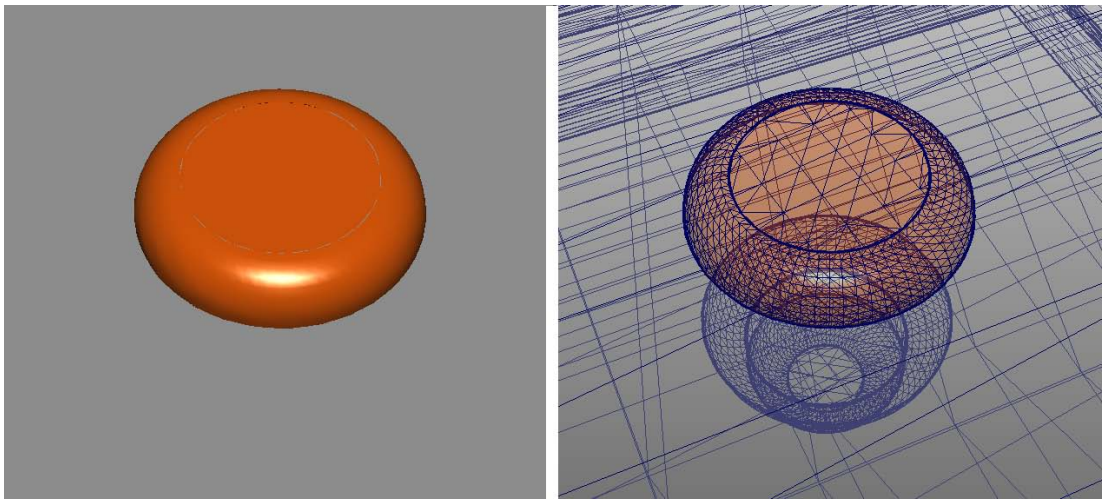


Fig. 4.19 A tiny bolt object is tessellated in high detail. The back side is never visible.

4.3.1.3 Problems related to Marketing and Entertainment Software

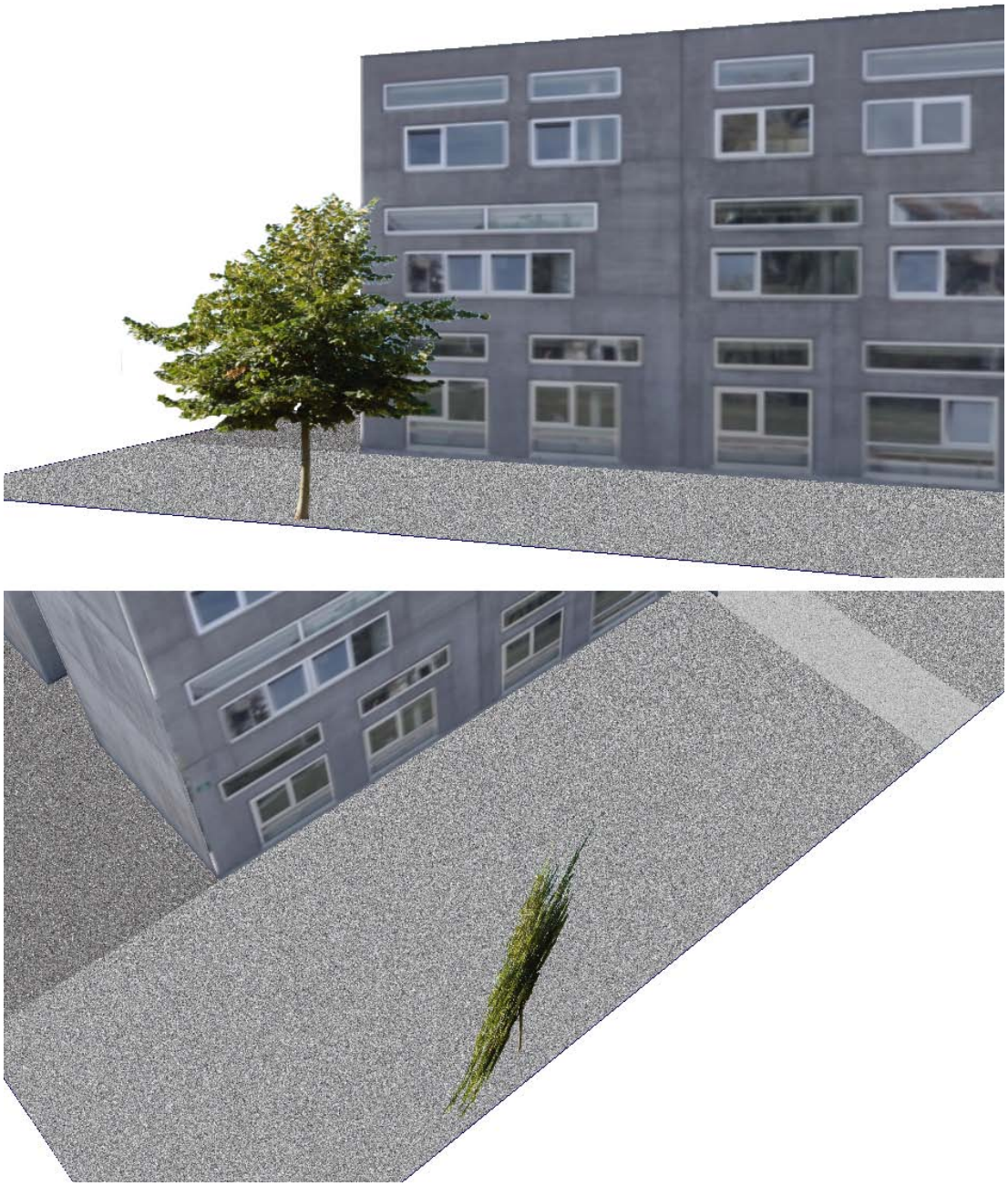


Fig. 4.20 The image of the tree looks good from one point of view. Other viewpoints are unsuitable.

Professional creation tools are regularly used for the creation of offline rendered images. Those images are needed for marketing purposes, for example, to advertise a new product. And the entertainment industry depends on 3D content for special effects, up to feature films that are more or less completely created in the virtual world. The following aspects may lead to problems when rendered in real time:

- The 3D content is often augmented with 2D elements. Those elements are not recognizable as such in rendered (2D) images and can hardly be combined with 3D content in an interactive manner if the observer can freely choose the point of view. In Figure 4.20 the tree is added as a 2D image. The image of the tree looks good from one point of view. For other viewpoints the tree looks like a part of a stage set.
- The scene may be lit by fake lighting effects which are added in a post-production step. Offline renderings are split into several layers and combined with a depth map. Parts of the image can be relit, repainted and changed to compose the final picture or video frame. Those effects can hardly or practically impossible be added to an interactive visualization. The light setup may have to be changed completely and the workflow for images effects is different.
- 3D objects use special materials which can only be used in the offline renderer. For example the human skin or even metallic car paints have to be used in a different way for real time rendering.
- Decorations are defined as extra geometry that shares the same position with base geometry. This can lead to so called z-fighting if not rendered in a special way. Figure 4.21 shows an arrow made of geometry which is set on top of a surface. With offline rendering (left image) the image looks okay but real time rendering and insufficient z-buffer resolution may produce flickering results (right image).
- The 3D scene contains too much geometry and the textures are often too large for real time rendering with acceptable frame rates.
- There are too many small objects in the scene organized in a scene graph with too many leaves. For example a model of a tree with each leave put into its own graph node produces a lot of overhead and slows down the draw traversal.

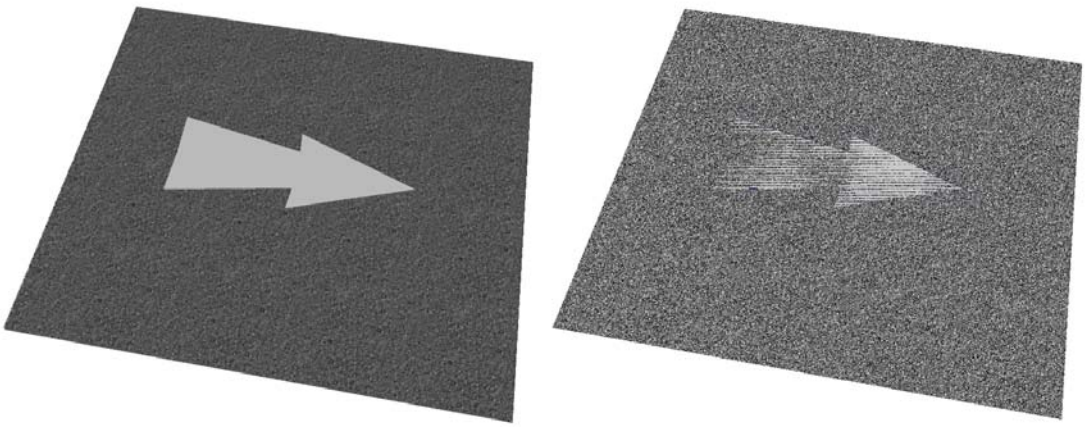


Fig. 4.21 The left image is rendered offline. The right image suffers from insufficient z-buffer resolution.

4.3.1.4 Reconstruction and Preservation

Acquisition methods like laser scanning and photogrammetry generate point clouds or polygon meshes. This kind of 3D data is aiming to reconstruct real world objects with the best possible quality. Parts which are well captured get a highly tessellated surface and other parts may be

very coarse or missing. Automatic tools can optimize the resulting content only depending on the captured data. That means it can reduce the amount data according to the structure (e.g. curvature). But it is hardly possible for automatic algorithms to decide, which parts are important in terms of higher level features. For those decisions it is necessary to add semantic information and to define a ranking of importance for the interactive rendering.

4.3.2 Practical Measures

The preparation steps that are necessary to render the 3D content in real time depend on the device on which it will be rendered. It can be a consumer desktop computer, a games console or a portable device like a tablet PC or mobile phone. In general, the device will most likely not be highly optimized for graphics applications in contrast to a system specialized to create 3D content.

The following describes possible procedures to prepare 3D content for interactive rendering:

4.3.2.1 Reduction

The reduction of data is an essential step to overcome limitations by poor graphics hardware. Geometry can be reduced in detail on parts which are not important. The importance of an object depends on the application. An object can be classified as unimportant because the part is hardly visible, because it is occluded most of the time or on the backside of a static object. On the other hand, an object can be unimportant because of the focus of the application. If a car is presented, it is inevitable to have all the details of the car in the scene. But if the visualization shows the concept of a car park, the cars are not important. Also if the geometry is usually far away from the observer it does not need as much details.

The open source mesh processing tool MeshLab offers some helpful procedures to repair meshes and to reduce the polygon count of a mesh while preserving the appearance. It implements the quadric edge collapse decimation proposed by Garland and Heckbert [GH97] in 1997. The commercial software Simplygon aims at game content tool chains. The software creates automatically levels of detail for polygon meshes, as well as for virtual characters.

Detailed geometry can be replaced by low resolution meshes augmented with a surface description in the texture. For example, the normals of a detailed model can be stored in a normal map. By using this texture on a low resolution mesh it is possible to render high quality geometry with far less polygons.

High resolution textures should also be resized to reduce the amount of data. Textures should be converted to a compressed format e.g. the S3 Texture Compression format [3]. Those compressed textures are decoded by the graphics hardware without much performance loss but they consume far less memory and the data transfer to the graphics hardware is faster. Again the question of suitable reduction depends on the application. The texture for a car radio system may be reduced if the visualization shows a model of a car in a large environment. On the other hand it could be very important if the visualization was created to promote the radio system and every detail should be visible.

For optimized reduction of geometry and textures in an automatic way it is essential to classify objects and parts. With semantic data it is possible to derive an importance level for parts and to formulate constraints for an algorithmic selection. In this way the reduction of geometry and textures can be performed by a machine with respect to the application.

4.3.2.2 Structuring and Organizing

An optimized organization of 3D content helps to further speed up render times. The key to fast render times is the reduction of state changes. To render some geometry, the graphics hardware has to be initialized to a state. For example, the material has to be initialized with the textures. Modern scene graph systems like OpenSG try to reduce the number of state changes by sorting the scene elements before they are send to the rendering hardware.

Using a scene graph for the 3D data makes it much easier to apply view frustum culling. This means, that parts (sub graphs) of the scene which are not within the camera view (frustum) are skipped at render time. Space partitioning methods help to speed up the rendering even more. Also the technique of occlusion culling can be used. It finds large objects (occluders) near the camera and identifies all objects which are completely occluded by them.

3D models can be rendered in different levels of detail depending on the distance to the camera or on the complexity of the scene. If objects are far away, they do not need much detail because they are too small for the resolution of the screen. In that case it is much faster to send a reduced model to the rendering pipeline without the observer noticing.

If the scene consists of many identical objects, they can share the same geometric information. This is called instancing. Not only the render performance can be increased with instancing but also the data size of the scene is decreased.

4.3.2.3 Pre-Lighting

To show appealing 3D scenes in real time it is common to calculate the lighting beforehand. It is possible to calculate a much nicer lighting offline than in real time. Figure 4.22 shows a model of a building. Using standard render techniques on the one-colored model makes it hard to recognize the geometric structure (top image). With pre-computed lighting baked into the texture of the model, the structure is much clearer (bottom image). The pre-calculation of the lighting can be done in many rendering tools. A light setup has to be created similar to the regular offline rendering process. For interactive rendering it is important that only position invariant lighting is pre-calculated. Specular highlights and reflections for example change their position if the observer moves around. Those effects should be added later with shaders. The lighting is commonly stored in a texture map. Another possibility is to write lighting values to vertex colors which may require a larger amount of triangles for accurate lighting. Textures containing the lighting are also called light maps. The quality of the light maps depends on the resolution of the texture map and also on the distribution of the surface to the texture. For example, if the texture map covers the whole object, but only a small part of it is lit, large parts of the texture will be black. This will unnecessarily consume large areas of the texture leaving only a small fraction for the lighting. Figure 4.23 shows two texture mappings. The left is created automatically. It contains a lot of unused space for the back surface of the wall panel. The right mapping is created manually and uses most of the pixels of the texture for the front surface. Creating those optimized mappings automatically would be possible by analyzing the scene geometry and recognize the visible parts of an object.

Tiled textures will need a different set of texture coordinates for the pre-lighting texture. If the lighting and the regular texture map are combined, the quality may suffer because of the reduced texture resolution compared to a tiled texture. In that case, it is better to use different texture layers for lighting and for texturing the object. This is a common procedure in rendering tools but may lead to problems while exporting to the desired file format for interactive rendering because not all formats support multiple texture layers. Also the rendering engine

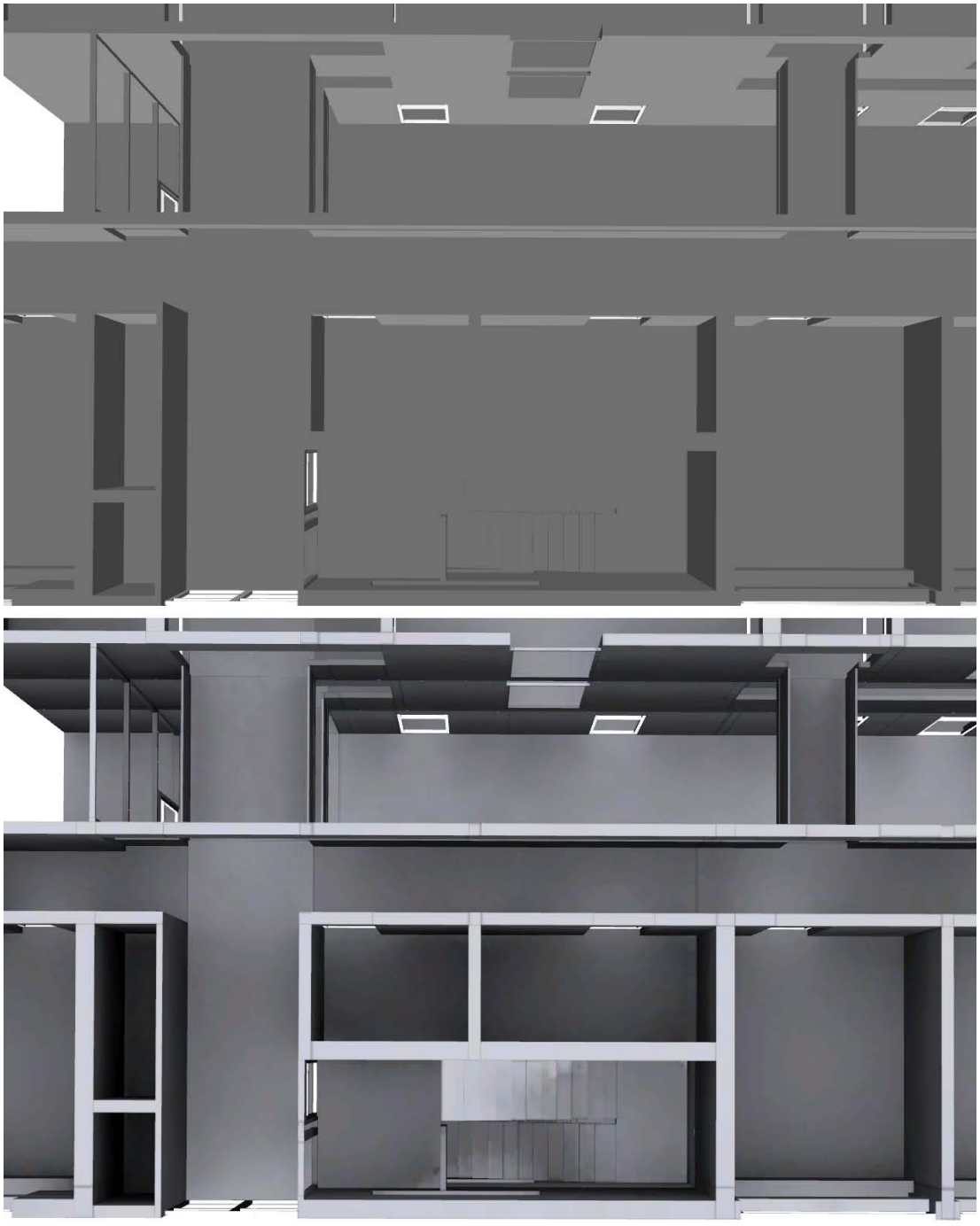


Fig. 4.22 Unlit geometry is hard to recognize (top). With pre-rendered lighting the scene is easier to identify and more appealing (bottom).

has to support the rendering of multiple textures using one as diffuse map and the other as light map. The pre-lighting process is time consuming and it generates more data (the light maps) which may again lead to problems with large amounts of data. On the other hand, it drastically increases the visual quality. A good trade-off has to be found.

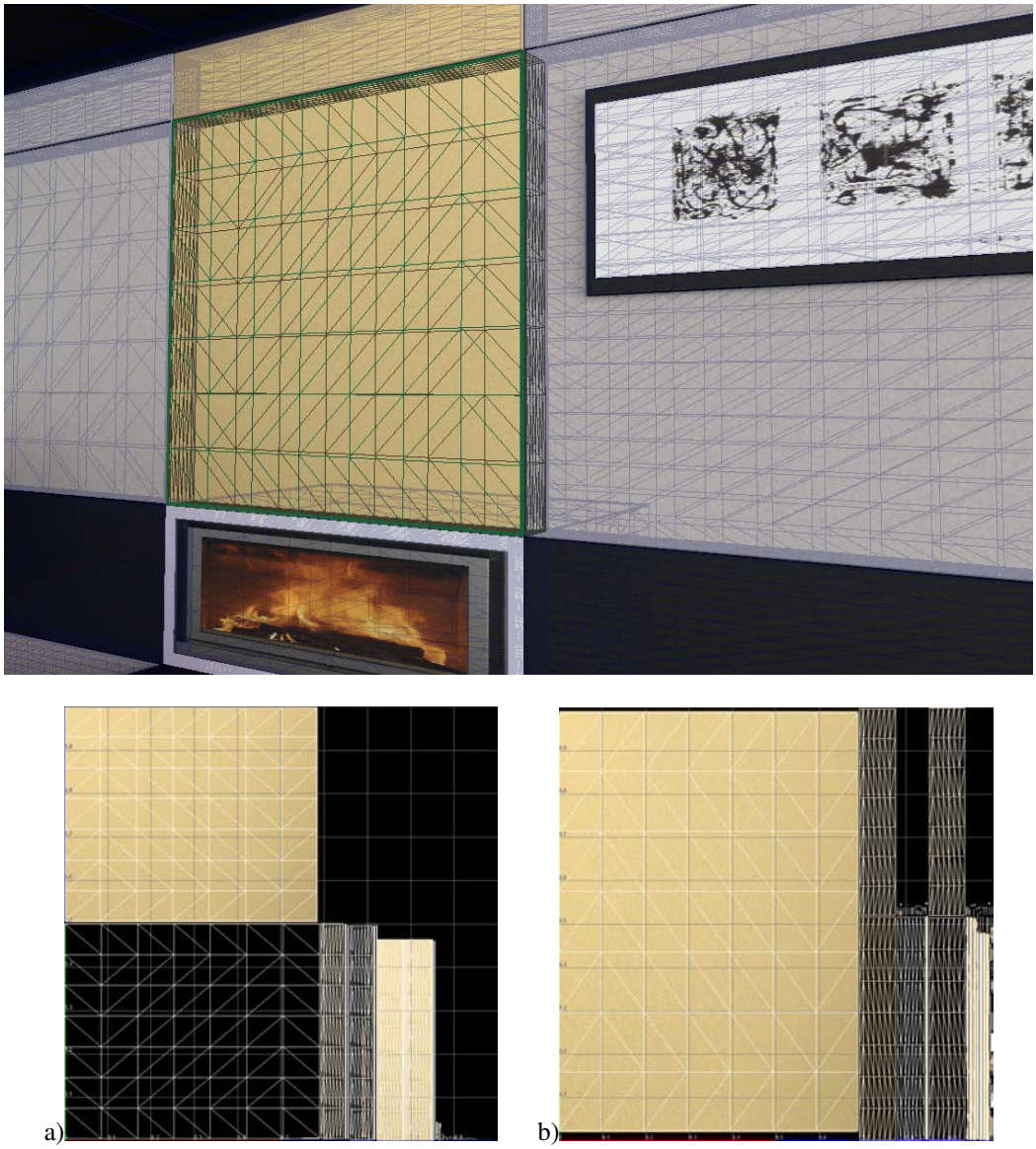


Fig. 4.23 If not optimized the texture mapping may lead to large areas of unused image space. The mapping in the lower left image (a) is created automatically. The right mapping (b) is created manually and uses more of the image space for the lighting of the visible part of the geometry.

The Swedish company Illuminate Labs (now part of Autodesk) offers a tool called BEAST[23] to automate the process of light map creation. The tool is also included in the game engine Unity[24]. With BEAST the creation of optimized texture mappings for light maps is done automatically. The tool creates true global illumination lighting and bakes it into texture maps. It also supports the creation of light probes for dynamic light effects. Some render engines offer a technique called screen space ambient occlusion which creates darkened regions in occluded areas without any pre-calculation [BSD08]. More advanced (game-) engines like the

[23] http://gameware.autodesk.com/beast
[24] http://unity3d.com/unity/

CryENGINE 3 use, for example, dubbed light propagation volumes for real time global illumination [Kap10]. Those techniques work in real time and require up to date graphics hardware.

4.3.2.4 Shaders

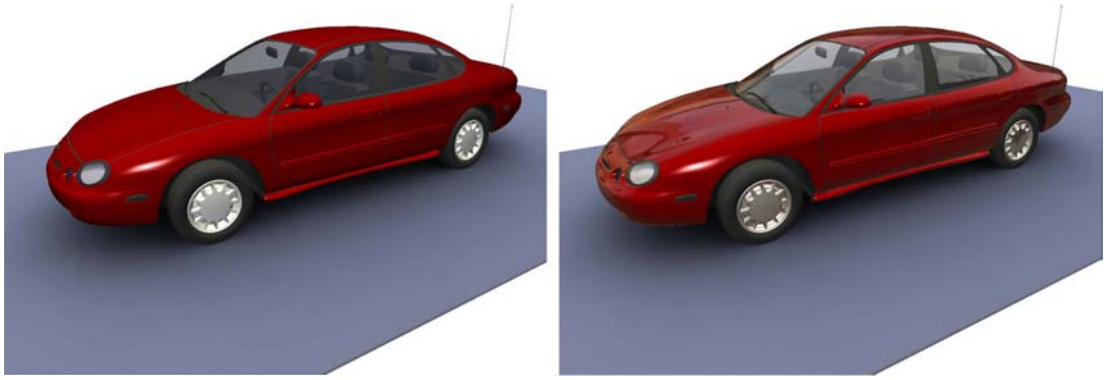


Fig. 4.24 A car model with Phong shader (left) and with a static reflection map (right).

The use of shaders helps to further increase the visual quality [RLKG*09]. Glossy materials can be created by using a Phong shader on top of a pre-calculated global illumination. Reflective objects are possible either by using pre-computed reflection maps (cube maps) or by actually calculating reflections in real time. Accurate reflections are time consuming and may not be supported by the render engine. But in many cases a static reflection map is sufficient. Figure 4.24 shows a car model with a static reflection map compared to a Phong shader. Cube maps for static reflections can be created with an offline renderer and a setup of six cameras.

It is also possible to add animations with shaders. A water surface for example can be implemented almost photo realistic using a shader program. As described earlier, shaders can also be used to pretend the presents of highly detailed geometry. By using normal maps the surface appears to be of much higher detail. For example a floor made of wooden planks consisting of many hundreds of triangles can be reduced to a plane with two triangles (see Figure 4.25).

The use of more advanced techniques like parallax occlusion mapping simulates geometric structures by using height maps. The free tool xNormal[25] by Santiago Orgaz helps to create normal maps and height maps. It loads in high detail geometry and a low detail version of the same geometry with proper texture coordinates. xNormal calculates the corresponding textures like normal map, height map and ambient occlusion.

Vertex shaders allow the modification of the perspective projection calculation. With small adjustments the depth precision can be improved and z-fighting effects can be minimized. Two methods are described by Upchurch and Desbrun [UD12]. One is a modification of the projection matrix and the other a small modification in the vertex shader.

Many rendering systems support the use of shaders but there is not one single programming language. The graphics API of the render engine defines which shader language has to be used.

[25] http://www.xnormal.net

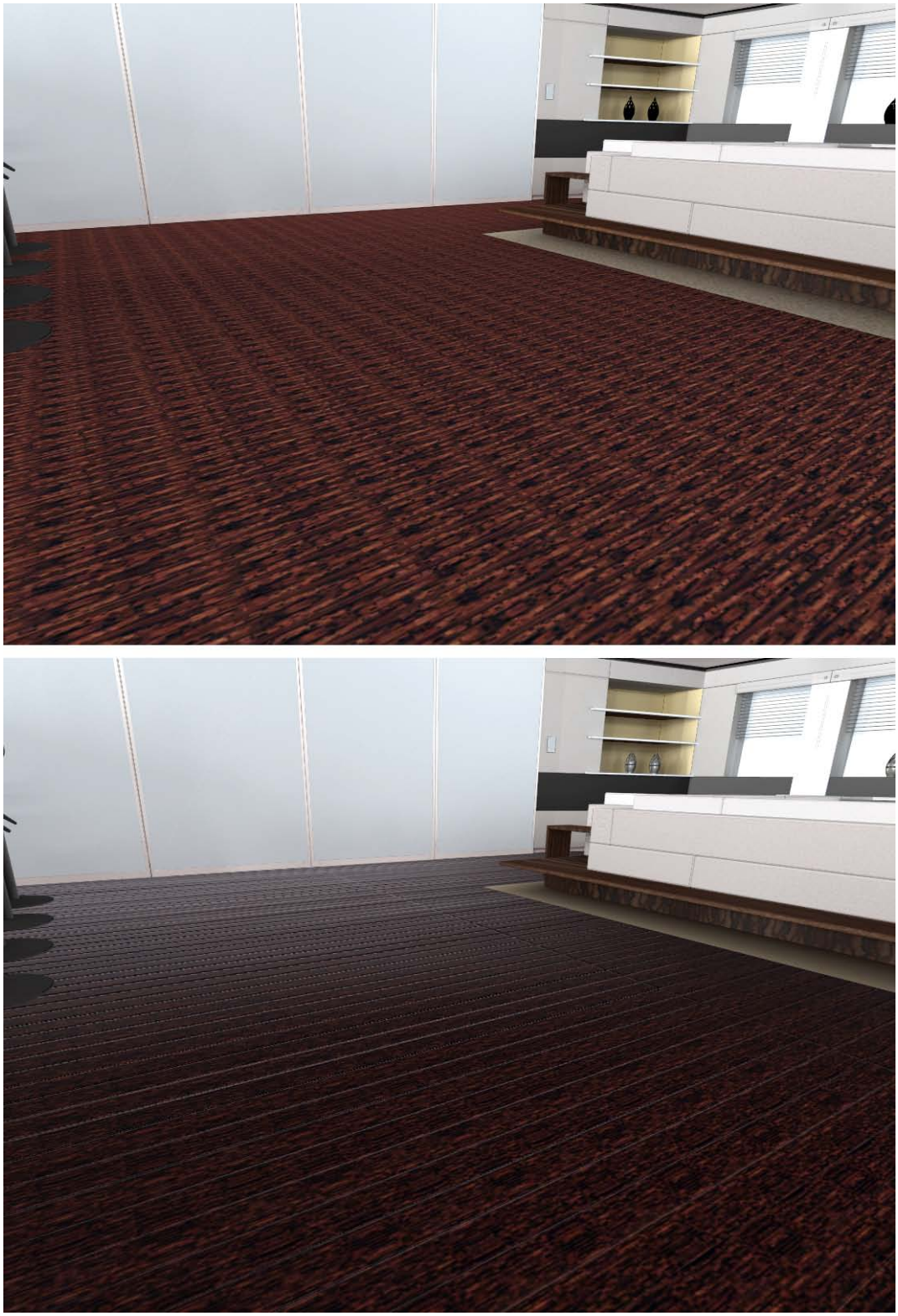


Fig. 4.25 Shaders are used to add structure to a wooden floor. The plants are added using a normal map. (Geometry by motion code: blue)

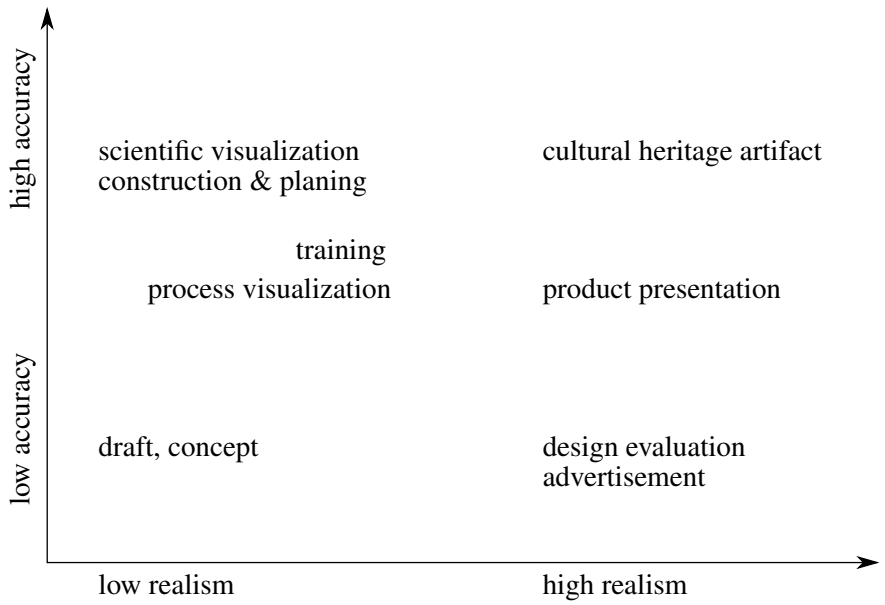


Fig. 4.26 Classification of interactive 3D visualizations with the attributes *realism* and *accuracy*.

OpenGL-based engines use the OpenGL Shader Language GLSL[26], for DirectX-based engines the High Level Shading Language HLSL[27] is used. In principle the languages are very similar. Hardware restrictions can be a limiting factor for the visualization using shaders. Laptops and tablets which are typically used for presentations are not always equipped with the latest graphics hardware and shaders developed on NVIDIA hardware many not work properly on AMD or Intel graphics hardware. Game engines like Unity and CryENGINE use their own variation of a high level shader language. The shaders are transcoded to support multiple platforms.

4.3.3 Insight

The preparation of 3D content is a necessary step for interactive visualizations. Depending on the application it is possible to automate some of the procedures but in some cases a manual revision is inevitable. For the usefulness of VR and for cost reduction it is important to minimize the manual preparation. Only in this way virtual engineering will be a useful part of the development of new products. Semantically enriched content helps to identify the importance level of parts in a large scene with many objects.

4.4 Types of Visualizations

For the classification of interactive 3D visualizations two attributes are of importance: amount of *realism* and *accuracy*. Realism is meant in the sense of appealing photorealistic rendering in contrast to functional rendering.

Photorealistic 3D graphics are desired for advertisements and appealing visualizations add quality and a professional touch to scientific results. But it is not always feasible to create interactive presentations in the highest possible visual quality. Reasons are money and time constraints as well as hardware restrictions.

In Figure 4.26 different types of visualizations are grouped according to their amount of realism and accuracy. Only drafts and concepts are allowed to be of low accuracy and low realism. Visualizations showing a process, a production workflow or a specific action for assembly or maintainance do not need realism. The focus is on the process or action and not on the appealing photorealistic graphics.

Accuracy is more relevant for training applications because the virtual objects should be authentic. Realistic rendering is not very important. For example the assembly of a machine can be trained on a 3D model with basic rendering as long as the data is accurate and the shapes are recognized correctly. The highest amount of accuracy is needed for construction and planing. CAD models are created to actually produce an object, a machine or a building. As many production lines directly use the CAD data for automatic production, the data has to be accurate in a sub-millimeter scale.

Product presentations, advertisements and evaluations of design or structure need an accuracy to a certain level but should be as appealing as possible. For those applications it is important to convince the customer or the decider that the product or the design is attractive. Functionality may be a part of the attractiveness of the product. In that case the accuracy level should not be too low.

The preservation of cultural heritage artifacts demands both high accuracy and high realism. For the analysis of virtual artifacts a realistic and authentic rendering is essential. The methods and the style of sculptors can be studied on highly accurate scans of their sculptures. A realistic rendering will emulate how the artifact was received by the observer even if the real object decays.

Scientific visualizations use an accurate data representation, the focus is on the data and not on visual effects. The data should be clearly readable and conclusions visible. Exaggeratively appealing graphics do not help to understand the data. Of course the audience defines the characteristic of the visualization. A scientist will expect a different view than a manager. The right visualization can influence the reaction on scientific results to a certain direction. The choice of colors for example will change the conception. This effect is further described in the next section for the application of distance visualization.

4.4.1 Distance Visualization

Analyzing differences between surfaces is a necessary task in many fields of research. Measuring the distance between two surfaces is a common way to compare them. In computer graphics, for example, differences of surfaces are used for analyzing mesh processing algorithms

[26] http://www.opengl.org/documentation/glsl/

[27] http://msdn.microsoft.com/en-us/library/windows/desktop/bb509561(v=vs.85).aspx

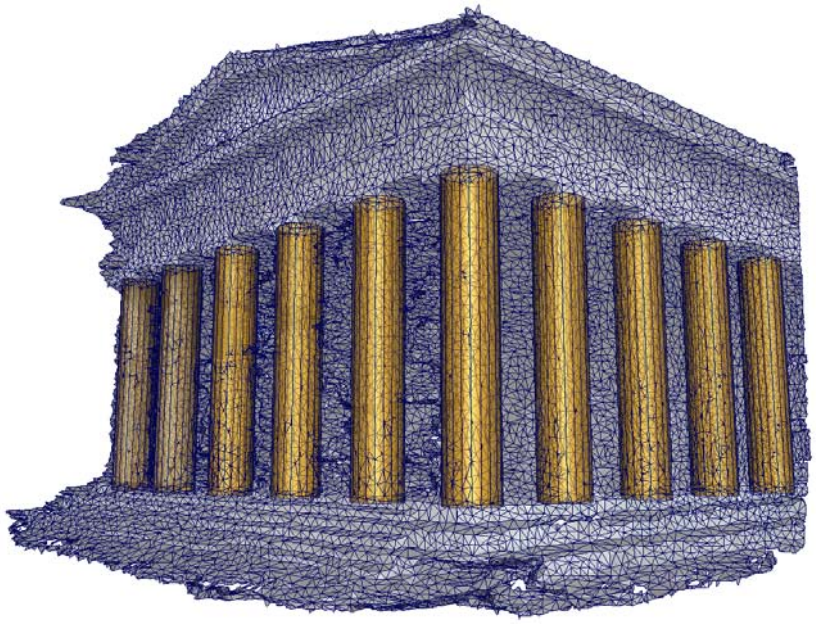


Fig. 4.27 This is an example of a bad visualization. It shows the result of a reconstruction process: a Greek temple and its fitted reconstruction. The surfaces intersect, but based on this image it is not possible to examine the reconstruction's quality. Due to a missing legend no quantifiable error can be determined.

such as mesh compression. They are also used to validate reconstruction and fitting results of laser scanned surfaces. As laser scanning has become very important for the acquisition and preservation of artifacts, scanned representations are used for documentation as well as for the analysis of ancient objects. Detailed mesh comparisons can reveal smallest changes and damages. These analysis and documentation tasks are needed not only in the context of cultural heritage but also in engineering and manufacturing. Differences of surfaces are analyzed to check the quality of productions.

A meaningful visualization of surface differences is a challenging task. The goal is a clean representation of facts without overextending the observer. This can be done using still images and also using interactive rendering. For example, Universal 3D files (U3D) embedded in a Portable Document File (PDF) allow to publish and share interactive visualizations on a widespread platform.

This section presents an application which optimizes the work flow to create such visualizations. It uses a classification scheme to group typical scenarios. Reasonable presets of settings are provided for quick output generation. The results can then be imported into common visualization tools like Maya™, 3ds Max™, or Deep Exploration™ to support the overall visualization task.

4.4.1.1 Related Work

Our application visualizes distances between geometric objects. It is related to three areas of research. The distance calculation itself is an algorithmic problem. The visualization has to deal with colors and color perception. Last but not least the application is embedded into a context.

Our algorithm calculates distances between dense samplings of geometric objects. Its main idea is based on the method for calculating errors between surfaces presented in *"Metro: Measuring error on simplified surfaces"* [CRS98] and *"MESH: Measuring Error between Surfaces using the Hausdorff distance"* [ASCE02]. In order to speed up the calculation of a Hausdorff distance, which has a quadratic runtime in a naive implementation, the samples are stored in kd-trees [GBY91]. The nearest-neighbor-search algorithm we use has an average runtime of $n \cdot log(n)$ and is described in *An introductory tutorial on kd-trees* [Moo91].

Our algorithm to calculate normal distances (i.e. the nearest neighbor search restricted to points inside a double cone) is based on a commonly used grid structure. This technique can be found amongst others in articles by Farin et al. [FHH03a] and Hege and Polthier [HP02]. Having calculated the distances our application offers predefined color palettes. The default color settings take the human perception and perceptual ordering into account. An overview on colors and color perception can be found in Maureen Stone's field guide to digital color [Sto03]. The set of predefined color maps contains the luminance-based maps with only small variations in the hue value as proposed by Levkowitz and Herman [LH92] and by Bergman et al. [BRT95] as well as the maps proposed in *"Rainbow Color Map (Still) Considered Harmful"* [BTI07]. The often used rainbow color map is available but not used as a default. The predefined color maps also contain neutral color settings. These settings do not have "signal colors" such as red. The selection of the neutral color ranges are based on *"How NOT to lie with visualization"* by Rogowitz et al. [RTB96].

Visualizations of surfaces differences are needed in many fields, e.g. comparison of mesh reduction results [KLS96]. The main context, in which we use our application, is the field of cultural heritage. Especially for issues on scanning, fitting and reconstruction the application turns out to be a valuable tool. An overview on current research topics in cultural heritage can be found in *"Recording, Modeling and Visualization of Cultural Heritage"* [BGVGP06]. As the program is not limited to this field of application, its context will not be discussed in detail here.

4.4.1.2 Classification and Algorithms

The main idea of the application is to classify the distance visualization problem into categories. Each category has a set of default settings which lead to feasible results.

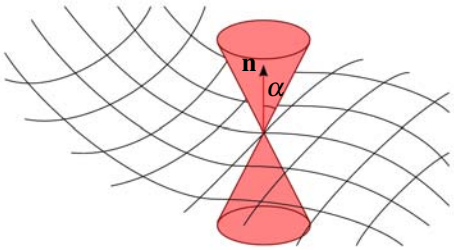


Fig. 4.28 In some cases it is sensible to restrict the nearest-neighbor-search to samples inside a double cone along normal direction. Undesired sample relations at parts, which do not have a corresponding counter part, can be avoided.

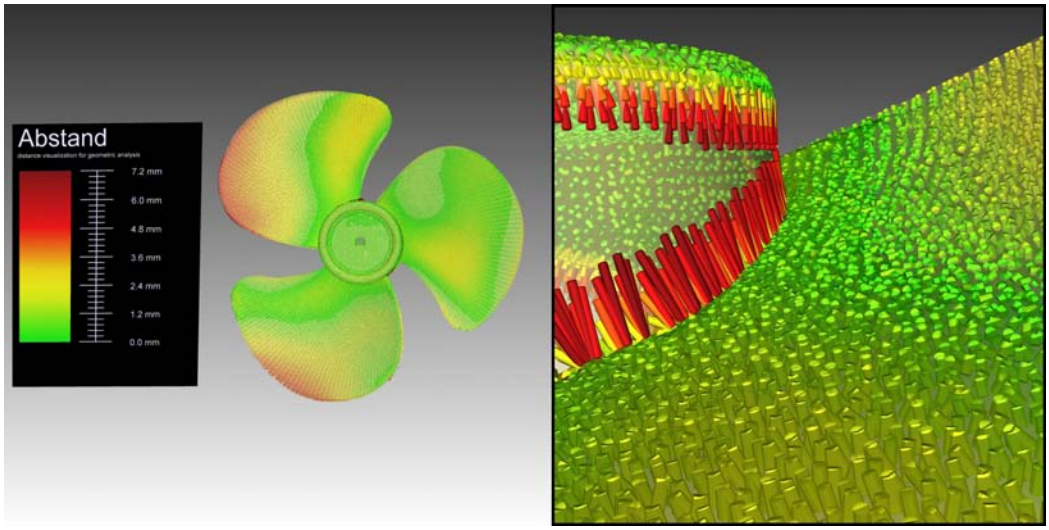


Fig. 4.29 This example shows a laser scan of a propeller and its corresponding CAD model. The reference model has been reengineered based on measurements of the real object. Using this input data, the application ABSTAND calculates an appealing visualization.
The scanned surface is colorized according to the distances to the reference model. To have a clear look at the colored distance visuals, the reference model is almost transparent. For this example, only the one-sided distances from the actual model to the reference model are visualized using small, solid cylinders. The user can choose the number of distributed cylinders. The polygon count of the result can be limited to a desired, maximum number. According to these settings the quality/level-of-detail of the cylinders is determined.

4.4.1.3 Variance Analysis

All distance visualization problems belong to one of two distinct groups. The *asymmetric* case analyzes two geometric objects assuming that the first object is the reference / nominal object. The second object is the actual object to be validated. Such a configuration can be found e.g. in the context of quality management using a CAD model as reference to check the resulting product (see cathedral example in Section 4.4.1.12).

The *symmetric* case is characterized by the absence of a reference model. Both objects are on a par. In contrast to the asymmetric case the results of the symmetric one do not change, if the order of the imported objects is swapped. A typical, symmetric situation is the comparison of two range maps of a laser scanning process. If overlapping regions of aligned scans are analyzed, none of them can be considered to be the ground truth (see chess pieces scan examples in Section 4.4.1.12). These two main groups require different settings.

4.4.1.4 Symmetric Distance Visualization

The distance analysis starts by generating samples of both input objects and calculates their normals. Then the one-sided distances are computed and assigned to the samples. The nearest-neighbor-search can be restricted in two ways.

- For some cases it is useful to restrict the search area to a double cone along the normal as illustrated in Figure 4.28.

- Lower and/or upper bounds are another way to filter the results – for example ignoring all distances smaller than an epsilon.

The remaining distances are sorted and then grouped in a histogram according to their length.

For the two analyzed meshes, a unobtrusive coloring is suggested. The transparency value may vary according to the signed distance. This technique enables to display inner parts which would otherwise be covered by opaque surfaces.

Solid cylinders (or prisms with a lower polygon count) are generated to visualize the distances (see Figure 4.29). These distance visuals are grouped using the calculated histograms.

4.4.1.5 Asymmetric Distance Visualization

Calculations for the asymmetric case start similar to the symmetric one. But the results can also be limited to one-sided distances. As one surface is considered as ground truth, the visualization emphasizes the actual object. The reference object plays a minor role in the visualization. Its main purpose is to provide orientation in 3D – especially if the actual objects (e.g. scanned remains of a vase) are much smaller than the reference object. The actual object may also be colorized according to the assigned distances.

4.4.1.6 Colors

The geometric objects / meshes as well as the distance visuals can easily be colorized using preconfigured color scales. These schemes include color maps with good order properties in terms of human perception. Most of distance visualization schemes use luminance-based scales, for example the black-body radiation spectrum. For surfaces isoluminant color maps with opponent colors are suggested (see Figure 4.30). These surface colorizations do not compromise the depth perception. Neutral color tables are also available, if extra highlighting of differences is not desired. If the geometry is shown in a single color (with possibly varying transparency), the application proposes a color which does not belong to the color scale. Furthermore it automatically generates a legend in an appropriate range.

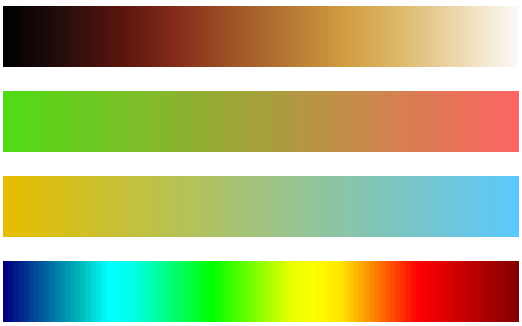


Fig. 4.30 The color settings include color ranges, which take human perception into account; e.g. the Black-Body-Radiation scheme in the first row. According to Borland et al. geometry should be drawn in an isoluminant scheme (second and third rows). The set of predefined color settings also contains some neutral color schemes, proposed by Rogowitz et al., as illustrated in the third row. It does not use "signal colors" such as red. Deceptive and misleading color ranges such as the rainbow color scheme (last row) are also included, as they are often used in wide-spread visualization systems.

4.4.1.7 Implementation

All of the described algorithms have been implemented into the Java-based application ABSTAND. The stand-alone program can be downloaded or started via a web browser:

http://www.cgv.tugraz.at/Abstand

The user only needs to provide two meshes to be analyzed.

4.4.1.8 File Format

For the input meshes and for the results of our application, a suitable file format is needed. There are too many "standard" formats for 3D data. The practical approach to this problem has been to look for a format that is widely used by common modeling applications and viewers. In this way, the import of geometry and further processing of the results is harmonized.

We use a subset of the 3D Object format (OBJ) introduced by Alias Wavefront. Coloring the results is done by a material file (MTL). In lack of per-vertex-colors, the material file stores a separate material for each color and transparency value. This subset ensures full compatibility to other applications.

A generated legend can be included into the scene. Special care has been taken to ensure a correct rendering of the legend. Therefore, the captions are put on a texture template whereas the color scale itself is not part of the texture. It is built out of quads using the same material file and settings as used by the distance visuals. This approach ensures that the appearance of the colors is coherent. Different texture environments, which specify how texture values are interpreted when a fragment is textured, may otherwise lead to color discrepancies.

4.4.1.9 Usability

The visualization of distances can be done in various ways with many parameters. To have a useful and supportive tool, it is very important to have a good set of predefined parameters working out-of-the-box. Only by choosing a scenario based on the classification in Section 4.4.1.3 the application is able to automatically generate an appealing visualization. But tweaking of all the parameters is also possible in the advanced settings.

4.4.1.10 Manifold Approximation using Normal Vectors

The normal vectors of the triangles are used to define half-spaces. With these half-spaces the inside and outside of the difference space in-between is defined. To work properly, the normals have to point outwards of tessellated objects. There is no way to determine correct inside and outside spaces fully automatically in all cases, especially for surfaces with holes. The distance calculation provides signed distances to each sample. Then it is possible to set the transparency according to the depth. The inside parts of a surface can be opaque and the outside parts transparent.

The distances can be visualized using simple cylindrical solids (or prisms with a lower polygon count). Using solids gives a more volumetric look than with lines. Furthermore, this representation is supported by almost all applications capable of importing OBJ files, while importing lines is only available in a small number of viewers.

4.4.1.11 Transparency

In most cases the visualization contains two surface layers. To show both objects it is necessary to use advanced techniques such as cut-away illustrations, which can hardly be made automatically and need special viewers, or adequately chosen transparency. In order to use conventional tools/viewers we chose an automatic approach based on transparencies.

While in general it is no problem to render still images with transparency, interactive display may not always render transparencies correctly. Correct rendering of transparent objects needs a back-to-front sorting of all surfaces. Special care has to be taken for interpenetrating objects.

To have an appealing visualization for the interactive rendering, we offer a transparency simulation by a varying wire frame representation. Each transparent triangle, for example, is replaced by three quads as shown in Figure 4.31. The area of the quads is inversely proportional to the triangle's transparency. For high transparency values, the quads almost form lines along the borders of the triangle. For low values, the triangle appears rather opaque leaving only a small hole in the middle of the face. This technique offers a comparable illustration in viewers, which do not render transparency correctly, at the expense of the polygon count.

4.4.1.12 Examples

The first example shows a result of the asymmetric preset of ABSTAND. It shows a data set of the Pisa Cathedral, which has been generated by the Visual Computing Laboratory at the Institute of Information Science and Technologies (ISTI) of the Italian National Research Council (CNR). The scan of the Duomo Pisa is compared to the results of an algorithm which identifies arcades automatically. The visualization – shown in Figure 4.32 – helps to verify the quality of the algorithm's output. Mismatching areas can be identified easily.

The second set of test objects demonstrates the symmetric case. It consists of two range maps from a laser scanner. The range maps have only a small area of overlap. Having set the upper bound of visualized distances slightly above the scanner's accuracy allows to concentrate on the alignment fit. Both objects have been acquired using a NextEngine™ laser scanner. Figure 4.33 shows a high-quality rendering of the resulting visualization. Furthermore a 3D representation is embedded in U3D format. Transparent faces are transformed according to the method described in Section 4.4.1.11 to ensure an appealing result independent of the capabilities of the U3D rendering plug-in. The Acrobat™ U3D plug-in allows to inspect the surfaces as well as the histogram structure of the distances. The predefined groups can be selected, marked and hidden using the plug-in's tree view.

Fig. 4.31 The overall transparency of a rendered object can be simulated by a varying wire frame representation. The area of the quads is inversely proportional to the triangle's transparency. This technique offers a comparable illustration in viewers, which do not render transparency correctly, at the expense of the polygon count.

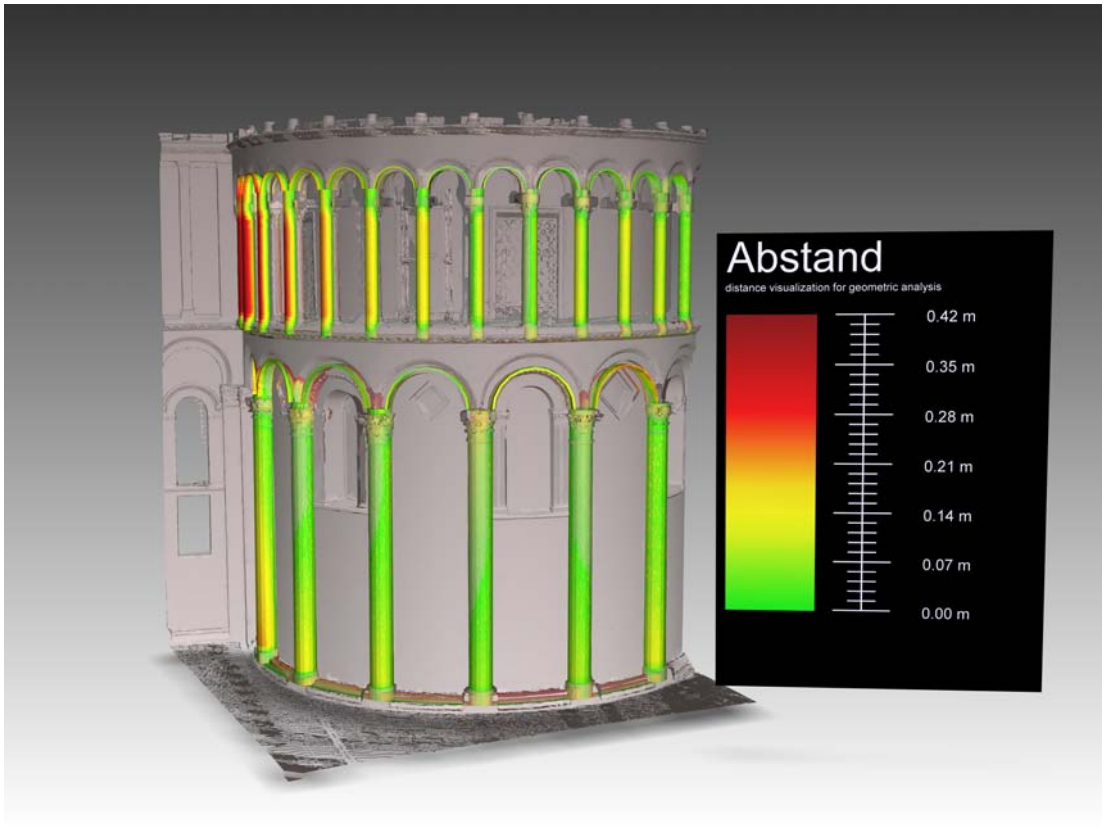


Fig. 4.32 In an asymmetric configuration an actual/nominal comparison is visualized. The reference object is a laser scan of the Pisa Cathedral. Two fitted arcades have been determined by an algorithm. The difference between the scan and the automatically calculated surfaces describe the quality of the fit. Using ABSTAND these differences can be analyzed and visualized easily.

4.4.1.13 Insight

We presented an out-of-the-box application to visualize distances between surfaces in an easy to use way. The proposed classification scheme is suitable to assign all surface comparison tasks to two main sets of default settings. Matching color tables are suggested automatically on a scientific basis.

By using a well established file format, the output files are harmonized with common 3D rendering tools and viewers. On the one hand the resulting files can be used to produce still images in the classic ray tracing fashion. For interactive illustrations on the other hand the application offers a technique to optimize the export settings to get appealing results. These optimized settings avoid irritating misinterpretations and visualization errors in different viewers. The overall work flow to produce high-quality visualizations of distances between surfaces has been reduced significantly. Furthermore the application takes the latest results in human perception and visualization techniques into account.

Although the presented application offers many possibilities, sensible defaults allow an easy handling. The application is organized in a check list-like manner. Processing each point (choose distance function, select color scheme, etc.) step-by-step reduces the probability of many commonly made errors in diagrams (e.g. no legend included).

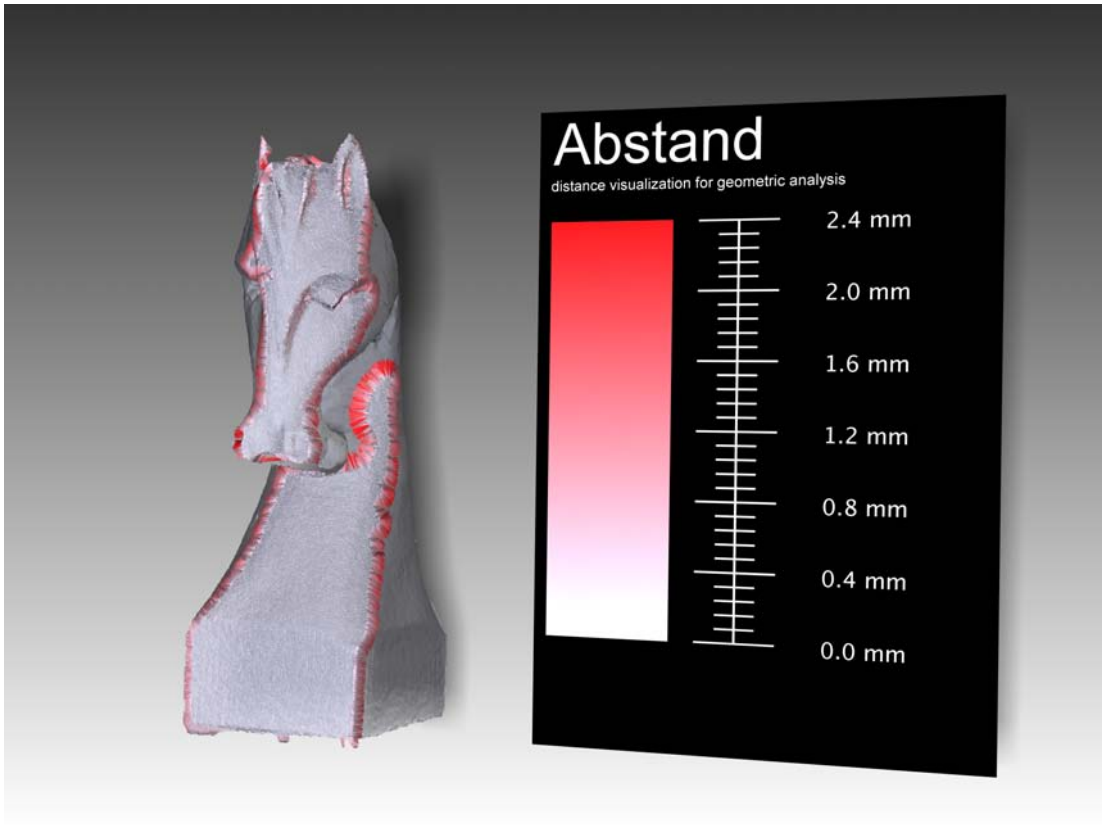


Fig. 4.33 During the scanning process several range maps of a scanned object have to be aligned to each other. The presented application ABSTAND is able to create the visualizations that highlight the alignment fit. Having selected the class of settings only the range of distances to visualize has been adjusted to the accuracy of the scanner. The export routine has been harmonized with diverse tools to allow easy conversion – for example to U3D for PDF embedding.

The application uses state-of-the-art algorithms to calculate the distances very fast, but in contrast to many other distance visualization programs it also concentrates on the resulting visualization: a clean representation of facts without overextending the observer. Last but not least, the application is based on web technologies, works out-of-the-box and does not need any configuration or installation:

http://www.cgv.tugraz.at/Abstand

4.5 User Interaction

Compared to pre-rendered videos the big advantage of interactive visualizations is the possibility to involve the user actively with some kind of control. The user may have free control over the camera, data selection, a virtual character or other parts of a virtual scene in real time. The user interaction is handled via input devices and output from the application. The focus of this work is output in form of graphics but other outputs can be combined like sound or vibrating elements. For the user input, many different devices are available. Depending on the environment and the application not all of them are useful. This is only a small list of devices:

Keyboard The Keyboard is the classical input device and it is part of many computers since the very beginning. It is perfectly suitable for typing long texts. Keyboards are easy to use as they are familiar for a large group of users. At the same time a keyboard is too common to be a fancy input device. Its usage for interactive visualizations requires the program to reveal a control description. Arrow keys may be somehow self-explanatory but as there are many different ways to interpret "left" and "right" an explanation is needed anyways. Keyboards have many buttons which can be used for different actions. But for the majority of applications, a keyboard means just too much input possibilities. Anyhow, for the input of text a keyboard is essential. With the broad acceptance of touch screens the keyboard is often simulated in software.

Mouse The development of graphical user interfaces demanded a pointing device to select elements on the screen. The computer mouse, introduced in the mid 80s, is one of the standard input devices which is available on almost any desktop environment. The classic version needs a surface on which it is moved around. Over the years the mechanical motion detector has been replaced by an optical sensor. The mouse has at least one button, many have three and a mouse wheel. For laptops the mouse was substituted by a touch pad or touch screen.
As the majority of users is familiar with the usage of a mouse, it is a good option for interactive visualizations. It is especially suitable for fine and precise movements. Users have some experience using the mouse to point to positions on the screen. The mouse buttons are easy to describe and the number of actions is clearly laid out. Still, for 3D navigation tasks there are different approaches on how to use the mouse. For example the distinction between rotation, panning and zooming can be handled with different buttons and movements. As the options are limited experienced users will easily learn the mouse controls while trying out. For the desktop setup mouse inputs are often combined with keyboard commands.

Joystick/Gamepad Many machines in real life are controlled by joysticks. It was only logical to use them also for computer applications, especially games. A typical joystick consists of at least one analogue stick-shaped element for capturing two axis of continuous value changes. A set of buttons is used for additional inputs. Starting with the Nintendo Entertainment System in 1985 joysticks have been replaced by gamepads as standard input device for game consoles. But many of them still have analogue elements like sticks. Some gamepads also have analogue buttons. Compared to the keyboard the game controller (joystick or gamepad) is easier to use for a small number of actions. It has less input possibilities and buttons are easy to describe. Gamepads are designed to be held in the hands of the user, typically cordless. Game controllers are inexpensive and can be programmed without much effort. But the connection to games make gamepads unpopular for serious applications.

Multitouch Surface The first touch screens, described by E.A. Johnson and A.M. Hlady in the late 60s, were capacitive and could recognize single touches. Then optical multi touch devices were developed, for example by Wellner in 1991 [Wel91]. Capacitive touch screens work with a conductive coating covered with a thin protection layer. By touching the screen

the user conducts the current at the position of the touch. This causes a voltage drop which is used to detect touch events. Capacitive touch screens only react on conductive objects. Resistive touch screens use two layers of metallically coated plastic. The layers are separated by air gaps. If the two layers are pressed together, the touch event is triggered. Resistive touch screens react not only on fingers but also on other objects like pens and gloves.
One of the most prominent developers of optical multi touch devices is Jefferson Han. In 2005 he presented his low cost scalable touch technique for rear-projected surfaces [Han05]. His camera based method uses frustrated total internal reflection (FTIR). Infrared light is send into a Plexiglas surface. It is reflected internally and stays inside the Plexiglas until a user touches the surface. At the touch points the infrared (IR) light is scattered to the outside where it is observed by a camera. The camera filters only for infrared light and the touch points can be calculated. Infrared filters on the outer screen surface keep out other infrared light sources. The Plexiglas can be used as a back projection screen at the same time by adding a diffuse layer on the outside.
The Microsoft Surface table also uses IR light but in a different way than FTIR. IR emitters illuminate the surface from within the table and cameras capture the reflections of objects or hands on the surface. In this way it is possible to use special marker stickers on objects which are recognized in addition to hands and fingers. The second version of the Surface uses Microsoft's PixelSense technology. Basically it is a flat panel screen with small infrared emitters and cameras build directly into the panel besides the pixels. The usage of objects (tangibles) which are recognized by the touch surface allows some nice applications. But both Surface models are quite sensitive to external light sources as IR filters cannot be used.
Another technique is to use line cameras at the border of the screen. Citron GmbH[28] offers frames for different sizes of screens. To avoid occlusion they use multiple cameras per side (22 are used for a 46" frame). The line cameras can only see a very small area directly in front of the screen. Comparable to the Surface the touch events do not require pressure onto the screen. On the other hand also sleeves of jackets are interpreted as touches. And the recognition of tangibles is hardly possible as the cameras only capture a very thin area in front of the screen. But in contrast to the Microsoft Surface the line cameras are not distracted by external light sources.
Touch input is useful for environments in which extra input devices are impractical. It is used for almost all state of the art smart phones. The device is reduced to a screen and the user interacts only with the screen. Multi touch screens are very popular for tablets like the iPad. There is only one big screen without any buttons. The same concept works for screens that are integrated into tables (see Table 4.2.4).
Touch events are different from mouse events in the following: The user does not move a cursor but touches directly on elements of the screen. This prevents the application from realizing the user hovering over an element before pressing it. It is also not possible to use the metaphor of cursor graphics changes to indicate special areas, for example a link on a web page. The finger of the user will cover the position of the touch. Graphical feedback has to be painted larger than a typical finger in order to be visible. And the size of a touched area is quite big compared to a single pixel being clicked with a mouse cursor. Touch interfaces have to be designed with respect to these restrictions. On the other hand additional functionality can be implemented with gestures. For example the pinch, spread and rotate gestures performed with two fingers are widely used for zooming out, zooming in and rotating images not only on Apple devices.

[28] http://www.citron.de/

Motion Sensor With the introduction of the Nintendo Wii in 2006 the motion sensor became a popular input device. An accelerometer is aware of the relative movements of the Wii controller (Wii Remote). The orientation is measured in two axis (rotations around the gravitation axis cannot be measured in the original version of the Wii Remote). In addition also buttons and an optical sensor are used. Many new concepts were implemented in the games for the Wii. But the controller can also be used for serious applications. Many head mounted displays use a motion sensor to determine the orientation of the users head and Citron GmbH[29] added an accelerometer to their touch screen frames for automatic detection of the screens orientation. In the EPOCH project an accelerometer was introduced to control virtual artifacts of museums in 3D [HSLF07]. Gyration[30] offers a mouse which is equipped with a motion sensor and accelerators are integrated in recent mobile phones.

Optical Sensor Computer vision techniques can be used to measure movements of simple objects, faces up to persons using video images. In 2005 Zeev Zalevsky invented an efficient method to capture depth information together with a color image. He uses a random infrared pattern projected with a shift onto the view area of the camera. Microsoft adapted this technique and introduced the Kinect in Nov. 2010 for Xbox 360 and in 2012 for Windows. The Kinect is a low cost 3D scanner that has a build in recognition of persons and also an advanced microphone. Although the Kinect was targeting on the game industry many serious applications have been developed. Its big advantage is the use of body motion instead of hand-held devices. The use of the Kinect in an immersive environment is described in Section 4.5.3.

4.5.1 Traveling in Immersive Environments

The next sections present different ways of traveling in immersive environments. Various navigation methods in immersive virtual environments have been developed. Obvious options like pointing based methods were described over two decades ago. Some of them are summarized by Ware and Osborne [WO90]. More recent overviews categorize existing travel methods like the ones from Bowman [BKH97, BKLP04] and Mine [Min95].

Several other application specific travel methods were implemented. For example, Ciger et al. use a magic wand for navigation using posture recognition or pointing in combination with speech recognition [CGVT03]. Without compromising interaction Deligiannidis researched how to navigate using a tracked cyber glove [Del04].

Some applications require hands-free navigation. LaViola et al. describe multiple hands-free techniques for multi scale ground based walk navigation and also address the problem of a missing back wall in a CAVE™. By amplifying the mapping of the user's orientation 360 degree views become possible [LFKZ01]. They also introduce a pair of special slippers for the navigation task. Their setup uses a magnetic tracking system and the user is required to wear a belt for the tracking of the waist in addition to head tracking. Bourdot et al. use a stateless approach where different zones for the user's position are used for special behavior of the navigation [BDA99]. Fuhrmann et al. use head directed navigation inspired by kids playing airplane [FSG98]. A few of such methods use additional input strategies to change the mode or state of the navigation, e.g. by tapping with the feet [WPA97] or by voice commands. Adamo-Villani et al. developed and evaluated a travel interface using stepping on a dance mat [AVJ07]. Beckhaus et al. also used a dance mat and developed a chair interface for

[29] http://www.citron.de/
[30] http://www.gyration.com

traveling [BBH05]. This hands-free method of navigation is not very natural and the user is distracted too much by choosing the correct floor button for traveling actions. The experience of physical movement with the possibility to precisely estimate the traveled distance is not covered by these techniques.

Another type of setup is a mechanical locomotion interface such as 1D or 2D treadmills or hollow rotating spheres the user walks in. Among others, Iwata et al. developed several innovative locomotion interfaces like treadmills or moving tiles [IYFN05]. Fels et al. build a virtual swimming interface [FKC*05]. *CyberSphere* [FRE03] and *Cyberwalk* [STU07] are special platforms that allows the user to walk within a virtual environment. While the former system uses the rotating sphere, the latter one employs balls which are actuated by a belt on a turntable. However, these locomotion systems require a huge mechanical effort and in practice they are often more difficult to use than one would expect. Eventually they still cannot reduce simulator sickness problems because the real physical and virtual visually perceived accelerations do not match.

For pose measurement and reconstruction of a user in a VR environment, optical, mechanical, inertial or magnetic tracking systems can be used. The work most closely related to our implementation in Section 4.5.2.2 is about inverse kinematics with an optical tracking system by Grochow et al. [GMHP04]. Given a subset of markers, a motion capture data base is used to synthesize the most likely pose.

Recent works describe experiments with the Microsoft Kinect as Natural User Interface for CAVEs™. For example Jung et al. use the Kinect in combination with a Nintendo Wii controller for traveling in virtual worlds [JKS11]. Other techniques use hand and arm gestures for navigation and traveling tasks.

4.5.2 Intuitive Navigation in Virtual Environments

Immersive environments allow to enter virtual worlds which offer undreamed-of possibilities and new experiences. But being totally immersed requires to travel through the virtual world in an intuitive way. This means that interactions which would not have to be explained in the real world, should not be in need of an explanation in virtual worlds. When disregarding this basic principle, the user interfaces create a barrier that compromises the immersion's impact.

Also, the user's inhibition threshold towards immersive interaction can be lowered remarkably. This is especially important for virtual reality (VR) installations that are often used for demonstration purposes or in exhibitions [FLL*93]. Visitors are almost always first time users who should quickly be able to try out the system. A good example in a non-immersive environment is *Nintendo's Wii Sports Tennis*. As no user tracking is available, only arm and hand movements are interpreted but not the user's movements and position. The computer sets the player's position automatically. In this way, the input device as well as the user interaction can be kept simple and user-friendly. In our DAVE we realized these basic principles in immersive environment applications that require special attention due to their unconventional needs.

4.5.2.1 Hardware Setup

In our DAVE and at our stereo wall we use optical tracking systems primarily to adapt the 3D viewpoint according to the user's eye positions (see Section 4.2.3). With multiple cameras passively reflecting markers are detected and triangulated. Since all markers look the same an

identification is only possible with heuristic estimations or a fixed constellation of markers that we call *target*. At least three markers are necessary for 6 degrees of freedom.

We greatly improved the user experience compared to our previous magnetic tracking due to a higher frame rate, a much higher precision and no need for wires. These issues and the constraints of available input devices must be taken into account when designing navigation methods. With additional acceleration sensors for example, completely different ways of interaction would be possible.

4.5.2.2 Applications

Glider Simulation

To achieve an intuitive way of controlling a virtual glider we track the user's outstretched arms. The glider's roll/yaw behavior is controlled by the user's hand-to-hand line. The angle α of this line to the horizontal plane is taken as the roll-angle of the glider, also impacting it's yaw-angle. This is illustrated in Figure 4.34. The user's position in the DAVE affects the pitch angle of the glider. Like walking on a seesaw, going forward causes the glider to fly downwards, moving backwards results in flying upwards. By accumulating the pitch angle also loops are possible. For a better way of understanding the relations between arm motion and the glider's reactions, users see the plane from a point of view behind the plane rather than a cockpit view. An informal user study showed that users are quickly familiar with this navigation method. We also tested more sophisticated interaction details for more experienced users. Wing-span and consequently lift force can be varied by hand-to-hand distance. Furthermore, flutter movements can trigger altitude gain.

We generalized the idea of using head and hand position to roughly estimate a user's posture and subsequently use it to control a VR application. The estimated posture reconstruction

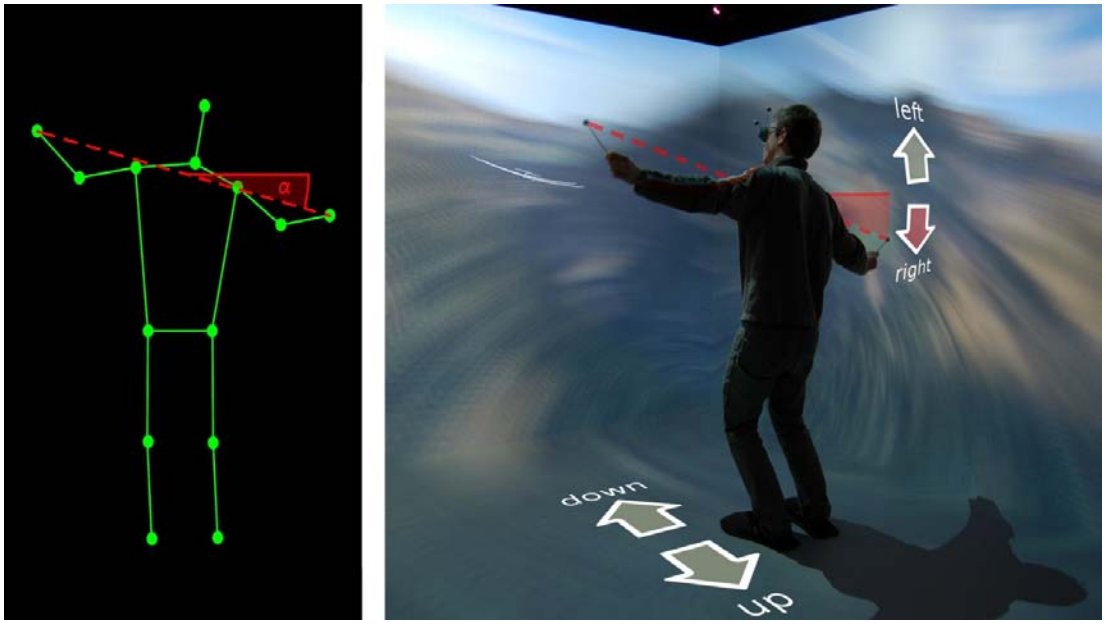


Fig. 4.34 A glider simulation. A skeleton is roughly estimated from the tracking data and used to control the glider. The arrows and lines are overlaid on top of the photo for illustration purposes.

yields positions of the user's principal joints including neck, shoulders, elbows, hips, knees and feet - listed in decreasing grade of certainty. Of course no individual leg movement is measurable without additional sensors that could be attached to the user's feet. The following three techniques for posture reconstruction were implemented and evaluated.

Initially we implemented an approach with inverse kinematics. The user's joint distances can be guessed from his body height. Involving a simple geometric model of each joint's position in relation to the other joints, a likeliness prediction of the user's skeleton can be made. With regard to the glider application our implementation was mainly designed for standing poses where it performs well. However, interpreting very different poses requires a more complex implementation which was not examined more closely.

To overcome this limitation, we tried another two methods based on self recorded motion capture data. For further processing, a normalization step is applied to remove the dependency of the user's orientation, position and size. After computing a skeleton, the inverse transformation is applied in a denormalization step.

The first approach uses a look-up table (LUT). We simply search the most similar hands and head constellation in that table and use their prerecorded joint positions. To decrease the best-fit search effort, instead of one big LUT we use several small sub-LUTs, each holding poses for one interval.

Finally we trained artificial neural networks on our motion capture data, where each joint is represented by three nets, one for each Cartesian joint position. For example, feeding the three right knee nets with the user's hands and head positions results in a hypothesis of the knee's position. By taking care of not overfitting the networks during the training phase, our implementation of this method shows the best results.

Since the problem is inherently under-constrained there is no general best setup for all possible applications. The last two methods can easily be adapted to a different application by using a different number and position of markers during interaction and another set of prerecorded typical poses.

Walk Navigation in Quake III Arena

As the full source code of ego-shooter game *Quake III Arena* was released by *Id Software* we decided to adapt it to our DAVE. Implementing a hands-free navigation the user is able to aim in another direction than his view, which is not possible in the original game. This is even more beneficial than a flystick, as one does not have to take control of two devices. We wanted the user to be able to aim with a gun independently of his viewing and walking direction unlike in the original game. Our solution is an intuitive hands-free navigation. Walking navigation is controlled by the user's head position. In center of the DAVE floor we draw a circle that defines a neutral area (see Figure 4.35). As long as the user is inside this region no navigation is triggered. Just as he steps outside the circle, the user's coordinate system in the virtual world is continuously translated so that he walks along in the virtual world. To enable moving just along the principal axes we use four areas positioned near the center of each axis. In the red hatched regions diagonal movement is possible. The distance from the center influences the walking speed exponentially. The concept with strict regions is easy to understand. However, in practice we use continuous functions for smooth accelerations.

As there is no back wall in our DAVE, the user's head orientation has to be taken special care of. We define a maximum neutral rotation angle to the left and right, relative to the direction normal to the front wall. If a user exceeds one of those angles, the whole scene smoothly rotates towards the front wall. We also implemented the actions "jump" and "duck" to be activated by

the body movement of the user. By estimating a user's height it is easy to detect those actions with the vertical position of the glasses.

More Applications

We ported the open source game *PPRacer* (formerly *Tux Racer*) to our DAVE. A penguin slides down a snow slope collecting fish. The user's head position controls the speed. The penguin accelerates when the user lies down, copying the penguin's pose. Holding the head to the side, the penguin turns (see Figure 4.36). While the input value for turning is continuous, unfortunately the game only uses a binary decision. This uncommon way of controlling a game is fun but some visitors are not willing to lie down, being afraid to make a fool of themselves.

Applications in that the user looks at a large virtual world from inside are most suited for the DAVE. In contrast, looking at a small object from all sides is not that well suited for our DAVE. For that reason we usually need navigations related to walk or fly-throughs. However, for another type of application we implemented a different navigation method. Playing a virtual 3D billiard game the user can orbit around the table by stepping to the side. The center of the table is always displayed on the front wall. This idea can be generalized to move along a predefined path with a corresponding view direction for each position.

Fig. 4.35 The user's head position controls the walk navigation speed in the game Quake III Arena. In this image the user moves sideways by standing left to the center. His hands are free for other interaction devices such as the gun. The bottom marks are overlaid for illustration purposes and are not visualized within the game.

Fig. 4.36 Hands free navigation. A user controlling a penguin by his head position. The arrows and lines are overlaid on top of the photo for illustration purposes.

4.5.3 Hands-Free Navigation in Immersive Environments

With all of the different methods for traveling in virtual worlds, none of the approaches listed earlier realizes a true hands-free navigation that is intuitive, allows interaction with a smartphone, the carrying of suitcases, etc. while at the same time allowing to estimate the traveled distance realistically. Therefore a new approach has been developed which is presented in this section.

4.5.3.1 The Hardware Setup

The projection screens of our DAVE are 3.3 meters wide and 2.5 meters high (see Section 4.2.3). Effectively an area of 3×3 meters can be traveled by physically walking. The user can simply walk around an object to inspect it from all sides. A big advantage compared to most head mounted displays is the very wide field of view. Also there are only relatively small changes to the projected images when the user rotates his head. However, natural walking is very limited due to the small room size and no haptic feedback is available. In order to explore a larger 3D world, navigation or so called travel techniques as mentioned in the last section are necessary. These techniques do not cover the experiences of a physical movement with the possibility to precisely estimate the traveled distance.

Installation of the Kinect Sensor

The installation of the Kinect Sensor was difficult because of the spatial restrictions in the DAVE. Three positions are plausible:

- At the ceiling, similar to the floor projection using a mirror
- At the back opening of the DAVE
- Above the front wall

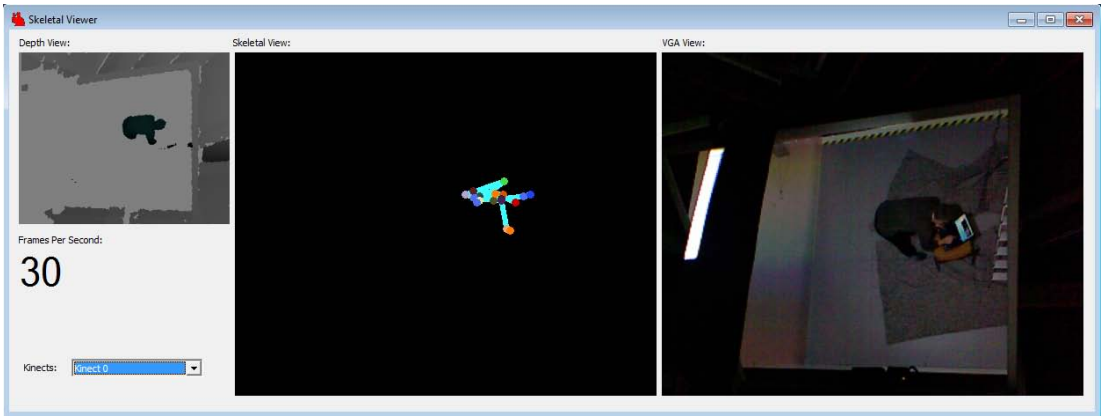


Fig. 4.37 Kinect placed at the ceiling of the DAVE: The test application was not able to reconstruct a skeleton for that angle.

All of those positions were tested for suitability. We evaluated the resulting data of the Kinect with a test application "SkeletalViewer" which is part of the SDK to visualize recognized skeletons. The application also shows the camera image and depth map. The position at the ceiling, next to the floor projector makes it necessary for the Kinect to look through a mirror. In principle that is not a problem, but the viewing angle is much too steep for the recognition of the user standing in the DAVE (see Figure 4.37). The test application was not able to reconstruct a skeleton for that angle. Placing the Kinect at the back opening of the DAVE gives much better results for the recognition of skeletons. Users are visible from the back and the software is assuming to see them from the front. This is not an issue for the navigation control. The setup has two disadvantages. First, observers at the outside of the DAVE might occlude the test user. For user studies this would be a problem due to the spatial restrictions of the current DAVE location. Second, the recognition of arm gestures is very limited because the body of the test user occludes the arms in many situations. In the third configuration the Kinect is placed on top of the front wall (see Figure 4.38). The viewing angle to the user is again rather steep, but still acceptable for the recognition of the skeleton. The test user is completely visible only if he stands about 1.5 m away from the front wall. If she moves closer, the feet and legs are outside of the viewing frustum of the Kinect Sensor. The person is never occluded by observers and also her arms are visible as long as she is facing the front wall. We choose this position for the Kinect as the most suitable. Even although both the Kinect Sensor as well as the optical tracking system work with infrared light, both systems work at the same time. With the chosen position, all sensors do not directly see the light sources of the other system respectively, so that they do not interfere.

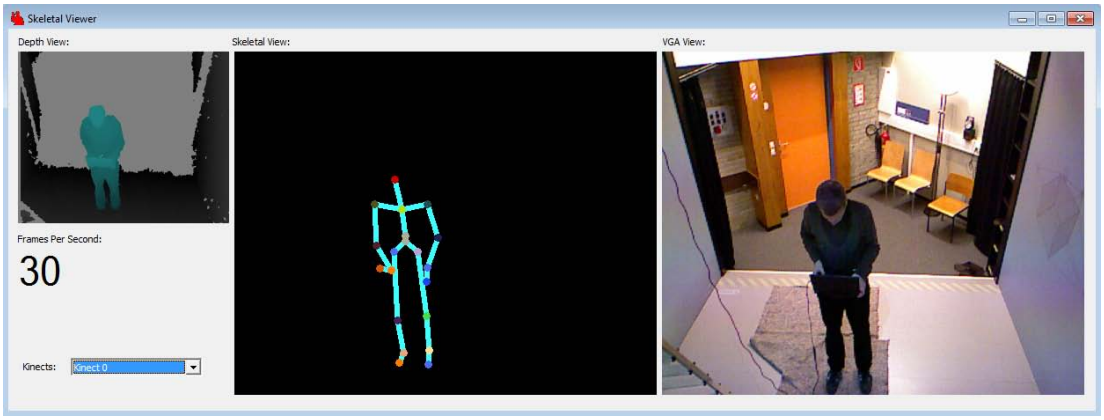


Fig. 4.38 Kinect placed above front wall: The viewing angle to the user is again rather steep, but still acceptable for the recognition of the skeleton.

4.5.3.2 Implementation

Kinect Application

For accessing and controlling the Kinect we use the Microsoft beta software development kit (SDK). To acquire the skeleton data from the Kinect and to use it in our DAVE applications, we modified the test application "SkeletalViewer". The application shows the camera output and depth image and also any detected skeletons. The recognized skeleton data consists of twenty 3D points for twenty joints of the human body. Arms and legs are divided into three segments, the head is one segment and the hip consists of two segments. With our modification the joints data is made available over a standard TCP/IP socket to the DAVE controller application. This is done analogous to the optical tracking system.

Calibration of the Kinect

The test application is working within the coordinate system of the Kinect. 3D data is sent without any modification. To match the coordinate system of the DAVE, we define a transformation matrix composed by two small rotations ($< 45°$) and a translation. Scaling is not necessary as both coordinate systems are based on meters. To calibrate the Kinect we use the already calibrated optical tracking system of the DAVE. With the known position of the head in both systems it is easy to calculate the required transformation. The user has to stand in the middle of the DAVE with his arms spread to form a T-shape. Now the head position and the midpoint of the feet represent the up vector of the Kinect coordinate system and the elbow joints can be used to calculate a vector pointing to the right. Those two vectors are used to calculate the pitch and yaw rotation. The latter is necessary because the Kinect is not placed at the exact center of the front wall. Finally, a translation offset is calculated using the rotated head joint of the Kinect data and the head position of the optical tracking. While other methods can lead to a much more precise registration, this method is already sufficient for our purposes.

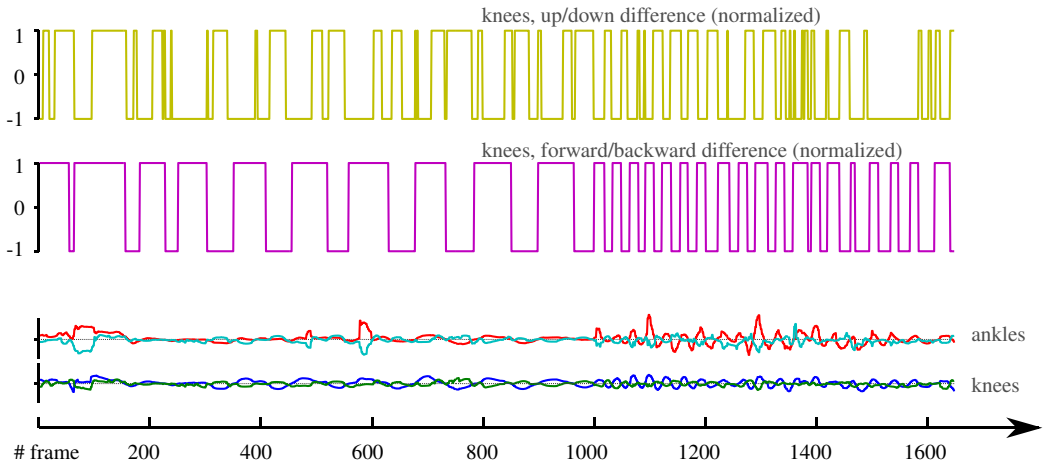


Fig. 4.39 Selected test data of a person walking in place, recorded by the Kinect.

Test Data for Movement

To determine the correct gesture for forward movement we recorded some test data of people walking in place in the DAVE. The test person was told to walk slowly for 30 seconds and then walk faster for another 30 seconds. The joints positions were recorded and analyzed for obvious repeating patterns. The positions of the feet turned out to be too noisy to be usable. The knees positions show a more favorable pattern. Figure 4.39 shows the up/down motion (blue) of the knees and the forward/backward motion (green). It also shows the same data for the ankle (red and turquoise). Even for a simple sign function of the knee position relative to the mean, the resulting patterns are distinct (yellow and pink).

Navigation Control Module

The navigation control module consists of two functions:

1. Rotation, centered at the test person
2. Moving forward

The rotation is realized using a combination of the shoulder joints and the hip joints. Shoulder orientation weights twice the hip orientation. If the person rotates his shoulders to the left, the navigator starts rotating the world to the right. By also rotating the lower body to the left, the speed of rotation to the right is increased. This technique allows the user to freely look around without influencing navigation. Visual feedback is given in the form of an arrow on the floor pointing towards the recognized direction. As the Kinect can only recognize the skeleton of a person if the legs and arms are not occluded, the maximal allowed rotation angle is less than 45 degree. If the person is facing one of the side walls, self-occlusion prevents the Kinect to accurately determine the correct rotation angle. To move forward in the virtual world, the user has to physically move his legs up and down. The local distance of the knees in the direction of traveling is used to trigger impulses of movement. The height of the feet influences the amount of force for the impulse. To prevent a jerky movement, the forward motion does not stop abruptly but simulates a simple kind of inertia as long as the legs are moving. Otherwise, a damping actively slows down the forward motion.

4.5.3.3 Pilot-Study

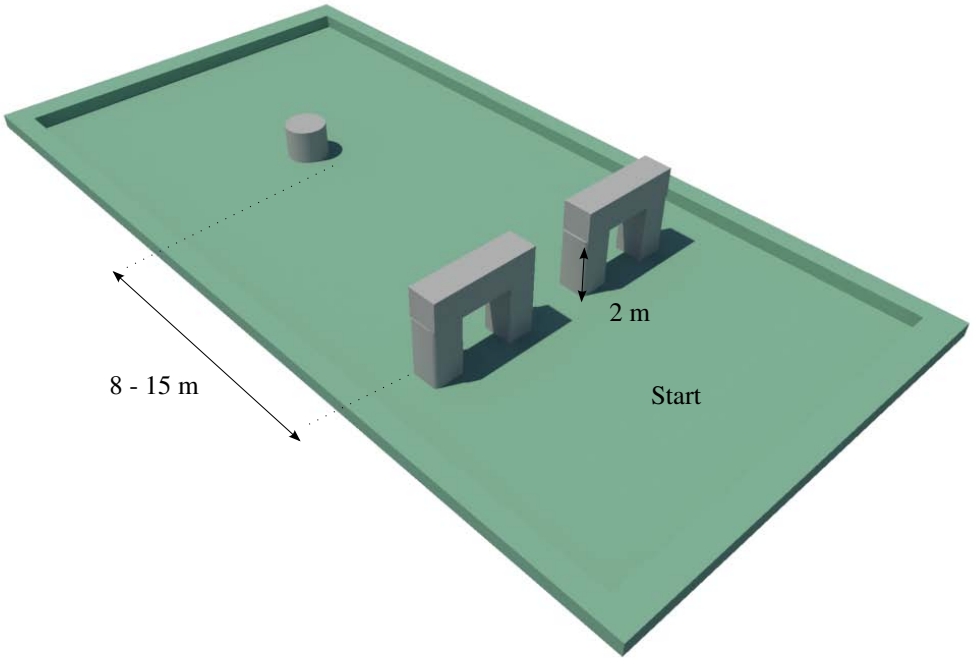


Fig. 4.40 Simple test scenario for the pilot study. The distance of the round column is adjustable between 8 and 15 meters.

A simple scenario was created to test the navigation control module. The scenario consists of a large planar area with two archways and a column (see Figure 4.40). The distance of the column is adjustable between 8 and 15 meters. 14 users (4 female, 10 male) attended the pilot study. Before walking through the virtual world the test persons were asked to estimate a distance of 6 meters in the real world only by vision. The results were kept secret in order to avoid influences of the estimation in the virtual world. The task in the DAVE was to walk through one of the archways, turn around at the column and walk back through the other archway. Seven persons used the Kinect navigation method and the other seven a pointing device. So half of the test group had to move their legs to move forward and the other half just pressed a button. Afterwards the test persons were asked two questions:

- Did you have difficulties navigating through the archway?
- How long is the distance from the archway to the column?

In the result the estimation of the real world distance was too low: the mediate estimation was 5.19 meters with a standard deviation of 0.699. The average error was 15.6 %. All of the test persons had no problem navigating through the archway. The estimation of the distance to the column is shown in Table 4.1.

The average error in estimating the distance is 23.1%. For the joystick navigation only it is 20.9% and for the Kinect navigation only it is 25.2%. The Kinect navigation had no relevant

Person	Navigation	actual distance (m)	estimated distance (m)	Difference (m)	Real world estimation of 6 m
1	Kinect	15	12.0	3.0	4.40
2	Kinect	15	12.0	3.0	4.72
3	Kinect	10	10.0	0.0	4.50
4	Kinect	15	10.0	5.0	5.15
5	Kinect	15	7.0	8.0	5.60
6	Kinect	15	20.0	5.0	4.65
7	Kinect	15	17.5	-2.5	5.15
8	Joystick	10	7.0	3.0	5.32
9	Joystick	15	13.0	2.0	4.80
10	Joystick	15	13.0	2.0	6.25
11	Joystick	15	12.0	3.0	5.10
12	Joystick	10	8.0	2.0	4.40
13	Joystick	10	13.0	-3.0	6.48
14	Joystick	15	12.0	3.0	6.18
standard deviation, total: 2.86, Kinect: 3.47; Joystick: 2.14					

Table 4.1 Estimated distance in meters in the virtual world.

effect on the ability to estimate distances. Probably the test scenario was too small for the walking movement, too. Most of the test persons estimated the distance by looking at the geometry and not by the walking time or physical stress. A test with a longer travel distance should be used in further studies. Nevertheless the first pilot study shows that in general the navigation task can be accomplished without holding any controller device or using the arms or hands for traveling.

4.5.3.4 Insight

Using the Microsoft Kinect in our four-sided DAVE, we designed and implemented navigation and movement controls using the user's gestures and postures. Our technique is completely vision-based. Therefore, the user does not have to learn how to use a new device. This is a more intuitive interaction and more related to real walking than commonly used approaches based on special input devices (3D joystick, cyber gloves, etc.), which compromise the user's VR presence. The approach is already implemented and tested with respect to the perceived realism. The perception of distances and walking time will be the focus of further investigations in this respect. This navigation builds the cornerstone for the development of a test lab that can be used in order to evaluate the effectiveness of smartphone based indoor navigation systems already in the planning phase.

4.5.4 Self-paced exploration of the Austrian National Library through thought

This section was published in 2007 in the International Journal of Bioelectromagnetism [LSFP07]. The results of a self-paced Brain-Computer Interface (BCI) are presented which are based on the detection of senorimotor electroencephalogram rhythms during motor imagery. The participants were given the task of moving through a virtual model of the Austrian National Library by performing motor imagery. This work shows that five participants which were trained in a synchronous BCI could successfully perform the asynchronous experiment.

4.5.4.1 Introduction

A Brain-Computer Interface (BCI) analyzes the brain activity and transforms the electroencephalographic (EEG) changes into control signals [WBM*02]. Therewith it is possible to establish a direct communication channel between the human brain and a machine which does not require any motor activity. Different EEG signals can be used as input to a BCI, either event-related potentials (ERPs), such as slow cortical potentials [BHKN03], P300 potentials [DSW00] or SSVEP [MPSBP05], or transient oscillatory changes in the ongoing EEG [PN01, PNB05, WBM*02]. Up to now most BCI's operate in a cue-based or synchronous manner, in which the BCI system presents a cue and the subject performs a mental task after this cue [GSN*01]. The EEG signals are analyzed and used for control only in a predefined time window after the cue. By contrast, an asynchronous or self paced BCI is constantly analyzing and classifying the ongoing EEG activity [MB00, SLS*08]. In addition to detecting the intentional motor imagery tasks (MI) the system should also be able to detect if the user does not wish to generate a control command (non-control state, NC). In contrast to a synchronous BCI where the performance may be stated in terms of e.g. the classification accuracy, the performance of a self-paced BCI is difficult to evaluate. For computing performance rates, it is necessary to access the subjects "real" intent and to compare it to the BCI output. Unfortunately, this information is not directly accessible. So either (i) the user is immediately reporting whether a command or control signal occurred correctly or not [BMB06] or (ii) the task given to the subject requires activity periods (MI states) and pause times (NC states) and the successful accomplishment of the task can be used as the performance measure [LFMP*07, SLS*08]. Such an approach with defined activity and pause times is termed "experimenter-cued asynchronous BCI." Therefore, in this study an experiment with such requirements was designed within a virtual environment (VE). The use of a VE ensured that the subject was motivated to perform the experiment and that these activity and pause periods could be incorporated into the goal of the task. Therefore the participant was placed in our multi-projection-based stereo VE system, the "DAVE". The participants had the task of moving through a virtual model of the Austrian National Library by performing motor imagery.

4.5.4.2 Material and Methods

The System

In order to carry out the experiments two different and complex systems had to be integrated: the BCI and the DAVE (Definitely Affordable Virtual Environment); see Fig. 4.41. Both systems run on two different machines (hardware) and different platforms (software).

A BCI-system is, in general, composed of the following components: Signal acquisition, preprocessing, feature extraction, classification (detection), post processing and application interface (left side of Figure 4.41). The signal acquisition component is responsible for recording the electrophysiological signals and providing the input to the BCI. Preprocessing includes artifact reduction (electrooculogram (EOG) and electromyogram (EMG)), application of signal processing methods, i.e. low-pass and / or high-pass filter, methods to remove the influence of the line-frequency and in the use of spatial filters (bipolar, Laplacian, common average reference). After preprocessing, the signal is subjected to the feature extraction algorithm. The goal of this component is to find a suitable representation (signal features) of the electrophysiological data that simplifies the subsequent classification or detection of specific brain patterns. There are a variety of feature extraction methods used in BCI systems; a non exhaustive list

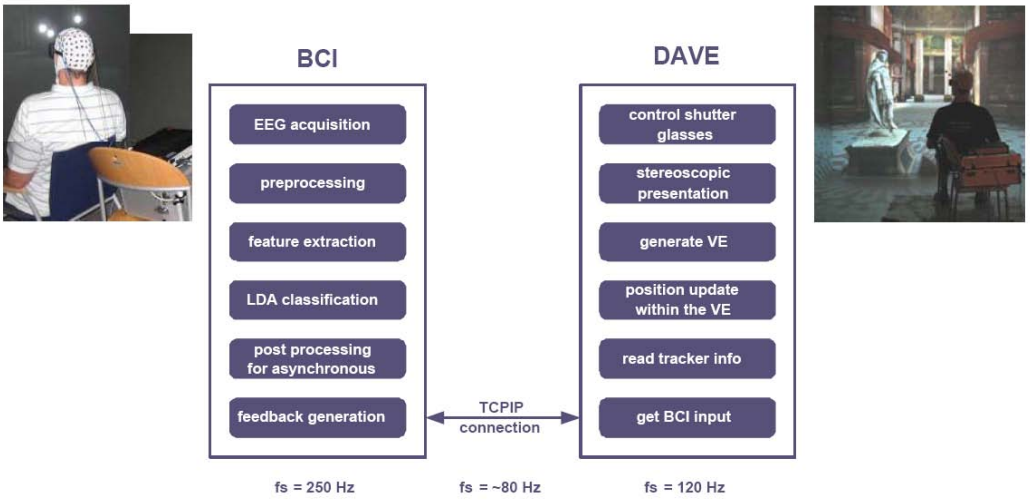


Fig. 4.41 System diagram of the hardware setup. The BCI system on the left analysis the EEG signals and the extracted control commands were transferred into movements with the VE projected in the DAVE system.

of these methods includes amplitude measures, band power, Hjorth parameters, autoregressive parameters and wavelets. The task of the classifier component is to use the signal features provided by the feature extractor to assign the recorded samples of the signal to a category of brain patterns. In the simplest form, detection of a single brain pattern is sufficient. This can be achieved by using a threshold method. More sophisticated classifications of different patterns depend on linear or nonlinear classifiers. Post-processing issues such as dwell time (time in a certain state before an event occurs), refractory period (time after an event), combination of classifier and time dependent modeling uses the pre-knowledge of the actual experiment to adapt the classifier output to the current experiment. In the case of an asynchronous or self-paced BCI not only between the different intentional motor imagery tasks but also the non-control state has to be identified. No output should be generated while detecting the NC state and control commands during the MI tasks. The final output of the BCI, which can be a simple on-off signal or a signal that encodes a number of different classes, is transformed into an appropriate signal that can then be used to control a VE system.

The used Graz-BCI system [PMPS*07] consisted of one biosignal amplifier (g.tec, Guger Technologies OEG, Graz, Austria), one data acquisition cards (E-Series, National Instruments Corporation, Austin, USA) and a standard personal computer running Windows XP operating system (Microsoft Corporation, Redmond, USA). The recording was handled by rtsBCI, based on MATLAB 7.0.4 (MathWorks, Inc., Natick, USA) in combination with Simulink 6.2, Real-Time Workshop 6.2 and the open source package BIOSIG (`http://biosig.sf.net`). The EEG signals were sampled and processed with a sampling frequency of $f_s = 250$ Hz.

The DAVE (right side of Figure 4.41) is described in Section 4.2.3.1 in more detail (see also Figure 4.42a). The stereo images are displayed for the right and left eyes on the respective wall with 60 Hz each (see Figure 4.42b). The projections in the DAVE are continuously adapted to the movements of the visitor by recomputing the projected images for the respective current viewing position and direction (update rate of 30-50 times per second). Thereby, the position of the subject is determined using four infrared cameras that keep track of a number of highly retro-reflective balls that are attached to the shutter glasses. It is vital that the im-

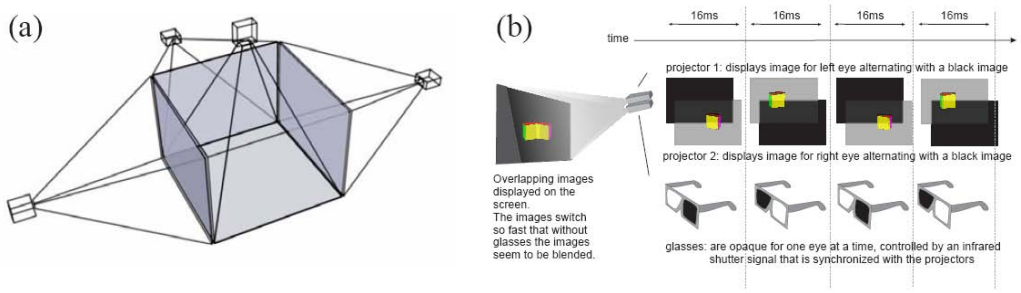


Fig. 4.42 (a) Physical Dave installation with indicated projection screens and (b) principle of time interlaced projections with shutter glasses so that each eye gets a different image.

ages from different screens meet seamlessly at the common border. The projectors are fed by 8 render clients, reasonably powerful off-the-shelf Linux-PCs equipped with the latest high-end graphics boards. Rendering is coordinated by the DAVE server. The VE presented was a model of the Austrian National Library (see Figure 4.43). It was modeled in Maya and 3D Studio (both from Autodesk Inc., San Rafael, CA, USA), but the statue was built using a photogrammetric 3D reconstruction (VRVis Research Center for Virtual Reality and Visualization, Ltd., Vienna, Austria). In total approximately 120.000 faces and 60.000 vertices together with 30MB of texture were necessary to create such a photo-realistic model of the 80 meter long and 14 meter wide main hall (center area 25 meter wide, see Figure 4.44) [SZZK05, SUF07]. The VE application was implemented using the scene graph library OpenSG. This software library hides away the complexity of developing software for a synchronized distributed system of nine computers behind a concise interface and ensures that all clients are provided with up to date observer coordinates and that they render the common 3D scene at the same time.

In all experiments the subjects were sitting in a comfortable chair in the middle of the DAVE (see Figure 4.43). An electrode cap (Easycap, Germany) was fitted to the subject's head, and seven Ag/AgCl electrodes were mounted over the sensorimotor cortex, either as three bipolar (electrodes located 2.5 cm anterior and posterior to C3, Cz and C4, respectively, according to the international 10–20 system [Jas58]) or as one Laplacian channel (over C3 or C4). The ground electrode was positioned on the forehead (position Fz). The EEG was amplified (sensitivity was set to 50 μV for bipolar derivations and 100 μV for Laplacian derivation; power-line notch filter was activated) and band pass filtered between 0.5 and 100 Hz (EEG acquisition and preprocessing block, see Figure 4.41). From these recordings logarithmic band power features (Butterworth IIR filter of order 5) were calculated sample-by-sample for 1-sec windows (feature extraction, see Figure 4.41) and classified with a linear discriminant analysis (LDA classification, see Figure 4.41) (further details about the Graz-BCI see [PMPS*07]). The post processing generated a control signal only when the LDA output of the specified MI was exceeding a selected threshold for a predefined dwell time [TGP04]. The detected events were transferred into control commands for the VE on a sample-by-sample basis of 250 Hz.

The DAVE system was connected to the BCI system via a wireless TCP/IP connection (see Figure 4.41). Approximately 80 times per second the DAVE requested the current BCI output. Thereby relative coordinate changes ($\Delta speed$ and $\Delta rotation$) were transmitted. Together with the current position of the subject within the VE and the tracking information of the subject's head (physical movements) the new position within the virtual world was calculated. Nevertheless the visual presentation of the scene was rendered with 120 Hz (60 Hz for each eye). The

Fig. 4.43 Participant with electrode cap sitting in the DAVE inside a virtual model of the main hall of the Austrian National Library.

whole procedure resulted in a smart forward movement through the virtual library whenever the BCI detected the specified MI..

The Experiment

Five subjects participated in this experiment. All subject started with a synchronous BCI training (two MI classes). During this training they learned to establish two different brain patterns by imagining hand or foot movements (for details see [LLK*07, MPSP07]). The frequency bands for the logarithmic band power features were individually adapted for each subject [PMPS*07] and are given in Table 4.5.4.2. After offline LDA output analysis, the MI which was not preferred (biased) by the LDA was selected for self-paced training (see Table 4.5.4.2). Each time the LDA output was exceeding a selected threshold (Th in Table 4.5.4.2) for a predefined dwell time (Tdwell in Table 4.5.4.2) the BCI replied the DAVE request with a move command (speed = 1.5 m/s and rotation = 0.9 °/s). The task of the subject within the VE was to move through motor imagery towards the end of the main hall of the Austrian National Library along a predefined pathway (see Figure 4.44). The reason for the curved path was the location of the two marble column rows and the position of the statue. The starting point was at the entrance door and the subject had to stop at five specific points indicated in Figure 4.44 (entrance, column row, statue, column row and exit). After a variable pause time (between 20–95 seconds) the experimenter gave a command and the subject started to move as fast as possible towards the next point.

Each subject performed six runs, in which the BCI was constantly analyzing the EEG activity and MI and NC tasks were detected continuously (asynchronously), but the task itself was cued by the experimenter. The initiation to move forwards was given by the experimenter (verbal cue, synchronous event), but the time necessary to move to the next specific stopping point depended only on the performance of the subject (asynchronous task). The duration of the pause time was given by the experimenter, but the activity within the pause was controlled by the subject. The reason for the experimental design chosen was that in case of an asynchronous or self-paced BCI the performance evaluation is not as easy as in case of a synchronous one. In this experiment, defined periods of moving (activity time) and periods of pausing (pause time) existed. For a perfect performance no MI and therefore no movement should be detected during the pause time, and during the activity time only MI should be detected. Additionally, the time necessary for accomplishing the task should be as short as possible. Therefore such an approach with defined activity and pause times is termed experimentercued asynchronous BCI. Periods of true positives (TP, correct moving) and false negatives (FN, periods of no movement

subject	*EEG-derivation*	*async.MI*	*f* [Hz]	*Th*	*Tdwell* [s]
al4	lap C4	left hand	11-15	0.1	1.0
			24-26		
al9	lap C3	right hand	9-13	1	2.0
			15-17		
			20-25		
al10	lap C3	left hand	9-13	0.5	1.5
			21-24		
x20	bipolar	right hand	8-12 (C3)	0.5	1.2
			24-30 (C3)		
x21	bipolar	left hand	8-12 (C3)	0.1	1.0
			23-27 (C3)		
			12-14 (C4)		
			24-28 (C4)		

Table 4.2 Type of EEG recording, used MI type, the frequency bands of the logarithmic band power, the finally used threshold values (Th) and the dwell time (Tdw) are given for each subject.

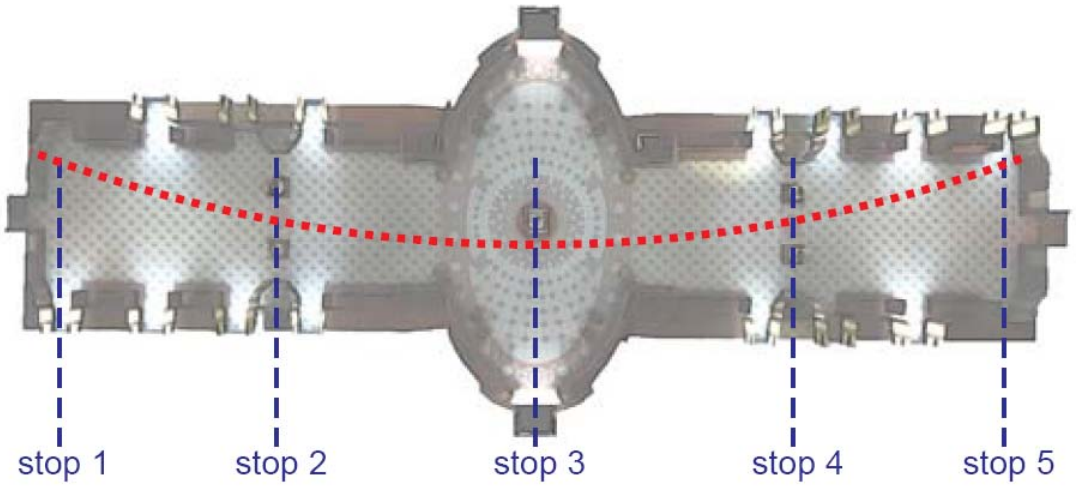


Fig. 4.44 Layout of the main hall of the library. Dotted lines shows the predefined pathway and dashed lines the five stopping points (entrance, column row, statue, column row, exit).

during activity time), as well as false positives (FP, movements during the pause time) and true negatives (TN, correct stopping during pause time) were identified (see Figure 4.45). The true positives rates (TPR) and false positive rates (FPR) were calculated as:

$$TPR = \frac{TP}{TP+FN} \cdot 100\ [\%]$$

$$FPR = \frac{FP}{TN+FP} \cdot 100\ [\%]$$

Following setups were performed in each experiment session:

- Electrode montage (Laplacian or bipolar)
- Check of the EEG quality (via BCI-scope)
- One synchronous BCI run (basket game [KSKP03])

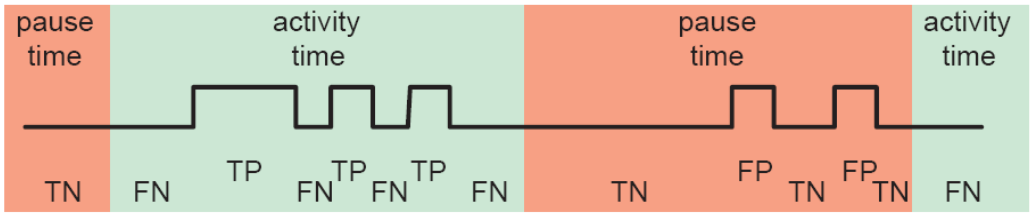


Fig. 4.45 Definition of TP, FP, TN, FN during activity and pause time.

- Update of the LDA-bias if necessary (preferred / biased to one class?)
- One asynchronous jump-and-run game run [MPSP07]
- Update of threshold (for asynchronous detection)
- Six DAVE runs (moving through thought)

4.5.4.3 Results

The online performance of the first run of subject al10 is given in detail in Figure 4.46. In this run the subject needed 226 seconds to reach the end of the main hall, whereby the duration of the pause times at the stopping points were 33.5, 30.5, 29.6 and 32.8 seconds. An additional pause before the first move instruction (3.7 seconds) and after reaching the end of the hall (6.9 seconds) existed. This resulted in a TPR of 50.1 % and FPR of 5.8 %. Each subject performed six runs in the virtual library. The averaged performance (TPR and FPR rates) and the duration of the activity and pause time are given in Table 4.5.4.3. Every run had a different duration, caused on the one hand by the varying pause time given by the experimenter (sum is given in Table 4.5.4.3) and on the other hand on the moving performance of the subject itself. For some subjects it was very easy to increase the LDA output over the threshold (MI detected) and for others the threshold was slightly too high and therefore not so many MI's could be detected which resulted in longer runs. In general all subjects had to move down the total way of the VE, so the same amount of MI time was necessary, only the ratio between MI and NC during activity time was different.

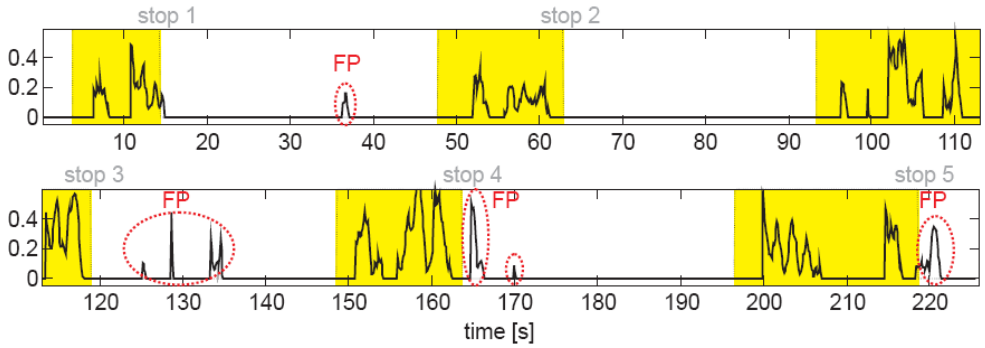


Fig. 4.46 Online performance of the first run of subject al10. The yellow rectangles mark the periods when the subject should move to the next point. The black line is the actual BCI output after the post-processing. Whenever the line is not zero a movement occurred. Periods of FP are indicated with red dotted circles.

	TPR [%]	FPR [%]	Tactivity [s]	Tpause [s]
al4	14.3	2.0	1511	795
al9	17.0	7.1	1219	1081
al10	50.0	4.0	506	885
x20*	13.9	5.8	1025	636
x21	20.8	1.8	1184	1128

Table 4.3 The averaged performance of the navigation task (TPR and FPR), the activity (Tactivity) and the pause time (Tpause) of all 6 runs are given (note: * data of two runs were lost with subject x20, due to technical problems).

4.5.4.4 Insight

In this experiment, a successful application of the Graz-BCI in an asynchronous or self-paced moving experiment with a small number of EEG recordings could be demonstrated. Five subjects were able to accomplish the task to move through the virtual model of the Austrian National Library by motor imagery. Only a small number of FPR (between 1.8–7.1 %) occurred in the experiments. Especially the results of al10 and x21 are encouraging. The numbers of TPR and FPR in relation to the activity and pause time show that there is still space for improvement. A challenge is to optimize the threshold and the dwell time to distinguish between MI and NC states, and thereby to improve TPR and FPR. The usage of a virtual environment created an interesting, challenging and highly visual appealingly task which ensured that the participants were highly motivated and tried to perform their best. Motivation is a very crucial point in such experiments.

One disadvantage of the applied experimental strategy was that the subjects had to perform the motor imagery over a very long period. In the example given in Figure 4.46 the subjects activity time was between 10.5 and 25.6 seconds. Unfortunately oscillatory EEG components need some time to appear and disappear, and subjects are unable to produce changes in oscillatory activity for extended periods of time. Therefore, the reported FN numbers are very high due to this task definition. Nevertheless it is much more import to decrease the FPs than to decrease the FNs. In a real-world situation it would be crucial that the BCI does not spuriously detect an event (FP), but it is not so critical if the subject has to imagine something more than once (FN) before the BCI detects it.

Perhaps in future experiments a different strategy should be used. The BCI could be used to switch the moving on and off, instead of continuing the MI during the movement period. So every time the BCI detects the MI, the status will be toggled. It is currently unknown if the toggling would be manageable and practical for the subject, especially if distinct stops must be achieved at specified points. Another different strategy would be to use every detected MI event to trigger one footstep (additional with a refractory period [TGP04]). This would result in several steps, but not in a continuous movement. A disadvantage of such a strategy would be that the subject could only move or stop on a grid basis, however shorter bursts of imagery could be used.

Unlike other asynchronous BCI experiments, long pause times have been used in these experiments. Pause times of up to one and a half minutes are not often used. Some works previously reported had a pause time of up to only 5 seconds [ZWG*07], up to 7 seconds [BMWB06], up to 8 seconds [BMBB04], up to 10 ± 8 seconds [MPSPR05], up to 17 seconds [SLS*08] or no pause at all [MM03, MB06]. In the work of Bashashati et al. [BMWB06], they described the use of a NC recording of 2 minutes, but did not present any detection re-

sults. The pause time and therefore the NC state is the most important part in an asynchronous or self-paced BCI. The community already showed that it is possible to detect the MI tasks, but the ability to correctly detect the absence of MI over a long period is still an ongoing issue. Patients will be using the BCI for hours or even longer and most of the time they do not want to perform anything. They will only choose to select an operation or perform a task in very short periods, but most of the time the BCI should be idling. Therefore, the pause times used in these experiments are also much too short for real-world applications, however it was an important step in the correct direction.

More detailed information about brain computer interfaces and virtual feedback can be found in Robert Leebs PhD Thesis[Lee09].

4.6 Integration of Legacy Software

In the use of personal computers (PC) – especially in the field of human-computer-interaction – a shift of paradigms takes place. The "Windows, Icons, Menus, Pointer" paradigm of user interface design started in the 1970s. Major developments and improvements have been realized (in alphabetic order) by Apple, Digital Research, Microsoft, Motif, NeXT Computer, Sun Microsystems, Xerox Palo Alto Research Center, and many more. This user interaction is based on a single physical input device (a pointer) controlled by a single user. Information and data are presented by icons and windows. Available commands are compiled together in menus, which can be started via the pointing device. This graphical user interface paradigm is easy to use for both novice and power users. With the usage of tabletop environments, tablet PCs, immersive environments, etc. this single-pointer, single-user paradigm started to change.

In this section we present a technique to cope with the graphical requirements; moreover it solves the integration problem of legacy applications in immersive environments. We use the remote desktop protocol to access a desktop, disassemble it into its components and rearrange/repaint it according to the setup of the environment, in which the desktop is embedded.

4.6.1 Related Work

Previous research has analyzed the influence of display setups to a user's performance in everydays work [BB09]. An overview of large high-resolution screen setups and cluster rendering software has been presented by Ni et al.[NSS*06].

The approach presented in this section is similar to other projects: In 1993 Dykstra introduced the idea of mapping X11 windows as textures onto polygons in a virtual 3D world [Dyk93]. As X11 – the network-enabled windowing protocol for Unix systems [GKM86] – plays a minor role for many end-users following projects concentrated on Microsoft Windows. For example, Regenbracht et al [RBW01] integrated 2D desktops and 3D data sets into a tangible, augemented reality (AR) desktop environment. Besides computer vision and AR techniques they use a modified Virtual Network Computing (VNC) to display 2d desktops on 3d geometry. Also Bues et al. [BBH03] and Nakashima et al. [NMKT05] presented a VNC-based solution.

Depending on the setting in which our approach is used, we utilize various input devices. As this section concentrates on the network and computer graphics part we only summarize the main input methods briefly. We support multi-touch interaction (based on the work by Kaltenbrunner et al. [KBBC05]), 3D-tracked pointing devices, gaming devices and standard input methods (mouse, keyboard). An overview on multi-touch techniques is presented online by B. Buxton [Bux09]. Implementation details of various multi-touch enabling libraries are described by P. Dietz and D. Leigh [DL01], S. Jordà et al. [JKGA06] and B. Ullmer and H. Ishii [UI00].

To access a desktop various network protocols have been designed. On Unix systems the most common solution is the X-server protocol. Enhancements for reduced bandwidth [Pin03], multi-pointer and multi-focuses support [Hut06, HT07] have been made and may be included into future X standards. On non Unix desktop systems the VNC protocol [RSFWH98] and the Remote Desktop Protocol (RDP) [Cor09] are predominant and both are available (server and client) for Mac OSX™and Microsoft Windows™.

The main difference between RDP and VNC is the abstraction layer. All information within the VNC protocol are bitmap based, whereas RDP specifies more abstract information such as

font and text handling, vector graphics and bitmaps as fallback. Consequently, with RDP it is possible to identify window design elements and to adjust them to the destination environment. Furthermore RDP needs less network bandwidth than VNC. Nieh et al. benchmarked RDP and VNC [NYN03]: "Overall, VNC [...is ...] faster at higher network bandwidths, whereas [...] RDP performed better at lower network bandwidths. This suggests that the more complex optimizations and higher-level encoding primitives used by [...] RDP are beneficial at lower network bandwidths when reducing the amount of data transferred significantly reduces network latency."

Having accessed a desktop's components we store them into texture images. Then the rendering can easily be done using any graphics framework such as OpenGL, DirectX, etc. Our rendering solution uses OpenSG [RVB02] – a portable scene graph system to create real-time graphics programs based on OpenGL. Its extensibility, multi-thread safety and clustering capabilities allow customizing the graphics rendering to various graphics environments (from mobile devices to virtual environments and high-resolution projection walls).

4.6.2 Implementation

The Remote Desktop Protocol was introduced with Windows NT™. Once a remote connection is established the client forwards user input to the server, while the server sends drawing commands of elements of the screen to the client. If not disabled explicitly, the server sends updates of all changed graphical elements. Even changes, which are not visible e.g. due to occulsion, are sent to the client by default. The protocol uses bitmap compression and color palettes to keep the amount of data transmitted small. Caching of bitmaps, fonts (glyphs) and desktop screenshots is another way to reduce network traffic within RDP. The current version of RDP (6.1) supports seamless windows. In order to support earlier RDP versions we use a simple alternative developed by Cendio (`http://www.cendio.com/seamlessrdp/`). They provide a server side application (shell) that enables a seamless mode. Furthermore there are patches for *rdesktop* (`http://www.rdesktop.org/`) and *seamlessrdp* available from Fontis (`http://www.fontis.com.au/rdesktop`) allowing the use of one single connection to launch multiple programs.

Based on these techniques and applications – especially rdesktop – we created an object-oriented framework, which wraps the pure C applications in a modular manner. The network, compression and authentication parts are now integrated into an RDP module. This module uses an abstract window factory and abstract graphics/user interactions to work with. These interfaces can be implemented, for example, by classes which map RDP events to a graphical user-interface (GUI) library. In this case the result is a "normal" RDP client (used for debugging purposes). Though the abstract graphics and user interfaces can also be implemented in a different way. We provide a texture-based approach which is integrated into OpenSG. In this manner, we can integrate a Windows™desktop into any environment which is supported by OpenSG.

The RDP screens are drawn into the data fields of OpenSG texture objects which are used in material objects. A shader program applied to the material can create advanced visual effects. The material can be referenced by any geometric object in the scene graph – even more than once. For the geometric representation of a window we normally use a quadrangular plane shape. The geometry is added to the scene graph as child of a transformation node.

The rendering is decoupled from the RDP connection to the hosting system. Therefore, the texture content may be updated independently from rendering frame rates. The overall per-

formance is sufficient for web browsers, word processor programs, spread sheet applications, presentation software, etc.. Fast frame updates for large display areas, for example a video player running in full screen mode, do not perform very well. Also areas of the desktop which are drawn in overlay/direct mode are not accessable for RDP applications.

User inputs have to be sent back to the RDP server. They can be divided into two classes: on the one hand there are user inputs concerning the arrangements of window elements (normally handled by a window manager); on the other hand there are inputs, which are passed directly to the corresponding application to which the window/dialog/etc. belongs. Both input categories now have to be handled in 3D. Instead of 2D screen coordinates an intersection point P_i has to be found, e.g. by shooting a ray along the input device into 3D object space. The intersection point belongs to some geometry in the scene graph and its texture coordinate system maps the intersection point P_i to screen coordinates. OpenSG has built-in methods to test the geometry in a scene graph for ray intersection. This method returns the hit object and triangle as well as the intersection point P_i.

The user input actions can be modified before they are sent to the RDP server. A double click event for example can be simulated with other kinds of actions depending on the input device. On the multi-touch table (see Figure 4.47) we are currently sending mouse clicks for each touch event that happens inside the remote window area. Obviously it is not possible to distinguish between a left click and a right click in this way. There are many ways of how to handle this restriction. One idea is to use gesture recognition. Another way would be to define different clicks depending on the time span between press and release events.

Having all the single desktop components the render system can modify the visual appearance of the desktop. In computer graphics the texture rendering is altered by shaders. It is possible to add special effects not only as an eye catcher, but also for practical reasons. We experimented with a shader for distorting a high resolution screen to fit in an area of lower resolution without clipping.

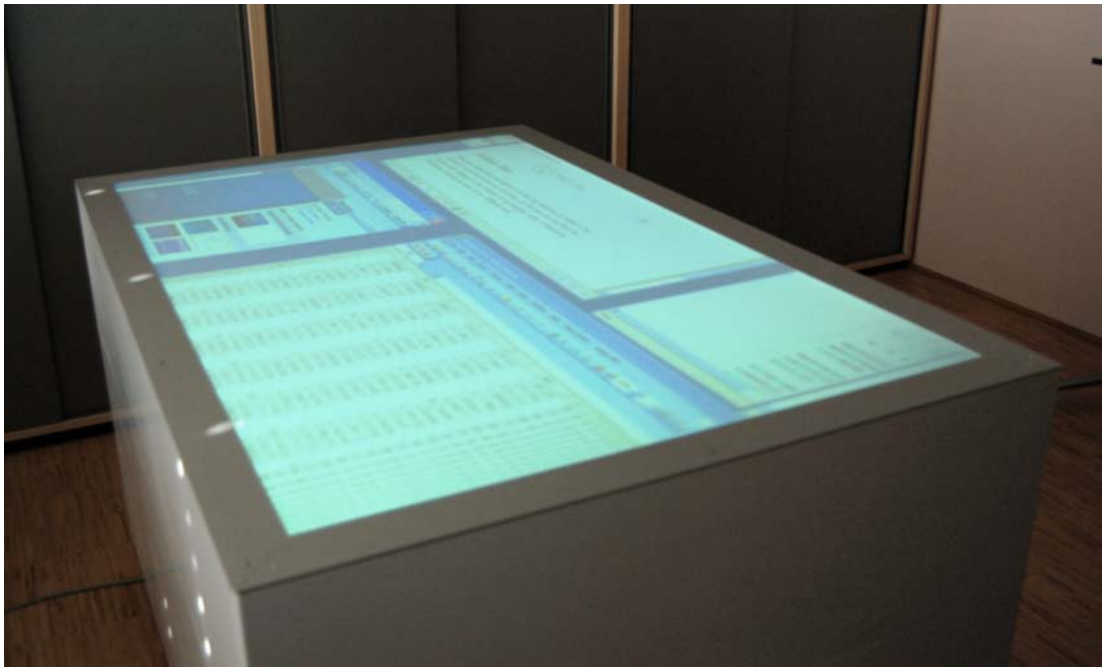


Fig. 4.47 At a tabletop display there is no fixed definition of top and bottom as people can move around the table and use it from any side. Furthermore all users can interact simultaneously.

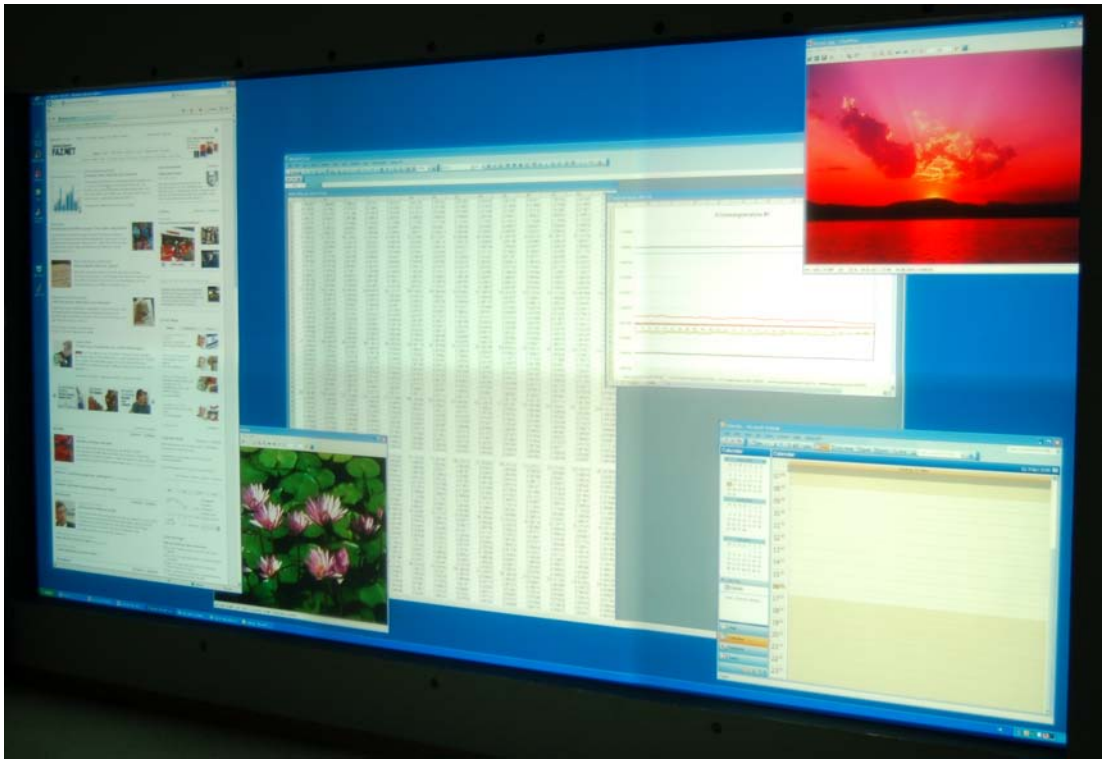


Fig. 4.48 Our tiled display with 3×2 projectors has about 4000×2000 pixels. The rendering is done on 3 PCs which are controlled by a master. Each render slave is connected to two projectors.

4.6.3 Applications

In this section we demonstrate a few examples in which our approach is used.

Desktop in XXL: In a tiled display setting with 4096×2048 pixels the desktop to render runs in a virtual machine. A virtual machine allows configuring the graphics settings independently from any hardware restrictions. Therefore it is possible to set the virtual desktop resolution to values up to 4096^2 (software limit of Windows XP™). Figure 4.48 shows such a high-resolution display with one single desktop rendered by three PCs and displayed by six projectors.

Presentations on Arbitrary Displays: The previous example tries to make the most of the display technology. Another relevant example is a low-resolution one. Using the RDP technology it is possible to access any Windows PC and Mac (RDP server is not installed by default). All applications, which do not use direct overlay techniques – such as video players – or direct graphics hardware access, can be shown on the presentation wall. In this way it is possible to show, e.g., PowerPoint™presentations, which are running on a simple laptop.

Desktop in 3D: Our approach accesses the single components of a desktop separately. A functional hierarchy (parent window $\rightarrow$ child dialog) is mapped to a scene graph hierarchy in OpenSG. As the default object space of OpenSG is three-dimensional the 2D desktop can be embedded into 3D. Figure 4.49 shows such a setting. The desktop is integrated into a CAVE-like virtual environment. The 2D structure is dissolved and merged with virtual reality. A tracked pointing device with buttons allows clicking into the 2D desktop. Windows can be arranged freely in the 3D space by clicking on the edges of the container geometry.

Fig. 4.49 2D windows are embedded into a 3D scene using a CAVE environment. The container geometry can be arranged freely in the 3D space.

Desktop Virtualization: A virtual environment does not need to be three dimensional. Using a full-featured rendering framework – such as OpenSG – our approach can visualize a virtual desktop of higher resolution than the display resolution. This concept of virtual screens can be used, e.g., on a multi-touch table. There we render a virtual desktop, which has three times the resolution the multi-touch system has. Each screen is a little bit smaller than the system resolution so that if one screen is displayed 1 : 1 some space is left. The free space is used to display down-scaled version of the remaining screen of the desktop (to the left or to the right as appropriate).

4.6.4 Insight

This section describes a system for rendering legacy Windows software directly in various graphics environments like tiled displays, tabletop surfaces and CAVEs. The system uses an implementation of the Remote Desktop Protocol to fetch a desktop or single applications. The RDP server can run on a separate PC or as a virtual machine. Applications can be used without modification. We showed examples running on a tiled display with a high resolution, in an immersive environment and on a multi-touch table. With the possibility to freely transform application windows we can solve the orientation problem of tabletop screens. The system can also be used to manipulate the visual appearance of a desktop. User inputs can be mapped from various input devices to the common keyboard and mouse input events that the legacy software

can process. The general restrictions of common desktop applications are not solved. To allow multiple users to work together in one application, substantial modifications on a lower level are necessary. The applications and also the operating system would need to allow multiple inputs simultaneously and multiple applications being in focus at the same time.

Chapter 5
Applications and Projects

Abstract The work flow of content creation, semantic enrichment and interactive presentation is now complete. The presented concepts and techniques are used in research projects and applications. In the following chapter some of the projects and applications are presented.

This chapter presents some of the applications and projects related to the work of this thesis. It is divided into three sections representing three areas of application for interactive 3D graphics: Cultural Heritage, Security and Mobility. Each of them benefits from semantically enriched 3D data.

5.1 Cultural Heritage

Antoine de Saint-Exupry (∗29 June 1900, †31 July 1944) belongs to the most popular representatives of modern French literature. His book Le Petit Prince (The Little Prince) became world famous. His definition of civilization is a follows:

"A civilization is a heritage of beliefs, customs, and knowledge slowly accumulated in the course of centuries, elements difficult at times to justify by logic, but justifying themselves as paths when they lead somewhere, since they open up for man his inner distance."

Each civilization wants to preserve its paths back to its roots because the legacy from the past is an irreplaceable source of inspiration and an indispensable element of understanding that civilization's history. The demand to preserve mankind's historical artifacts is a body of thought that is gaining more and more acceptance in society. Using state-of-the-art information technology, the limited resources made available to conserve cultural heritage may be stretched further. The technical process to acquire historical artifacts is a nontrivial task even if the questions of preservation, restoration, and refurbishment shall remain unanswered within this section.

The main contribution of the following sections is the integration of 3D models into digital cultural heritage libraries. Especially the sustainable linking of semantic knowledge with 3D data is one of the challenging tasks in this area. The described solutions and applications are part of the results of the EPOCH project (2004 - 2008).

"EPOCH is a network of about a hundred European cultural institutions joining their efforts to improve the quality and effectiveness of the use of Information and Communication

Technology for Cultural Heritage. Participants include university departments, research centres, heritage institutions, such as museums or national heritage agencies, and commercial enterprises, together endeavouring to overcome the fragmentation of current research in this field." (EPOCH web site)

EPOCH was funded by the European Commission under the Communitys Sixth Framework Programme, contract no. IST-2002-507382. The insights of the EPOCH project helped to formulate the targets of the 3D-COFORM[1] (3D collection formation) project. The 3D-COFORM project carries on the work of EPOCH and advances the state-of-the-art in 3D-digitization and 3D-documentation in cultural heritage.

5.1.1 Integration of 3D Models into Digital Cultural Heritage Libraries

Man-made 3D objects do not exist in isolation. They have a meaning, in particular they have a historic meaning. Historic artifacts are usually involved in a whole number of semantic relations and contexts. Real artifacts are three-dimensional, solid, physical objects, and often very old. Culture on the other hand is an abstract concept. It constitutes itself in the *semantics* of the physical objects: Who has made them, where do they come from, what were they used for, etc.

Small physical differences can lead to great semantic differences when, e.g., the classification of an amphora is based on the shape of its handles, or when the fact that the arm of a statue was created by a different artist is derived from the different chisel strokes. The goal of archeology is to understand ancient cultures only from the remaining physical evidence. **Any further loss of information must be prevented.** So, accurate documentation of physical artifacts and the faithful recording of all (collected and derived) semantic information are of prime importance for all archeological activities.

Photographs and drawings are the traditional means of archeological documentation. Photos are indispensable for their perspective precision, drawings because they allow the author to *interpret* and *emphasize* what was found. Drawings, together with paintings and with physical replicas, are also the main devices to *hypothesize* about the historic past.

Information technology brings huge benefits to archeology. It helps to organize the traditional workflow more efficiently, but it opens also radically new and innovative perspectives. One great option is advanced knowledge management using *digital libraries*: All the tiny information fragments, which are so typical for cultural heritage, can now be interconnected and related to other pieces, forming versatile semantic knowledge networks that can be queried, filtered, visualized, and presented in various different ways.

Another great progress is marked by new types of *digital artifacts*, from digital photography to 3D laser scanning. Digital replicas have great advantages: They are easy to transport, they consume no archive space, require no maintenance, do not have to be cleaned; they can be studied for any duration, by many people at the same time; and finally, at least in principle, they will never fade away.

The main focus of the following is a review to the technical pre-requisites for the integration of digital 3D models into Digital Cultural Heritage Libraries. It is published as *"On the Integration of 3D Models into Digital Cultural Heritage Libraries"*.

As described earlier, the challenge is to connect the three-dimensional digital artifacts ("3D-models") seamlessly with the various other pieces of information. Furthermore we aim at a digital library integration of 3D not just as a proof-of-concept, but on an ***infrastructure*** level

[1] http://www.3dcoform.eu

by providing a reasonably general, broadly applicable software framework. We argue that it is not sufficient to treat 3D-models as anonymous 'BLOB's (Binary Large OBjects), which is the conventional way to deal with them. A major step forward, and the only satisfactory solution, is to store semantic information *within* the 3D-model, both exploiting and respecting its inherent three-dimensional structure. Only if a markup of parts and regions in a 3D model is possible, then these parts and regions can also be referenced. A reference, in the sense of a hypertext link, needs an URI and an anchor as the link target. The goal is that a part of a 3D model can be retrieved using a common web search query such as

http://www.epoch-net.org/DL/search?q=venus+milo#head[2]

This query might retrieve the complete 3D model of the *Venus De Milo*, show an overview, and then, smoothly flying, zoom in to its head, which is discreetly highlighted. This requires that the 3D viewer on the client side understands the semantic information. The viewer architecture is therefore a vital part of the infrastructure, and the main technical contribution of this section, explained in detail later on.

A 3D model will typically have to accommodate different kinds of semantic information. Some of the standard formats, Dublin Core and CIDOC-CRM are described in Section 3.2. In addition to those standards the processing history documentation is an important part of information in cultural heritage. Digital raw data typically undergo many processing steps before being published or archived: Images are white-balanced, cropped, sharpened or smoothed. Heavy image processing is done without any special notice by every digital photo camera, and by the driver of every digital flatbed scanner. When taking data from an archive it is vital to know how they have been recorded and processed, in order to judge their fitness for a specific purpose: A photo-montage cannot be taken as evidence for a historical fact, judging colors without knowing the white balance is pure guessing.

Accurate recording of the processing history is even more important for 3D objects than for images because the spectrum of 3D editing operations is much wider.

5.1.1.1 The ambiguous notion of 'a 3D model'

The term "3D-model" is in fact used for a number of different things. Most of the following entities can have meta-information attached, such as Dublin Core, CIDOC-CRM or the processing history.

- **range map**: single raw scan, basically a photo with depth information. One z-value per pixel: $2\frac{1}{2}D$, height field.
- **scanned artifact**: dense triangle mesh. Several range maps merged and edited: fill holes, remove noise, etc.
- **synthetic reconstruction**: constructed by an interactive 3D drawing or CAD program, usually very clean model
- **3D scene**: many objects, often structured hierarchically: city→ house→ floor→ room→ table→ cup→ spoon
- **animation**: deforming shapes, solid objects moving along a path, particles, articulated skeletons
- **interactive experience**: anything from the limited interactivity of VRML routes to high-end computer games

[2] notation example, not a working URL

The digital counterpart of a historic artifact from a museum exhibition would be the scanned artifact: an amphora, a sword, a sherd. Two complications arise: First, the scanned artifact is not the natural "atomic unit" as in conventional archeology. It is already the product of a process (range map merging). Second, a range map, and thus the scanned artifact, contains typically a number of objects that are captured in the same "depth photo". So, digital acquisition already *starts* with compound objects. Consequently it requires means to distinguish parts of a model. This requires a markup facility on a *sub-object level* (see Figure 5.1).

Multi-object markup, on the other hand, makes sense as well: For grouping similar objects together (columns of a Greek temple), and also to denote whole ensembles of objects; from the arrangement of burial objects to the hypothetical formation of the troops of an ancient battle.

Fig. 5.1 Creating a scanned artifact. Simplified versions of the range maps, textured and un-textured. A rectangular region is marked in different triangulations. Several range maps are integrated and smoothed. The gravestone is manually segmented for a semantic markup.

5.1.1.2 The great variety of 3D surface representations

Computer graphics provides many different ways to represent one and the same three-dimensional object. This is a fundamental difference to other multimedia formats such as sound, images or video: All bitmap formats describe the same thing, a regular grid of rectangular pixels; and every video format boils down to a stream of images. The difference lies only in the encoding (lossy/non-lossy, etc.).

There is no conceptual "master representation" to which all 3D formats are just an approximation (compare with Section 2.1). All representations have their strengths and weaknesses, and also their preferred application domains. Furthermore each representation comes with its own set of diagnostic routines and editing operations. Removal of statistical noise makes sense for discretely sampled surfaces (point clouds, triangles). Continuous surfaces (parametric patches) provide excellent diagnostic tools for high-quality fairing and optimization, while primitives are very handy to roughly discriminate object parts: A door may be just an anonymous hole in the wall, but it can easily be distinguished with a door frame made of three boxes.

This heterogeneity has important, wide-ranging consequences for 3D markup. First, it is not clear *what* to denote. A notion that is intuitively simple and unambiguous, such as the *cheeks of the Venus De Milo*, can become quite complex on the technical level: Triangles are transient objects, and they provide only a very fragile reference. But the surface representation can also not be neglected: Delicate structures such as ridges, creases, corners etc. are *surface features* that are very important for the semantics of a shape. We formulate this as the following open problem.

Problem 1: Generic, stable, and detailed 3D markup.
A method to reference a portion of a digital 3D artifact, irrespective of the particular shape representation used, in such a way that also detailed surface features can be discriminated. The markup should survive simple editing operations (cutting, affine transformations). In case of more complex shape operations there should be a well-defined way to update the reference accordingly, e.g., with a recomputed surface feature.

The drawback of the requested *generic* markup is, of course, that it cannot exploit the specifics of a particular shape representation ("the largest triangle", "vertices contained in a box") but only intrinsic surface properties ("the point of maximal curvature").

5.1.1.3 The variety of 3D editing and processing methods

It cannot always be avoided that shape is edited although it carries a markup. Example: A complex 3D artifact is assembled using parts from different sources (photogrammetry, scanning campaigns, manual repair). Much later somebody discovers an important shape detail, cuts it out from the larger shape (manual segmentation), and sends it electronically to an expert. It is of course vital for the expert to trace back the provenance of the different parts of the shape.

Remark: This example may seem far from practice. There are probably huge numbers of *real* artifacts in museums and archives with dubious provenance and processing history. It is not acceptable, on the other hand, that the annoying loss of information that was inevitable with the procedures of the past should still continue with the digital workflows of today. Managing huge amounts of administrative information has become feasible. The challenge is now merely to set the appropriate standards.

If a shape is edited that carries a markup, then the shape reference must be updated. How can this be done if the referencing method (solution to Problem 1) is not known beforehand? Well, every referencing method can be evaluated to obtain a list of geometric primitives. **The editing method has to keep track of the affected geometric primitives**. Then the primitives can be converted back to a reference (of the same type). Example: Some part of a triangle mesh of a temple is marked "ionic column" using a bounding box. In order to cut out the capital of the column, the triangles inside the box are determined. After the cut, a new bounding box is generated around these triangles to update the reference to restore the "ionic column" markup.

But how can the fact be recorded that the column capital was cut out from the model of a temple?

Problem 2: Provenance and processing history record.
First, a standard for describing the sources of digital 3D data. Second, a standard way of recording how the source data have been processed, and how they were combined to obtain the resulting 3D dataset. Ideally the processing history is *complete*, i.e., it has the **re-play property**. This means that it permits to re-generate the result, also with varied parameters.

The enormous complexity of Problem 2 may be not apparent. First, it has to cope with two levels of heterogeneity, namely the various shape representations and the various operations on these representations. Shape editing operations are not canonical. Well, every 3D software has its own great mesh editing functions, its NURBS intersection routines, and its own CSG implementation. Second, with interactive editing it is barely feasible, and hardly useful, to store each single manual processing step. Third, the re-play property is extremely difficult: It requires that all tools in the chain maintain and add to the processing history; and this depends on the weakest link in the chain. Issues such as outdated software versions and incoherent tool chains (operating systems, private solutions, use of scripting languages) are practical problems. More subtle problems have been reported by Pratt in the context of CAD model exchange [Pra04].

This does not mean that Problem 2 is unsolvable. It means only that the infrastructure needs to be particularly flexible with respect to the future solution(s) to this problem.

5.1.1.4 The notorious issue of the right 3D file format

It is surprisingly difficult to find a 3D file format that is suitable for cultural heritage, due to the many demands to meet.

- **Extensibility** to cope with the variety of shape representations, to be open for new shape representations and editing methods developed especially for cultural heritage
- **Digital preservation and long time archival** to avoid that the 3D digital artifacts degrade within a few years, whereas the real artifacts have survived centuries
- **Size-efficient encoding** is indispensable since faithfully recorded 3D datasets contain huge amounts of data
- **Well supported and broadly accepted** since the best format is useless if it is neglected by the user community
- **Open standard** that is adaptable to cultural heritage requirements

A file can become useless in several ways: by *physical degradation* from an unreadable storage medium, or by *format degradation* when an obscure, undocumented binary format was used, or an outdated format version that no software can decipher any more. Finally *semantic*

degradation occurs when data can be read and decoded, but the provenance and processing history is unclear because no metadata are available. A 3D scene may contain many objects from different sources, so that decent metadata are in fact required for each and every object in the scene.

To some extent, well-supportedness and extensibility are contradictory requirements. Commercial standards such as DXF and 3DS are broadly used, but cannot be extended. IGES and STEP are extremely complex open industrial exchange standards, difficult to adapt to the needs of CH and to extend. For similar reasons most other 3D formats can be ruled out. But the example of the VRML standard shows a potential way out of this dilemma.

5.1.1.5 XML based formats

Today the standard solution for extensible data formats is XML. Its greatest practical advantage is that it makes writing parsers obsolete, which is tedious and error-prone. A single parser, the XML parser, is sufficient to support all XML based data formats. The structural integrity of a file can even be tested automatically when a *document type definition* (DTD) is available.

Its usefulness is illustrated by the VRML ISO standard from 1997 [VRM97]. Originating from SGI's *Inventor* format, but designed as extensible (via the infamous *PROTO*), the success of VRML was impeded by the very complex parser it requires. This problem was solved using XML.

The result, X3D, still carries many legacy concepts. The main problem is the ambitious intention to create a general language for the description of "virtual worlds". As a consequence, X3D describes many data that are today part of the application. X3D specifies the navigation (walk/fly/orbit), an event system with triggers and interpolators, and a data flow model (*ROUTE*). The interaction facilities are rather limited, of course, compared to high-end computer games (Doom, Halflife etc.). Things like the outdoor background in a 3D computer game, for example, are so complex that they are realized by the application. The simple VRML backdrop (color gradient or texture) is insufficient. X3D's generalization to the *background stack* on the other hand nails viewers down to a particular background model, a very limiting policy as well.

As described in Section 3.4.1 one possible solution is the Collada format.

5.1.1.6 Separation of structure from content

Although Collada provides the most common shape representations, it provides certainly not the most efficient encoding for, e.g., huge triangle meshes: The specification says that a Collada file may not contain binary data. However, a reference to binary data stored outside the Collada file is allowed (*external data*), such as *"Pantheon_06.obj.gz"* in Figure 5.2. Note that this reference is a URL. So it may also point to a remote resource, i.e., a web link or even a database query, as was requested in Section 5.1.1. This is also the key to using more advanced XML technology (see Section 5.1.1.16).

The decision was taken to store geometric data exclusively as external data, in binary form. This works well since it reflects a clear separation between structure and content. The Collada file contains only the scene hierarchy and transformations, and is typically small. This makes it particularly easy and efficient to create Collada files on-the-fly, e.g., as response to the aforementioned remote database query.

```
<COLLADA>
 <library_nodes>
  <node id="Pantheon">
   <instance_geometry url="Pantheon_XY5.obj.gz"/>
  </node>
 </library_nodes>
 <scene>
  <visual_scene>
   <node>
    <translate> 1.0 2.0 3.0 </translate>
    <rotate> 1.0 0.0 0.0 90 </rotate>
    <instance_node url="#Pantheon"/>
    <extra>
     <technique profile="Epoch">
      <link rel="schema.DC"/>
      <meta name="DC.creator" content="Smith"/>
      <meta name="DC.subject" content="Temple"/>
     </technique>
    </extra>
   </node>
  </visual_scene>
 </scene>
</COLLADA>
```

Fig. 5.2 Collada example. Note the attached DC metadata.

Note that the *digital preservation* requirement demands that only well documented binary file formats may be used for cultural heritage in order to avoid semantic degradation.

The only problem remaining is to how make Collada support an integral part of the 3D infrastructure of cultural heritage.

5.1.1.7 From Collada to OpenSG, and back again

OpenSG is the scene graph system used in the EPOCH network of excellence [Epo04]. As part of the 3D infrastructure OpenSG permits to rapidly create 3D applications, e.g., for museum presentations. It is also useful to quickly add interactive 3D to C++ applications that were previously non-3D. Some of them can be quite demanding with respect to flexible markup and advanced metadata: There are, for instance, management tools for complete archeological sites (e.g., ArchEd, StratiGraph). They would greatly benefit from a striking 3D visualization of the complex three-dimensional structures of an excavation site, and of the various relations between the numerous collected archeological items.

Export, exchange and import of rich cultural heritage visualizations requires that 3D objects can be provided with markup and metadata. It should also be possible to edit these attached data, and to send the modifications back. Consequently, it must be possible to store attachments within the Collada file. Then one option to extend an application technically is

1. to include an arbitrary XML parser to load a Collada file, thereby converting it to a so-called document object model (DOM) tree, the in-memory representation of an XML file,
2. to create a 3D scene graph from the DOM tree

3. and to keep both structures in sync: all editing operations on the 3D scene graph are propagated to the DOM tree,
4. since a DOM tree can easily be exported to an XML file.

The great drawback of this straightforward approach is that the editing operations are application dependent. Thus, the delicate step 3. has to be implemented anew for each application that is to be extended with 3D.

Is there a more generic way to establish a robust connection between the scene graph engine and the important cultural heritage specific attachments?

5.1.1.8 Solution: XML in the scene graph

Our solution is a tighter integration of OpenSG and Collada. This is a delicate decision: A loader usually just parses a file and produces scene graph nodes. The proposal, however, is to actually preserve certain fragments of the file, and to write these fragments verbatim into the scene graph. Of course, this strategy trades greater flexibility with one format against reduced compatibility to other formats: When exporting to a low-level 3D format, a loss of information is inevitable.

We have found a way, however, to reduce this effect. Instead of implementing just 'yet another' Collada loader, we provide a general, clean, and also concise interface (API) that facilitates adding support for *any* XML based file format. So this module is intended to be used by authors of XML file importers. However, we focus on Collada to illustrate the concepts that might be useful also in other settings.

5.1.1.9 Mapping a Collada file to an OpenSG scene

For the reasons described before our Collada importer is deliberately limited. We concentrated on the scene graph aspect and did not include things like animation, shaders or physics, which are also part of the 1.4 Collada specification. Neither does it support geometry, since that comes from external sources, from OBJ, 3DS, WRL and, with our addition, also from Collada files. Note that this way also future custom shape representations, which come with their own optimized loader modules, can be accommodated.

The transformation of a Collada scene graph to OpenSG presents some challenges: Some of the Collada elements cannot be directly mapped to a corresponding OpenSG object.

5.1.1.10 Transformations

The <node> element forms the basis of the Collada scene graph. It can have child nodes of many different types. All children that are transformations are accumulated, in the order in which they occur. The resulting transformation affects equally all other, non-transformation child nodes.

We map one <node> to ≥ 2 nodes of the OpenSG scene graph: One *Transform* node carries the accumulated matrix, and the non-transformation nodes are children (all siblings), see Figure 5.3. It would not help to recreate the chain of transformations since OpenSG knows only one type of transformation (a matrix). The XML text could be conserved, but this runs into trouble when the transformation is changed. As a consequence, the matrix assembly information is lost.

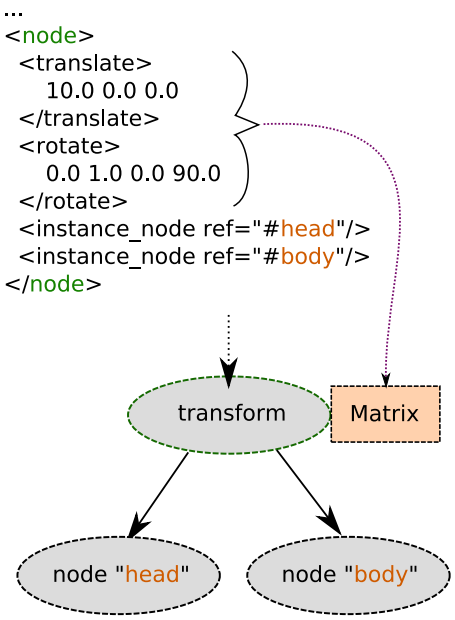


Fig. 5.3 Collada <node> mapped to OpenSG sub-scene.

This shows why the *exact import/export property*, stating that every file remains unchanged when it is imported and then exported, is unfortunately not a realistic requirement.

5.1.1.11 Geometry

For our external content strategy we need only two Collada elements: The <library_node> references the external resource with its *URL* attribute, and it defines a local name. This name can be referenced in the scene by any number of <instance_node> elements using a *relative* URL (Figure 5.2). Unlike an XML file, which is a tree, the OpenSG scene graph is a DAG (*directed acyclic graph*). So sub-scenes can have multiple parents, which comes in handy when a <library_node> references a Collada file. Note that it is vital for nodes resulting from external sources to remember the URL (the file name); otherwise the export will fail.

5.1.1.12 Attached <extra> and <technique> elements

The <extra> element, which is so useful for cultural heritage, can be attached to almost any other Collada element. This is a problem for element types that have no matching OpenSG entity; it is unclear then where to store the attached data. Sometimes dummy nodes can be inserted, like for a <library_node> or for a <visual_scene>, but this strategy does not work in all cases, and it complicates the scene structure.

An <extra> can have children of type <technique> but also <asset>. This is the place for *asset-management information* like <title>, <contributor>, and <keywords>. Although useful in principle this may lead to another complication: cultural heritage requires to use much more sophisticated metadata schemes anyways. This makes the <asset> information redundant, and even potentially contradictory.

5.1.1.13 Attachments

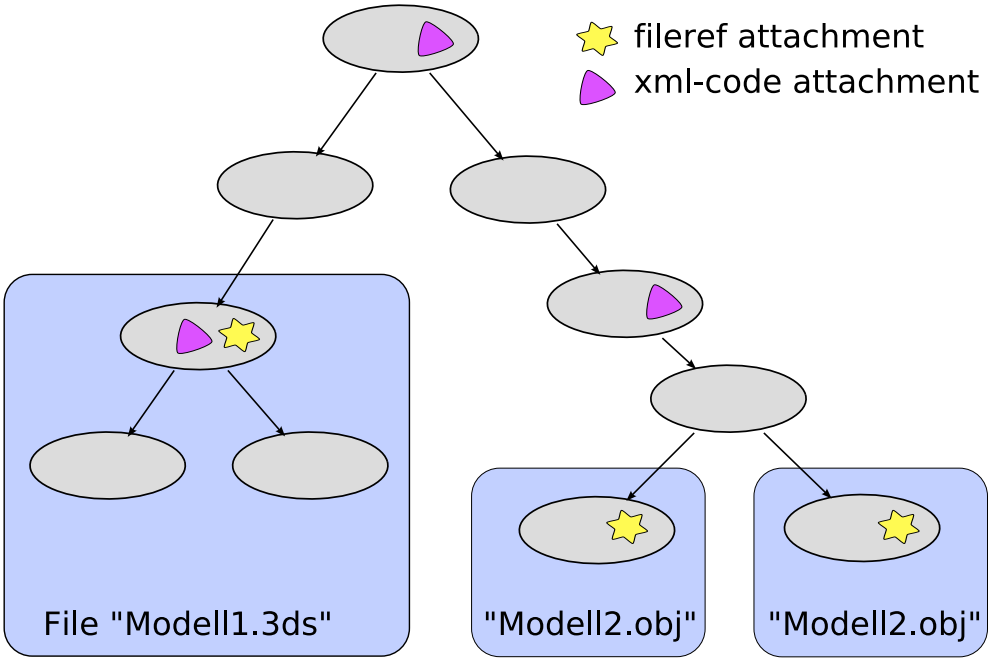


Fig. 5.4 The use of xml-code and fileref attachments.

A great feature of OpenSG is that arbitrary user data can be attached to almost anything. All major *FieldContainer* classes such as *Node*, *NodeCore*, *Image*, *Material*, *Camera*, etc., are derived from *AttachmentContainer*. Whereas the set of fields is fixed and the same for all class instances, each individual instance can carry any number of attachments.

Each attachment has a certain type and a unique name under which it can be retrieved using the *findAttachment* method. This is very useful. For example can all OpenSG objects be named, simply by attaching a character string, a *string attachment*, under the name *"name"*. New attachment types can be defined easily, by instantiating a template.

This mechanism can now be used to store additional XML data directly with each object in the scene. For the greatest compatibility, and to be independent of any particular XML library, the XML code is attached as character string. We have defined the custom attachments

- ***"xml-code"*** to store XML data of any kind, and
- ***"fileref"*** to identify external geometry.

They can be comfortably accessed with *setXMLcode*/ *getXMLcode* and *setFileRef*/*getFileRef*, as shown in Figure 5.4.

5.1.1.14 The XML manager class

XML in a character string is difficult to access. Greater convenience offers the XML manager class with its methods *getXMLtree*/*setXMLtree* to manipulate XML attachments as *TiXmlElement* tree using *TinyXML*. This is a minimalistic XML "library" which consists of just a few cpp files. It offers a very straightforward API to set up and manipulate an XML document tree,

```
// Receive the XML subtree in form of a TinyXML element
// from the container object (e.g. a node):
TiXmlElement* getXMLtree( AttachmentContainerPtr container );

// Set (and overwrite) the XML subtree in the container object
// with xmlCodeElem, a TinyXML element object:
void setXMLtree( AttachmentContainerPtr container,
                 const TiXmlElement& xmlCodeElem );

// Clear the entry path list:
void clearPath( void );

// Append one element to the entry path list,
// optionally require one attribute to have a particular value:
void addPath( const string& pathElem,
              const string& attrib1="", const string& val1="" );
```

Fig. 5.5 The most important functions of the XML manager.

as illustrated in Figure 5.7. The string attachments are converted to a DOM-like tree structure only *on demand*. This solution combines maximal robustness with the greatest ease of use: The XML manipulation in the scene graph is completely decoupled from whatever XML library may be used in some import/export module.

We found that applications often repeatedly need to access the same elements in the XML attachments of 3D objects. The attachments often have a very similar structure, and the code for traversals is repetitive and tedious to write due to the security queries. The XML manager therefore allows to pre-define a path through the XML tree, by node and attribute matching, to provide an easy and robust direct access to a specific element in deeply nested XML attachments. – The API of the XML manager is shown in Figure 5.5.

5.1.1.15 The Fileref Manager Class

It is vital to keep track of all filenames and URLs for two purposes: For a subsequent scene export, and not to import any resources multiple times. This applies also to "node cores" (e.g., geometry) imported through several levels of indirection: A Collada scene might import a Collada "object" scene that imports an OBJ file importing a material file (MTL) loading a jpeg-texture. Even if the texture (e.g., a logo) is used many times it must be loaded only once. The *fileref manager* takes care of all file related activities. For the above reasons the existing importers (OBJ etc.) were also reorganized to use it. This has the added value of a transparent access to remote resources since URLs are resolved. Files are loaded asynchronously to prevent the application from blocking. A placeholder (a sphere) is temporarily inserted into the scene graph until the loader thread terminates. Every scene graph node can be exported. The exporter works in either of two modes: It exports references, stopping at nodes with a fileref attachment, i.e., that were imported, or it exports everything, descending down to the leaves of the scene (sub-)graph. In the latter mode ("Save as") the external binary files are simply copied, which implies that they may contain only relative references to other files.

5.1.1.16 W3C standards in the light of 3D

Information technology in Cultural Heritage speaks XML. A seamless and efficient adoption of XML is therefore the strategic key to making digital 3D artifacts an integral part of cultural

```
NodePtr loadFile ( char* filename ); // Returns a node with
                                     // a placeholder geometry.

bool isLoaded ( NodePtr node );      // Returns true, if the referenced
                                     // file for node is already loaded.

bool isLoaded ( char* filename );    // Returns true, if the referenced
                                     // file with filename is already loaded.

void abortLoading ( void );          // The second thread is stopped im-
                                     // mediately, loading is aborted.
```

Fig. 5.6 Fileref Manager API: small and beautiful

```
...

// request the geometry
NodePtr scene = OSGfilerefManager::the().loadFile( "torus.obj" );

// set some xml annotations
TiXmlElement elem_extra( "extra" );
TiXmlElement elem_technique( "technique" );

elem_technique.SetAttribute( "Profile","epoch:meta" );
TiXmlElement elem_creator( "dc:creator" );

TiXmlText textName( "John Doe" );

elem_creator.InsertEndChild( textName );
elem_technique.InsertEndChild( elem_creator );
elem_extra.InsertEndChild( elem_technique );

// link the tree to the node
OSGxmlManager::the().setXMLtree(scene, elem_extra);

...
```

Fig. 5.7 A small example for using the fileref manager and the xml manager.

heritage libraries and databases. The homepage of the W3C lists more than two dozen XML related technologies. They can open a wealth of new possibilities because many of them are directly applicable to 3D in Cultural Heritage.

XInclude: Transparent inclusion of remotely stored subscenes. In case an object is unreachable a locally stored low-res approximation is automatically used.

XPointer / XPath: Flexible referencing of sub-parts of a 3D object. Very general framework for references, e.g., for spatial indexing, for indexing nodes and sub-scenes. Transparent access to objects stored in the file system, in a database, or on a web server.

XSLT: Personalized rendering, context-dependent views, can deal with different versions of objects, can be used to provide information in different languages.

XLink: Multi-directional links between several documents, e.g., to relate artifact and interpretation in a bi-directional way. Stored externally, so works also for read-only archives. Link on link, to refer to the fact that a link exists.

Web Services: All sorts of operations on XML: filter, process, create views. Actions to perform can be encoded in a URL. Can generate XML scenes dynamically, as response to a web query. Billing, cookies, user rights management.

XML Encryption To protect intellectual property rights: Signing of documents, certificates of authenticity.

5.1.2 3D Powerpoint

Cultural objects in museum exhibitions are sometimes not easy to appreciate. They are too small, very detailed, behind glass, and too precious and delicate to let arbitrary visitors take them in their hands. It is particularly difficult to let visitors from the general public study the amazing, intricate detail and the traces a long history has left on cultural artifacts.

3D technology can help tremendously to enhance the appreciation of museum objects. Our guiding vision is the idea of a *complementary exhibition*: Real artifacts in a museum exhibition are complemented by digital artifacts whose sole purpose is to deepen the understanding and appreciation of the real ones. A particular form of a digital artifact is the *digital replica*. To show it in a museum combines the authenticity of the real with the ease of manipulation of the digital. As a result, museum visitors become more engaged since they can actively participate. This approach is also quite successfully used in science and technology museums, which have greatly increased over the last years. School children are encouraged to actively acquire knowledge by trying out scientific experiments. – Cultural museums are unfortunately still lacking behind in exploiting this *edutainment* aspect.

It is important to emphasize that we do not recommend the use of technology under all circumstances. We advocate instead designing *modest systems*, where never the technology is in focus, but always the content. This implies, for instance, that technological gadgets must be extremely easy to use. They shall not require manuals or instructions, they must work robustly and, most importantly, they must behave in a predictable way. No bad surprises, no frustration, because that takes away visitor's attention from the artifact.

5.1.2.1 Strategic Vision

It is essential for EPOCH that all the beautiful ideas and approaches as, e.g., shown on the VAST series of conference, find their way to the public audience through museums and exhibitions. In terms of interactive 3D presentations, much more is possible than what can be found today in average museums. We want to change this situation and make using interactive 3D a standard. The key features of our solution are therefore usability and affordability. The technical agenda of the EPOCH project is based on the idea of a cultural heritage pipeline, a complete workflow that ranges from the archeologist that finds an artifact in the field to the presentation of the artifact in a museum. We think we should start to create a demand for content at the end of the pipeline in order to stimulate the development of a cultural heritage market based on EPOCH technology on the whole pipeline.

5.1.2.2 A Concrete Target Scenario

A museum curator decides to make a new exhibition on the Etruscian period. He hires a company that digitizes the tiny golden brooches and small pieces of precious jewelry he is going to present. From a recent excavation campaign he purchases the scanned 3D datasets of different strata of the archeological site where the beautiful historic artifacts were found, as well as laser scans of the remains of houses, pillars, statues, etc. He puts all the digital artifacts into the 3D

presentation software, chooses a theme (skin) that fits with the look of the physical exhibition, and uploads the new presentation to the 3D multimedia kiosks via network.

In the exhibition, visitors can see all the real artifacts, as usual behind glass. But in every exhibition hall there is also a 3D multimedia terminal where the visitor can inspect the small beautiful artifacts interactively from all sides, which is not possible with the real. The visitor can also have a quick look at the archeological site where the artifact was found, which also brings up a few statues and some interesting architectural ornaments. The latter are shown in the form of high-resolution 3D scans.

5.1.2.3 Insight

The contribution of this section is to present one part of the target scenario as work in progress: The software infrastructure that allows to design complementary exhibitions, and to present them to the public. Note that it has some features in common with Microsoft Powerpoint:

- It has an authoring and a presentation mode.
- It is based on customizable presentation templates.
- It is made for non-programmers focusing on content.

Since our's is a 3D tool we chose the name *3D-Powerpoint* as the working title for the project. Before starting the project we have made some research on tools to use. None of them fulfilled our requirements. This lead us to a refined list of requirements, our feature wish list in Section 5.1.2.5.

5.1.2.4 Related Work

The presentation of 3D objects to a public audience is often considered a solved problem since a number of possible approaches exist: 3D standard tools, professional 3D presentation software, game engines, scene graph engines, and certain academic approaches.

Usually the first idea would be to use 3D standard tools such as a VRML/X3D viewer, 3D embedded in PDF (Acrobat3D), Shockwave/Flash-3D, etc. Also for 3D modeling tools such as Maya, 3DStudio, Blender presentation plugins exist. But these "closed" solutions can be immediately ruled out since we target location-based presentations with high-quality cultural heritage content, and smooth interaction with non-standard input devices. We could certainly program extensions to, say, a standard X3D viewer, but then we would be tied to this particular piece of software over which we have no control.

A much better option would be to use professional 3D presentation software such as Virtools, Quest3D, Shark3D, OfficeFX, and others. These tools provide professional rendering quality, support a wide range of input formats, hardware setups, and all possible input devices, and they have impressive feature lists, ranging from physics engines over audio to networking. However, an in-depth evaluation some years ago of a similar tool, Realimation, revealed some fundamental problems with such packages. They are

- **monolithic:**
  Not a component, but a complete stand-alone application
- **proprietary:**
  Vital features may change from one version to the next
- **not low-level extensible:**
  They impose strict limits on what developers can access

- **not a modeler:**
  Every non-trivial piece of geometry must be imported

It must be noted, however, that these tools provide a remarkable degree of usability: their authoring environments are extremely interesting, also from an academic point of view.

For most of the professional game engines – the 3D engines database on `http://www.devmaster.net/engines` currently lists approximately 345 of them – basically the same considerations apply: Game engines such as Torque, 3DGameStudio, Ogre, or Irrlicht are optimized for efficiency and use the latest graphics hardware effects. This matches also the expectations of museum visitors, as more and more people are becoming acquainted with game technology. The downside, however, is that content creation for games requires much low-level realtime know-how, much programming, has a low long-time sustainability, and for serious non-game applications requires extra effort to get rid of the game overhead.

A much more acceptable alternative is to use a "neutral" scene graph engine such as Coin/OpenInventor, OpenScenegraph, OpenSG and the like. They combine efficiency and support of (almost) latest effects with openness and extensibility. In fact, they are usually *designed for extensibility*. This is a huge advantage if, like in our case, we have to custom-tailor a 3D solution to the demands and existing standards of a particular application area, in our case cultural heritage.

There are only two academic approaches for 3D presentation environment for cultural heritage we have found. First, Costagliola et al. [CMFP02] publish a configurable presentation environment particularly for guided tours. The other work is the *Virtual Inspector* tool from our colleagues from Pisa, Italy [CPCS08, CPCS06], which focuses on the display of huge meshes.

However, through this research we have come to the following catalog of criteria for our envisaged solution.

5.1.2.5 Feature Wish List

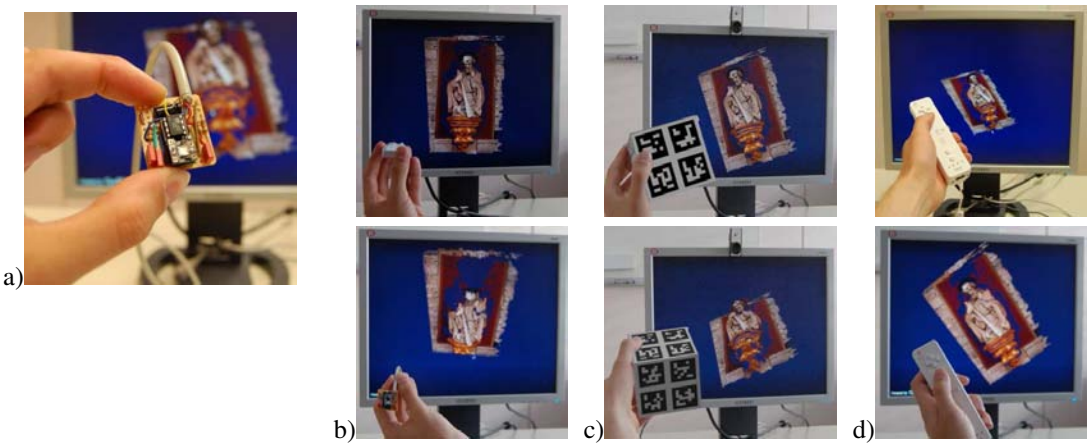


Fig. 5.8 Tangible input devices: (a,b) Accelerometer, can measure 2 of the 6 DOF of a static pose, as it can detect in which direction gravity points. (c) Array of 2×2 ARToolkit markers, to make the camera-based detection more robust against occlusion by fingers. (d) Nintendo Wii controller, with accelerometer and integrated low-resolution camera, can in principle determine a 6-DOF pose plus 2D acceleration.

Drag-and-Drop Authoring:

Every presentation consists of *layout* and *content*. Both is easier to generate and arrange in 2D than in 3D. As we want non-3D-experts to design digital exhibitions, the best solution would be to have *pre-defined layouts* that can be *filled with content* from the local file system or from the Internet via drag-and-drop.

Easy 3D navigation:

Once the presentation is defined it shall run in a public museum. Average museum visitors shall be able to use the 3D kiosk systems without having to read instructions. This is demanding in particular for the notorious problem of 3D navigation: Users shall never get *lost in 3D*, not reach impossible view points or see nothing on the screen, nor get locked somewhere. We want to allow as much 3D control as possible and restrict it only as much as is necessary to enforce consistency.

Cultural Heritage Integration:

3D presentation is only the very end of a long processing chain. Interoperability requires standards. The presentation environment should permit to make use of any additional information attached to cultural objects present, e.g., in the Collada 3D format used in EPOCH. In the long run, even using CIDOC/CRM should be an option, as pointed out by Havemann et al. [HSKF06]: Every cultural artifact is part of a network of semantic information. The ultimate form of a *Cultural Presentation Browser* would be a tool that allows average museum visitors to navigate through this semantic network.

Cultural Heritage Sustainability:

3D presentations will be a new form of collecting knowledge about history and culture. Ideally, our knowledge about cultural heritage should be as long-lasting as the artifacts we show. This issue actually causes fierce reservations against the use of digital technology in the cultural heritage community. However, there is a tradeoff between using the latest technology and being independent from particular software and hardware versions. The solution we envisage is to use advanced, but *well-documented* file formats and algorithms. This way presentations can use state of the art hardware shaders and mesh compression, but are not deemed to be obsolete in five years.

Low-Level extensible:

Today the usual form of representing scanned artifacts is as a textured triangle mesh. A serious 3D infrastructure, however, requires a more diverse set of shape representations. Scanned cuneiform tablets, for instance, usually have a multitude of view-dependent textures or under different lighting conditions, whereas computer tomography produces a volumetric “image” of what is inside, e.g., an Egyptian mummy [BSly]. This requires that new shape representations can be integrated with the viewer, e.g., loaders for new file formats, and new 3D rendering modules.

Template extensible:

Whereas the main use of the 3D kiosk is to let a visitor explore one particular artifact, there is a wide range of possible presentation scenarios. Users might pick one among many artifacts from a shelf, or from a digital replica of the museum room, or even from a historic scene digitally rebuilt in order to contextualize the cultural artifacts by showing them in their historic surroundings. This flexibility shall become possible through *customizable presentation templates*, very basic 3D scenes with objects that have a reasonable pre-defined behavior and whose appearance (geometry+texture) can be configured via drag-and-drop. It is envisaged that curators can download presentation templates from suitable web pages.

3D modeling of ad-hoc geometry:

Sometimes ad-hoc objects are needed for a presentation. Static objects could be created photogrammetrically from digital photographs using the EPOCH Webservice [VG06]. However, this is not applicable in all cases. To let users generate simple objects, e.g., extruded 2D contours, a very simple 3D modeling tool should be part of the authoring software. This tool is comparable to the vector-based diagram editor included in, e.g., Microsoft Powerpoint. And just like Powerpoint it should allow to animate these diagrams by animating the object parameters. This way a 3D stone wall could vary its position and (x,y,z)-extents very much like a line segment in a 2D drawing can.

Non-monolithic:

From a software point of view the presentation viewer shall behave like a component, rather than like a stand-alone application. The reason is re-usability: Along with the 3D presentation additional textual information might have to be displayed, a HTML page or a PDF document. It shall even be possible to integrate the 3D presentation with another application that has its own GUI, such as a numeric simulation or a database front-end. The consequence is that the 3D presentation viewer shall requires not much more than a 3D window to render into; another consequence is that it does not provide a sophisticated 2D GUI with a menu hierarchy (like MS Powerpoint has). It should be possible, though, to later add a 2D GUI with a menu.

Developer Levels:

We envisage a hierarchy of users of our systems. Each level requires more knowledge and, thus, will reach out to a smaller community:

Level 0: End-user who consume the 3D-presentations
Level 1: Authoring of presentations: 3D-GUI, drag&drop
Level 2: Authoring of presentation templates: Scripting
Level 3: Extension developers: C++ programming

Users on levels 1-3 are creative people, the DCC providers, which stands for *digital content creation*.

Fig. 5.9 Example of a 3D presentation. Left column top to bottom: The user can rotate the chandelier horizontally. Right column top to bottom: One object is chosen for inspection, and the user can orbit around it. – The objects shown are not high-quality artifacts but only examples.

5.1.2.6 A First Version of 3D-Powerpoint

The proposed solution is based on the combination of the OpenSG scene graph system with the GML scripting language [Hav05].

We have developed a series of GML scripts for 3D modeling, for presentation templates, and for particular presentations. The first phase of our work concentrated on providing OpenSG with the functionality needed, and on making it accessible via GML in a suitable fashion. Next we have begun to create a number of example presentations on this basis. The next step, which will start soon, is to revise and refactor the GML code for those presentations. The goal is to distill a set of basic GML components out of these presentations, in order to produce a GML framework that will be useful for all sorts of presentations. The final step will be to create a conventional GUI and menu system, which makes use of the 3D presentation as a component.

5.1.2.7 Input device: The tangible proxy object

By far the most intuitive 3D input device is a 3D object. The idea of the *proxy object* is that the virtual object displayed on the screen moves exactly like the real object that the user holds

in his hands. Ideally, the user can move and rotate the object, and the digital artifact is in exact sync. Note, however, that we want to map the 6-DOF pose directly, not in a mirror fashion, so that when the user stretches out the hand with the proxy object the virtual object also goes farther away. It does not come closer as would be the case with a mirror. – Technically, the problem is to determine the 6-DOF pose of the proxy object. We have experimented with the three technologies shown in Figure 5.8.

First technology: ARToolkit.

We have tried camera-based tracking using the ARToolkit from www.artoolkit.org. With one marker per side of a cube and a single camera we had serious robustness problems: Whenever a finger only *touched* the black boundary of the marker the recognition algorithm of ARToolkit broke. Consequently we have made the cube a bit larger and used an array of 2×2 markers. This seems to be a good compromise between robustness and processing load, as the latter affects the recognition speed and, thus, the frame rate. It is quite unlikely that the user occludes all four markers of one side at the same time, and usually more than one side is visible.

Second technology: Accelerometer

ARToolkit markers have a very technical appearance which we wanted to avoid. The position tracking also created slight problems since users tended to move the 3D object out of the frustum: To inspect the object they took it very closely until the view showed only a detail of the surface; but then, in order to rotate it, they took it from one hand to the other, thereby actually moving it quite a bit to the right or the left. This made the object suddenly disappear, and the users got lost and felt uncomfortable struggling to bring the object back.

Consequently we tried to get along by using only the orientation (acceleration) information. With an object held still, gravity causes the strongest acceleration, so the downwards direction can be robustly detected. Roll information around the gravity vector, though, cannot be detected, so it cannot be decided whether the user points north, east, or west. So we used the accelerometer information only for relative motion (spinning speed). This worked well and robustly.

The accelerometer is a standard electronic device and quite cheap (15 Euro). It can easily be connected to a serial or USB port. Due to its small size it can also be put inside another object, e.g., one that resembles a cultural artifact. This looks much better in a museum than ARToolkit markers.

Third technology: Nintendo Wiimote.

The controller of the Nintendo Wii, the *Wiimote*, communicates with standard Bluetooth. Free software tools exist to decode its protocol, e.g., Kenner's GlovePIE [Ken]. The Wii can deliver also position information, as it contains an optical sensor that, when pointing towards a certain configuration of LEDs, determines the pose relative to the LEDs. The Wiimote is a mature device and quite robust to use, which made it our preferred test device, despite its non-museal appearance.

5.1.2.8 The 3D Presentation

A first example of a 3D presentation is shown in Fig. 5.9. The user sees an object selection menu that is shaped like a chandelier. With a slight rotation of the Wiimote to the left or the right the chandelier begins as well to rotate smoothly, showing the next object in the respective direction. By tilting the Wiimote upwards the close-up view is activated: The chandelier gradually moves away and the chosen object comes close until it fills the view.

One of our goals was that users can always keep track of what is going on. There are no abrupt transitions and we have taken care that all motions are smoothly animated. Before the chandelier moves away, it retracts; when the close-up inspection is finished, the chandelier appears again and unfolds, see Fig. 5.10.

Fig. 5.10 The "chandelier" smoothly retracts before moving away, as a clear signal that close-up inspection of the selected item begins

The close-up object can be inspected in detail: With the two DOFs of the Wiimote (rotate L/R and U/D, for left/right, up/down) it is only possible to orbit around the object center in a fixed distance: In terms of Euler angles, L/R determines the azimuth and U/D the elevation of the object.

We have experimented also with a combined interaction mode: The elevation must be clamped, e.g., to $[-70, 70]$ degrees to avoid the gimbal lock. When the elevation is maximal or minimal, a further increase or decrease makes the object come closer or get farther away, respectively. – Users usually reported a good feeling of control in this combined mode. The problem was only that they found it uncomfortable: It is apparently more convenient, or natural, to first navigate (orbit) to a particular spot on the surface, and then to adjust the distance to this spot. Our mode required to first adjust the distance, and then orbit over the surface.

Note that the objects in Fig. 5.9 are low quality reconstructions, generated photogrammetrically from a single range map and decimated to 10K vertices using Michael Garland's quadric-based simplification tool qslim. A much higher rendering quality can be obtained using the BTF-rendering module for OpenSG from Gero Müller and Reinhard Klein (Univ. Bonn). A BTF provides much more surface detail as it approximates the BRDF with much more than only a single texture value per surface point. Especially small and shiny BTF surface parts are brought out by the headlight much more clearly. Fig. 5.11 can only deliver part of the experience to hold a shimmering object virtually in his own hands.

Fig. 5.11 Authoring a presentation with BTF-models. They are loaded and rendered using an OpenSG extension module that makes use of advanced vertex and pixel shaders. Models are courtesy Gero Müller and Reinhard Klein, Univ. Bonn, Germany

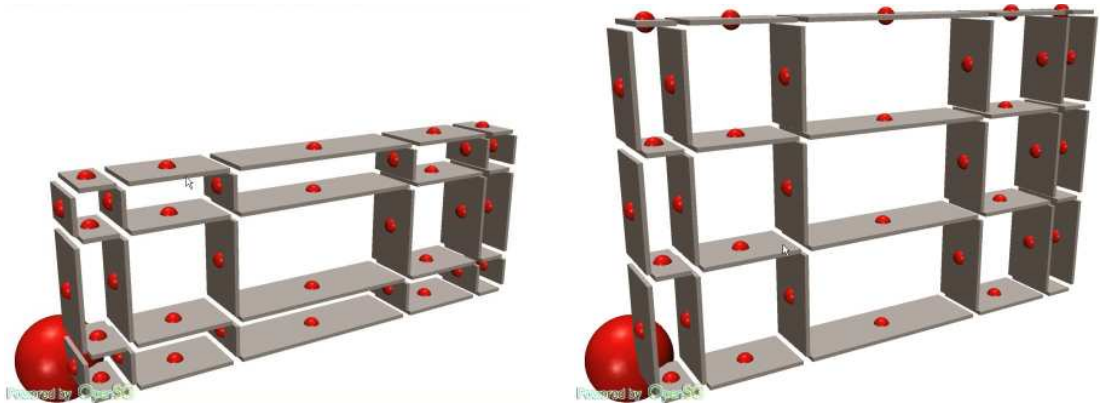


Fig. 5.12 Presentation layout: Parametric shelf. All the boxes can be dragged to adjust the spacing, and pushing the red spheres inserts a new row or column of planks.

5.1.2.9 Authoring of a 3D Presentation

We have experimented with several sorts of layouts. The first idea was a rack or shelf (Fig. 5.12) where the spacing can be interactively adjusted to match the sizes of the artifacts. Fig. 5.13 shows our chandelier-like design. Its special feature is that it rotates non-linearly in order to clearly highlight the object that can be chosen for the detailed inspection.

Fig. 5.13 Presentation layout: Chandelier. Even when densely populated, the selected object sticks out clearly due to the uneven circular spacing.

Fig. 5.14 Authoring via drag-and-drop. a) A blank layout consists of many drop targets. b) The "model bar" is filled via drag-and-drop from the file system with 3D models, 2D images, and 1D text strings. c) Objects are dragged interactively from the model bar to drop targets in the 3D scene, where they automatically align to the local coordinate frame.

5.1.2.10 Authoring via Drag & Drop

The main idea is that our layouts are almost completely composed of so-called *drop targets*. Figure 5.14a shows such a "blank" layout. All the boxes and spheres are successively replaced. Three different types of objects are supported: **3D models** (Collada .dae, Wavefront .obj, Stanford .ply, OpenSG .osb, etc), **images** (.png or .jpg, immediately applied to texture a quadrangle), and **character strings**, which are rendered as true 3D text. The replacement proceeds in two steps:

- **Filling the model bar:** The user drags an object from the file system (the Windows Explorer) to the 3D window where it appears in a row along the lower border of the 3D window, the *model bar* (Figure 5.14b)
- **Replacing the drop targets:** Individual models can be dragged interactively from the model bar to drop targets in the 3D scene (Figure 5.14c). Whenever dragging the object over a suitable drop target the object temporarily snaps and aligns with this target. The user can decide whether to leave it there (mouse release), or continue to drag it elsewhere.

Note that objects cannot be dragged immediately from the file system to a drop target in the scene; the model bar always acts as an intermediate storage. The reason is that with larger models there is a noticeable delay until the object is loaded and appears in 3D under the mouse pointer. Users would instinctively think that the dragging operation has failed, and stop

dragging – only to see that after a short while the object appears *somewhere* in the scene. This was perceived so frustrating that we decided to introduce the model bar.

Another thing that has proven successful was that when loading an object we immediately show a temporary geometry, a sphere that is replaced by the true object as soon as its loading is finished. We do not, however, use the sphere for immediate drag-and-drop because of size issues: The temporary sphere cannot reflect the true size of the object that is being loaded, simply because the bounding box of this object is only available after it has been loaded.

We plan to solve this issue using the so-called *Collada light* 3D format: The (lightweight) Collada XML file contains only semantic and metadata information, in particular the bounding box, and it references another (heavy) binary file that contains the 3D data, for instance a huge U3D file with compressed triangle data.

5.1.2.11 Modeling Included: Creating Ad-Hoc Geometry

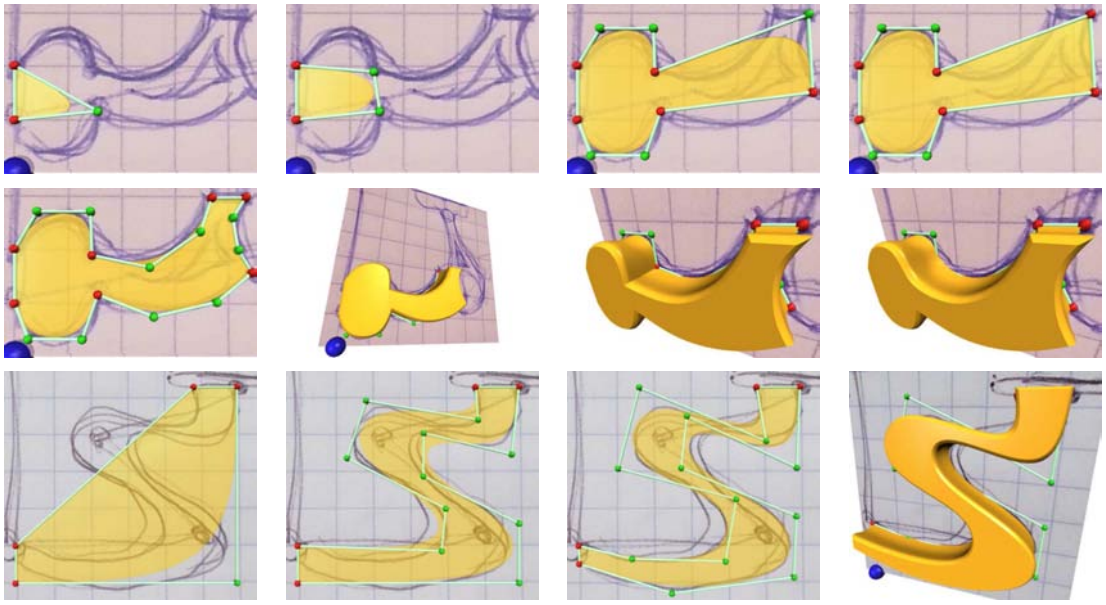


Fig. 5.15 Creating ad-hoc geometry from a sketch. Row 1: A rubber band triangle is successively expanded. Row 2: The profile is extruded, polygon vertices can be sharp (B-spline) or smooth (corner). Row 3: Most users understand quickly how to place vertices in a way such that the resulting raw shape can be adjusted efficiently.

The design of the arms of the chandelier was based on a small sketch on a piece of paper that, in the museum scenario, would have come from a curator or from a graphics designer (see Fig. 5.16). The sketch was photographed, the photograph was perspectively corrected by marking four points and then it was loaded into the GML based modeler. The modeling proceeds in a very simple rubber-band fashion using a control polygon (Fig. 5.15): Clicking on the polygon boundary inserts a new ball. Balls can be freely dragged around on the construction plane. Just clicking on a ball toggles its red/green status: green balls control a B-Spline, red balls are corners. Clicking on the blue ball extrudes the polygon. The extrusion profile can also be adjusted, but this is not shown here.

The ad-hoc geometry thus created is then dragged into the authoring toolkit.

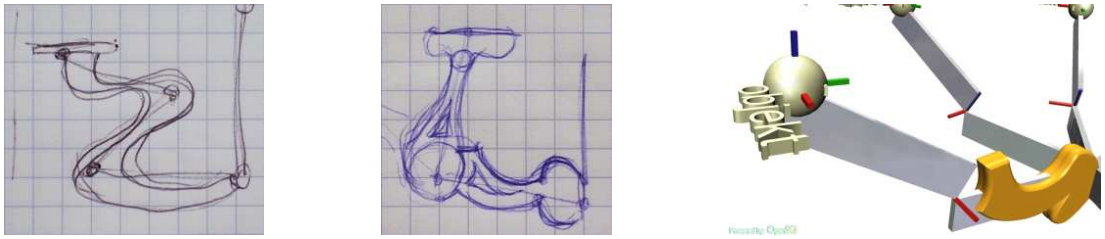


Fig. 5.16 Two designs for the arm of the chandelier. The ad-hoc geometry created from the sketches is being dragged onto the drop target. As soon as the mouse button is released it is copied on all instances of the arm.

5.1.2.12 Insight

The 3D-Powerpoint toolkit presented in this section leaves much room for improvement; in fact, it is work in progress. Although our results are far from perfect we claim that the foundation is sound with respect to the feature wish list from Section 5.1.2.5, with which we would also highlight some areas for further research:

- **Drag-and-Drop Authoring:** The drop targets work extremely well in the authoring mode. Future layouts will have also drop targets for images (backdrop) and text. The model bar should be part of a 2D GUI, though.
- **Easy 3D navigation:** The tradeoff between comfort and control must be improved further by employing more intelligent camera behaviors. 3D navigation in the authoring mode must be improved as well.
- **Cultural Heritage Integration:** As soon as Collada files offer more standard (meta-)information, this information should be available for 3D presentations. Examples: The textual caption of the 3D model, and information links embedded in the surface.
- **Cultural Heritage Sustainability:** Clearer separation between framework and individual layout, so that the layout style of a presentation can be exchanged like in MS Powerpoint.
- **Low-Level extensible:** This is accomplished.
- **Template extensible:** This is accomplished as well.
- **3D modeling of ad-hoc geometry:** It is unclear how sophisticated the 3D modeling should be: Should it be like Google Sketchup or significantly less powerful? – Very interesting though would be the use of animated parametrized models as 3D-illustrations.
- **Non-monolithic:** This is definitely accomplished. The presentation viewer can in fact be embedded as a component into any application providing an OpenGL window. All program functions can be accessed through GML scripts that can even be synthesized at runtime.
- **Developer Levels:** A proper documentation of the scripting facility is the only thing that is required for others to develop interesting presentation templates.

Cultural Heritage is inherently 3-dimensional. On the one hand the remains from our past form a VAST collection of 3D objects. On the other hand, every item of cultural value is embedded in time-varying social, political, geographical, traditional, and personal contexts. Each artifact has its own history, but also belongs to an object class, which develops over time as well. In fact, each cultural artifact belongs to many classes: Shape, material, manufacturing technology, possessor, all provide equally valid views on an object.

This shows that when attempting to embed 3D objects into their semantic context, things quickly become extremely complicated. As a consequence, Cultural Heritage is a very rich and interesting, but also very demanding field from the information modeling point of view.

5.1.3 The Arrigo Showcase

Fig. 5.17 The Arrigo 3D presentation from ISTI-CNR, Pisa. Beautifully made, but it is a closed solution. All texts shown are just bitmap images that are inconvenient to change.

The motivation for the work described in this section comes from a beautiful museum exhibit, the *Arrigo Showcase*.

The *Arrigo Showcase*, a beautiful museum exhibit, was created by Paolo Cignoni and Roberto Scopigno of the Visual Computing group of CNR Pisa, Italy. It represents one of the rare cases where *interactive* 3D technology was successfully used in a museum exhibition. Using a mouse, visitors could interactively explore detailed 3D models of all 12 statues of the Arrigo ensemble. The *VirtualInspector* technology developed in Pisa [[CPCS08, CPCS06]] maintains interactivity even with close-up views of massive 3D datasets with millions of vertices. It was deployed for the public display of the *David* statue scanned in the *Digital Michelangelo* project by Marc Levoy from Stanford. The display was located right next to the real statue, which is 5 meters tall. Visitors could discover details in 3D that were barely visible on the real statue. This way the 3D visualization does not replace, but enhances the appreciation and understanding of the real artifact.

The interesting part of the **Arrigo Showcase** is that it encourages exploring not only the 3D dataset, but also its semantic embedding. Some parts of the statues, like the mutilated hands or the hollow back, lead to obvious questions. With information icons on the 3D model visitors could access an information pane with a short textual explanation (Figure 5.17). In that way semantic information is made accessible via 3D.

5.1.3.1 Drawbacks of the Arrigo Approach

In 3D software there is always a tradeoff between **fine-tuning** and **generality**. Optimal results are possible only by adjusting all components: input devices, display hardware, rendering algorithms, graphical user interface, content quality, graphic design, interaction metaphors, and look and feel.

As beautiful as it may be, the Arrigo Showcase suffers from fundamental limitations with respect to the following requirements:

- **Sustainability**: Computer hardware can become outdated, as well as display algorithms and file formats. But cultural objects keep their semantics and should be available independently of the technical development status.
- **Extensibility**: Adding new information icons, including additional views on the subject, alternative interpretations. Linking into the showcase, and links from the showcase to information outside. Adding such links much later.

- **Changeability**: To replace a system component is possible only with clear, well documented specifications of the interfaces between the modules. Modules should comply to standards rather than to ad-hoc design decisions.
- **Versatility**: Ideally, data and systems cover many application scenarios, e.g., using the same data in a museum exhibition and for scientific research. Data should be Internet accessible, and should be reliably citeable in a scientific publication or in an e-mail.
- **Openness**: Every piece of information leads to other pieces; information can be hierarchically grouped or arranged in a graph (*hypertext*); each of the information icons is in fact part of a larger *semantic network*.

5.1.3.2 Requirements Analysis through Use Cases

It is not an easy task to overcome the limitations of the Arrigo system. An analysis was carried out to identify a number of use cases that a more general system would have to cover.

- **Scenario: Web browser with coupled 3D browser**
  Following a hyperlink that points to a 3D object should bring up a 3D browser displaying this object. 3D objects may contain embedded hyperlinks (information marks) leading to other 3D objects, or to web documents. Such software shall be easy to use and ideally be free.
- **Scenario: Museum Presentation**
  The 3D browser and a web browser are seamlessly coupled together in a fullscreen application. The 3D layout and the web page are beautifully tuned to match. The web browser can show multimedia content (images, video, Flash) triggered by links on the 3D model. The 3D view (camera, shading) can be modified using hyperlinks.
- **Scenario: Scholarly Research**
  The 3D browser and the web browser are connected to a semantic data base via a combined frontend. It gives access to a semantic network of entities (generalized "multimedia" documents). Relations between entities can be browsed, and new relations can be established.
- **Scenario: Authoring semantic networks**
  There must be an easy way to create 3D attachments. An authoring application allows importing raw datasets, objects can be spatially arranged via drag-and-drop, and relations between 3D objects can be defined. Ideally, scenes can also be provided with behaviors (*e.g. drag-and-drop behavior*).
- **Scenario: Development of 3D-Applications for CH**
  CH requires a plethora of specialized applications to cover all aspects from excavation management over GIS databases to artifact collections and museum archival software. They would all greatly benefit from functionality like in the Arrigo showcase.

5.1.3.3 Information Model: Design Decisions

The first step is to define the information infrastructure. It must be simple, reliable, versatile, extensible, and standard compliant. Our basic decisions and definitions are:

- **Integration into semantic networks**
  Cultural information is modeled today as a network of entities connected by semantic relations. 3D should be part of them.

OpenSG/XML/GML	OpenSG engine with GML scripting, maintaining XML attachments to 3D scene graph nodes
ActiveEpoch	*OpenSG/XML/GML* as ActiveX control that consists only of an OpenGL window
EpochViewer	Thin layer to wrap ActiveEpoch into an application that resolves links to 3D object parts
EBHO	Internet browser extension, redirect .dae to *EpochViewer*, including relative links (#Head)
AuthoringTool	Markup application combining *ActiveEpoch* control with an *InternetExplorer* control
PresentationTool	Integrated 3D/web browser with simplified interaction for museum exhibitions

Fig. 5.18 The components of our software infrastructure. The ActiveEpoch control encapsulates the 3D functionality.

- **XML for structure and semantics**
  These semantic networks are encoded in XML. Many XML technologies exist to manipulate XML encoded knowledge.
- **All references are URLs**
  An entity is nothing but a *uniform resource locator*, a URL. A relationship (e.g., a RDF-triplet) links two entities (two URLs) together by a predicate (another URL).
- **Bi-directional linking HTML $\leftrightarrow$ 3D**
  A 3D object is uniquely identified by a URL pointing to a model file with the appropriate MIME type (.dae):
  http://www.CH-models.org/statues/arrigo5.dae
- **3D-Links on sub-object level (part level)**
  A part of a 3D object is uniquely identified by a relative link, similar to linking to an anchor in an HTML document:
  http://www.CH-models.org/statues/arrigo5.dae#Head
- **3D annotations in XML: Collada**
  .dae-links refer to Collada XML documents which have a *library part* and a *scene part*. 3D objects in the library may have information attached. Our attachments consist of a *region in space* (link anchor) and a *URL (link target)*.
- **3D objects in binary formats**
  The actual 3D datasets are not encoded in XML but in native 3D formats (obj, ply etc). The 3D objects in the library part of a Collada file only refer to them via URL.
- **3D scenes in XML: Collada**
  The scene part of a Collada file defines the spatial arrangement of the objects defined in the scene part. It consists of hierarchical transformations, a *scene graph*.
- **Multiple interpretations simultaneously**
  Different .dae-Files can provide the same 3D model with different sets of annotations (interpretations). One model part can have multiple hyperlinks attached, different views or aspects of that part.
- **Mechanisms and behavior are possible**
  The 3D scripting language GML has an XML-based encoding (.xgml). GML code can in fact be inserted directly into Collada to provide objects or whole sub-scenes with behavior.

The next step is identifying a set of functional units that can be combined in different ways to realize the various application scenarios. The resulting software modules are shown in Figure 5.18, they will be further explained shortly.

This section shall demonstrate how this infrastructure works by re-doing the Arrigo Showcase. The purpose of this exercise was to assess the usefulness and the effectiveness of our tools. Furthermore, we consider it one step towards providing best-practice examples of sustainable information modeling in Cultural Heritage – which is our main goal.

5.1.3.4 Related Work

The conceptual basis for our work are *The London Charter* (TLC)[Lon06] and CIDOC-CRM [CDG*05]. The TLC introduces the notion of *intellectual transparency*, which reflects the necessity of maintaining the distinction between measurement (wall with measured height) and interpretation (wall inferred from foundation wall). As a consequence, it prescribes to collect *paradata*, provenance and processing history, throughout the whole work flow.

The *Conceptual Reference Model* CIDOC-CRM from the "International Committee for Documentation of the International Council of Museums" (ICOM-CIDOC) is the standard for relational semantic networks in Cultural Heritage. It provides 84 *entity classes* (actor, place, time-span, man-made object etc). It also defines 141 relations, e.g. participated in, performed, at some time within, took place at, is referred to by and has created.

Niccolucci et al. have demonstrated the integration of 3D into semantic databases based on the X3D format using the MAD/SAD framework [ND06]. Our current work aims at an extension of their ideas, but is based on Collada instead of X3D. Collada is an exchange standard from the *digital content creation* (DCC) industry originally defined by SONY to streamline content exchange for the Playstation [AB06]. It is now an open standard hosted by the Khronos group.

Our work can also be seen in the context of the *Seven open problems in 3D documents* [HF07], where maintaining a consistent relation between a shape and its meaning was pointed out as one major problem that is unsolved with current 3D technology. One step in that direction was the integration of XML into the scene graph [HSKF06] and the availability of a sustainable presentation framework for museums (aka *3D-Powerpoint*) [HSLF07]. The current work can be understood as a generalization of these approaches.

Concerning the processing of textual data into semantically enriched representations we have seen some projects connecting the Text Encoding Initiative (TEI) to other information resources using conceptual modeling. The Henry III Fine Rolls Project used RDF/OWL [CSVP07] and New Zealand Digital Library used Topic Maps [Tuo06]. And both used CIDOC-CRM in their ontology building. But the integration has been to objects such as historical persons and places and not to 3D representation, and the linking has been on the level of single objects. The distinction between an object and a part of an object (a statue and the arm of the statue) has not been taken into consideration.

After the production of a 3D dataset, Collada files for annotation can be generated with the AuthoringTool. The software is capable of retrieving 3D objects from a local file system or remotely from a URL. This also works for follow up files like material definitions and textures which may not be included in the model file.

The 3D data is embedded into a scene to add the necessary additional data. The conceptual components of the scene are:

- **Hierarchical structure**: The scene graph structure is used to arrange one or multiple 3D models.

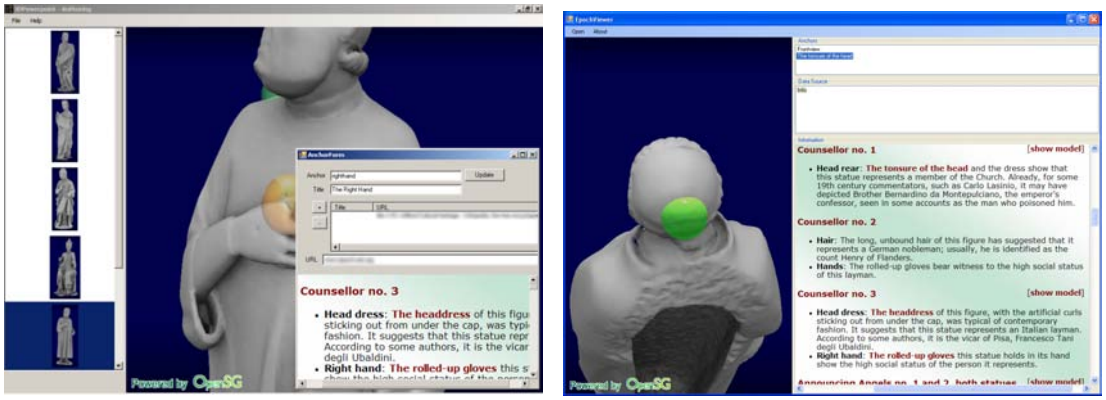


Fig. 5.19 The AuthoringTool (left) and one of the presentation tools with integrated web browser (right).

- **Reference to 3D data**: The 3D data, potentially massive datasets, is referenced using a URL.
- **Transformation**: Spatial location and orientation of the imported 3D model or sub-scene.
- **Annotation data**: All of the necessary data to define a markup is directly attached to each 3D model.

In the first step of the authoring process the model can be arranged properly. It may be necessary to re-align the data to match a desired up-axis for consistency, or to align with other objects. Also scaling to the original size may be necessary. Depending on the used shape acquisition method the original data may or may not be in a unified metric coordinate system. All transformations are stored only in the scene file, the original 3D data are not modified.

The scene is written to a "lightweight" version of the Collada format (just scene, no shape data) so that referenced shape data remain in a binary format. This way, optimized binary formats beyond the Collada standard can be supported. All those formats have to be well defined and documented for long term archival. This was achieved by using the open source scene graph engine OpenSG as a unified basis for the whole process of 3D acquisition. The supplemental data containing the markup definition and associated URLs is stored in a so called extra element provided by the Collada scheme. Extras containing custom data can be attached to every scene graph object. Note that Collada compliant applications are not allowed to discard unknown extra data.

Having set up the Collada scene for the 3D model it is now possible to add markup areas with the authoring software. One of the easiest ways to define an area in 3D is using a sphere. AuthoringTool allows creating spheres by clicking on the model's surface and dragging the mouse to modify the radius of the sphere. Other primitives like boxes may also be considered as markup geometry. Additional geometry definitions for annotation areas can be added in the future with little expense.

The user will most likely navigate to a good point of view to define the area. This behavior helps to store a suitable camera definition for the markup area. There are some algorithms available to find such view points automatically, but in general it is best to let the knowledged user choose it manually in the authoring process. The camera is described quite similar to the common Collada specification. The annotation geometry and the camera are defined in the object space of the model. Independent from the transformation of the scene it is possible later on to reuse the annotation definition for other scenes.

Now the user has to set an identifier for a later reference of the annotation. This identifier is similar to an anchor point of a web page. It has to be unique within the scene. One or more URLs can then be attached to an anchor leading to the additional data. This step depends a lot on the purpose of the scene. For a museum application the URLs lead to appealing presentations of details on the model. For scientific markup, a data base entry with collected facts may be more appropriate. The authoring software supports the user in finding the associated data with the integrated web browser. Drag and Drop functionality helps to copy URLs easily from one place to another.

The information is stored directly with the scene graph object as XML code. Later modifications of the scene will not destroy the defined markup and the connection to the model. An example of the extra element is listed in Figure 5.20.

```
...
<annotation id="head" title="Head of Arrigo">
  <annotation_geometry>
    <annotation_sphere>
      <center>13.1 31.96 2.32</center>
      <radius>3.73</radius>
    </annotation_sphere>
  </annotation_geometry>
  <annotation_camera name="eyepoint">
    ... similar to Collada
  </annotation_camera>
  <annotation_url title="Some Text">
   http://www.txt.com/statue/
  </annotation_url>
  <annotation_url title="A Photo">
   http://photos.org/statue/
  </annotation_url>
</annotation
...
```

Fig. 5.20 Example ⟨annotation⟩ within the Collada scene "Statue07.dae": The annotation "head" with one sphere as the annotation's markup geometry, a camera definition and links to two external information sources.

5.2 Security

5.2.1 Project Autovista

Project leader: Institute for Computer Graphics and Vision (ICG), project partners: Institute of Computer Graphics and Knowledge Visualization (CGV), Austrian Institute of Technology GmbH, Mobility Department, Transportation Infrastructure Technologies and Siemens.
This project is funded by FFG under FIT-IT Visual Computing.

Autovista stands for Advanced Unsupervised Monitoring and Visualization of Complex Scenarios. For large areas with closed-circuit television (CCTV) surveillance, such as train stations or shopping malls, a large number of cameras has to be used for full coverage. On a typical monitor array it can be hard to follow a person through multiple camera images or to get an overview. In contrast, in the Autovista project an integrated 3D model of the entire site is used. Automatically detected persons are shown as a billboard directly in the 3D scene and the video images are back projected onto the model.

5.2.2 Visualization of Image Detection Results

In the image processing research community the common way of presenting the results of, for example, a new object detection algorithm is to show grayscale images with colored rectangles around the detected objects. This approach permits to compare different algorithms by running them on the same set of benchmark images or videos. Both false positives and false negatives are easy to spot and to compare using this approach.

However, in real-world applications this might not be the best way to visualize the results of detection or tracking algorithms. While researchers naturally focus on the methods and algorithms, users are typically only concerned with the meaning of a detection event. In some cases they are interested in specific events or locations, while at other times a more ambient impression may be sufficient. The visualization must be scalable, from individuals to groups and to flows, as well as from concrete video pixels to rectangles to more abstract motion patterns.

A detailed 3D model overlaid with full video and detection information from multiple camera sources can still be remarkably irritating for a human operator. To realize and manage different modes of information presentation are a challenge not only from a computer graphics point of view, as image synthesis problem, but also a comfortable user interface is a not easy to realize – let alone the problem of intuitive 3D navigation. In principle different options exist to obtain a coherent integrated information space, as outlined in the next section. We propose to use a scriptable scene graph engine that already has much of the required functionality built in. In this section we describe the basic system, as well as the enhancements we needed to develop in order to accommodate video textures and to integrate the image processing information. Finally we demonstrate what needs to be done to realize an example application, a surveillance task of a building complex using eight cameras.

5.2.2.1 Related Work

In principle a whole number of different options exist to realize the vision of an integrated 3D information space as an infrastructure for computer vision researchers to visualize the results of image processing algorithms, such as object detection, object recognition, and object tracking.

The first, most straightforward option is to use a standard low-level 3D API such as OpenGL or DirectX to directly code video billboards, to simply replace the colored 2D rectangles by textured 3D quads. Although absolutely feasible, this approach has little sustainability as soon as more and more standard geometry (modeled buildings, materials, interaction components) are to be integrated to achieve a more realistic result. So this approach is not likely to result in a flexible, sustainable, easy-to-use infrastructure.

The next option is to use a standard game engine. This is absolutely feasible as well. However, the drawback may be that game engines are typically highly optimized to guarantee a fluent game play. An open architecture for easy extensibility is often less of a concern, which impedes the flexibility.

Consequently, the resulting option is to use a scene graph engine. Our solution is based on the open source scene graph system OpenSG in combination with the GML for scripting and geometry creation. Our visualization framework is similar to a presentation tool for cultural heritage proposed by Havemann et al. [HSLF07].

The general problem of visualizing image recognition results in 3D is not a new one, of course. Hall et al. present an integrated visualization of outdoor scenarios and live video images.Although they have a good integration of multiple remote video cameras their system is restricted to flat outdoor scenes.

Impressive approaches that are very similar to our surveillance application were presented by Neumann et al. [NYH*03] ,Sebe [SHYN03] and Sawhney et al. [SAK*02]. Neumann presents a system that implements projecting images of cameras. Sawhney describes a system called Video Flashlight for projecting videos of static and moving cameras onto a 3D model. It also uses a shadow mechanism on the graphics hardware and is able to use smooth blending between two images that partially overlap. However, these systems are aiming at the specific surveillance setting. The goal is not so much to provide a general infrastructure.

Fleck et al. present a distributed network of smart cameras for real-time tracking [FBBS06]. Similar to our solution, they integrate 2D detection results into a 3D scene as billboards. 3D models of the observed area are acquired as point clouds. Their system allows large scale scenarios with hundreds of cameras. However they only show the detected objects as billboards but do not project the whole video image. In areas of low point density their visualization exhibits disturbing holes.

The implementation of our video texture projection is based on a combination of projective texture mapping [SKvW*92] and depth map shadows [RSC87].

For our example application we tried different techniques for the visualization of buildings. Among others Wang et al. show methods to visualize multiple floors simultaneously, e.g. floors of a building can slide apart so that they do not occlude each other [WKCB07].

Automatic detection and classification of moving objects with multiple surveillance cameras was presented by Collins et al. [CLK*00]. We use the output of the person detector presented by Roth et al. [RB08].

Our proposed visualization system is a combination of a general OpenSG application and a script defining the scenario specific parameters as proposed by Ousterhout in [Ous98]. The input data consists of a set of videos, detection results and a 3D model of the observed area. Figure 5.21 shows a schematic overview of the system. The main application can connect to

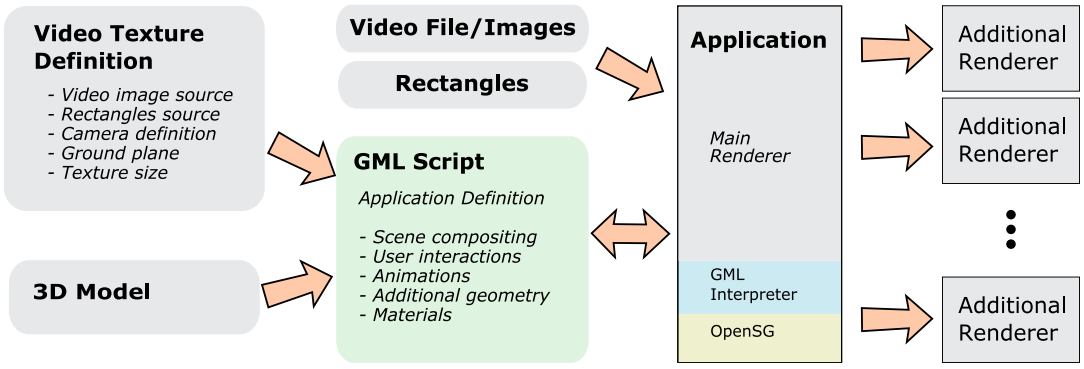


Fig. 5.21 Overview of the visualization system. The Generate Modeling Language (GML) script defines all of the main aspects of the application. It takes care of the scene initialization, animations and user interactions. The definition of the video input data is stored in a separate location for each video texture.

multiple OpenSG render servers to form a visualization cluster. In this way it is possible to realize a tiled display or a DAVE-like immersive screen setup.

5.2.2.2 Video Input Data

The definition of a video source is encapsulated in a simple XML configuration file. Each camera is defined in a separate file. A typical configuration file is shown in Figure 5.22.

```
...
<!   the_camera   >
<camera position = "24.0,32.3,6.1" direction = "0.8,0.3, 0.5"
        up = "0.4573,0.3078,0.8343"
        projectionDistance = "0.86" aspectRatio = "1.333"/>
<!   the_geometry   >
<geometry groundp = "0,0,0.36" groundn = "0,0,1"
          humanheight = "1.85"
          rectangles = "detections/085.log" />
<!   video source   >
<server videofile = "085.avi"
        serverAdress = "localhost" serverPort = "5678"
        serverID = "1" />
<texture width = "512" height = "256" />
```

Fig. 5.22 Example configuration file for a video source. The position of the camera, the ground plane and the human height correspond to the coordinate system of the 3D model.

Video images can be stored in video files or delivered via network from one or multiple custom video servers. A network connection is defined by the server name, a port number and the server ID. Local video files are defined by a file name with a path relative to the configuration file location.

The camera frustum is described by *projectionDistance* and *aspectRatio*. The projection distance refers to the distance of the camera while capturing a one meter wide object filling

Fig. 5.23 A typical area definition of detections in a video frame. In this example, three persons are detected. The rectangles are stored in a simple text file together with a time stamp as a reference to the video frame.

the whole width of the image. Lens distortions are ignored in the current implementation. The position and direction of the camera as well as the up vector are defined as float triples. A ground plane is defined in the same way by position and normal. The plane will be used to place the detections in 3D which is described later on. All of the position definitions refer to the coordinate system of the 3D model.

The resolution of the input video is arbitrary. In the configuration file a different resolution can be defined to deal with hardware restrictions or counteract performance issues. Depending on the graphics hardware it may be necessary to choose a power-of-two width and height.

5.2.2.3 Detection Data

2D detections are defined as regions in the video frame. For now our solution supports image aligned rectangular areas defined by two 2D points. This is the most common definition of regions in an image. In principle our system allows to define an arbitrary number of detections for one video frame.

Similar to the video frames, the rectangle coordinates can be read from a file or they are sent via network. The file format for the detection and tracking results is very simple. It basically is a text file with pixel coordinates of rectangles and a corresponding time stamp, as shown in Figure 5.23. Time stamps are used to connect the rectangles to their video frame. We also support additional data like IDs and certainty values for each rectangle.

5.2.2.4 3D Model Preparation

A 3D model of the scenario is needed to create an integrated visualization of the image detections. Geometric accuracy of the area observed by cameras is essential for a matching video back projection. Instead of creating the model by hand a schematic approach was chosen: The GML was extended with a shape grammar. Using this grammar a set of building blocks was created by measuring the rooms and analyzing the structure of the building. Within a few weeks the geometry of the whole building was reconstructed. In addition, semantic data (floor, part of the building) were included into the generic model. The hierarchical structure of the data

automatically generated semantic level of detail. In this way it was instantly possible to highlight or hide selective parts of the building by addressing to the semantic names (for example hide “floor 2”). Using the same model with minor variations it was also possible to generate different levels of geometric detail for the building.

The straight forward way of rendering a 3D model using OpenGL lighting leads to artificial and sometimes irritating results. Buildings in particular have many planar surfaces, a fact that amplifies this effect (see Section 4.3.2.3). Advanced lighting techniques can create a more realistic and less confusing look but need a lot of performance if calculated in real time. We recommend using professional render software like Autodesk Maya™ to bake the illumination into a texture. OpenSG supports texture compression which should be used for high quality visualizations as the amount of textures can become quite big.

5.2.2.5 Technical Details of the Video Textures

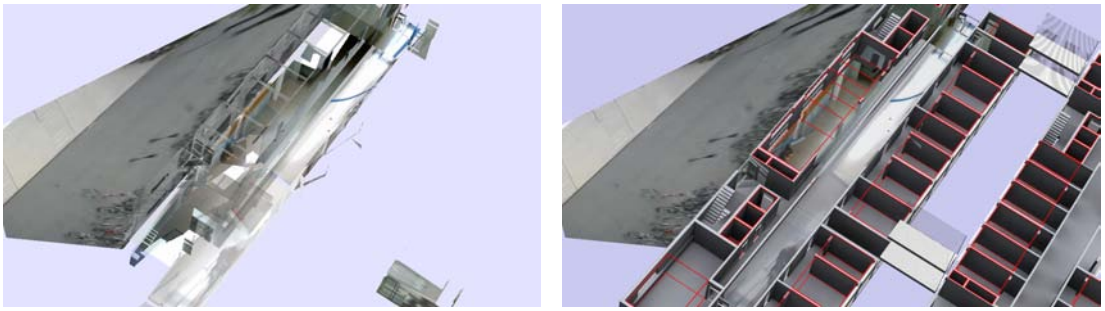


Fig. 5.24 Multiple video textures are added to the scene by duplicating the geometry. The left image shows the output of the video texture shaders which have discarded all pixels outside of the video image. In the right image the video textures are combined with the original scene geometry.

To display a video in the 3D scene we need all of the information in the configuration file described in Section 5.2.2.2. Since we use a single texture unit for each video, there is no upper limit for the number of projected images. As described the image of an input camera and the detection rectangles can be received via network or read from file. The image is converted into a texture with the target resolution which is then used in a special material. For the rectangles we prepare some simple geometry and apply the video texture material. The rectangles will be hidden until some of them are used to display detections.

5.2.2.6 Projective Texturing

Our approach to project the video onto the 3D scene uses the GL shading language (GLSL) which is attached to a material. The material has to be applied to the scene, at least to the parts which are visible in the video. The GLSL shader gets the camera parameters and the video image in form of a texture as inputs. In the fragment shader we use the camera parameters to project the corresponding 3D world position of each pixel into the 2D video image. If the shader calculation discovers a position outside of the video image space, it discards the pixel. As the shaders knows the normal of the rendered pixel it can also discard back facing fragments.

By now this would result in a selective scene showing only geometry within the view frustum of the video camera (see Figure 5.24). To compose the video texture with the original scene, a duplicate is generated and rendered in addition. This allows combining multiple video textures with the texture baked 3D scene even if the video areas are overlapping.

So far the video image would penetrate the whole 3D model within the camera frustum (see Fig. 5.25, left image). We only want to project the images onto the visible surfaces from the camera position (Fig. 5.25, right image). A slightly modified shadow algorithm is used to detect occluded areas. The visualization uses an additional preceding render pass generating a depth map for each surveillance camera. This depth map has to be updated only if the camera description changes. It represents the first hit of the projection rays from the camera. In the final render step the distance between the camera and the currently rendered pixel is compared to the first hit in the depth map. Pixels behind the first hit are also discarded. A small offset is used to avoid artifacts due to numerical errors.

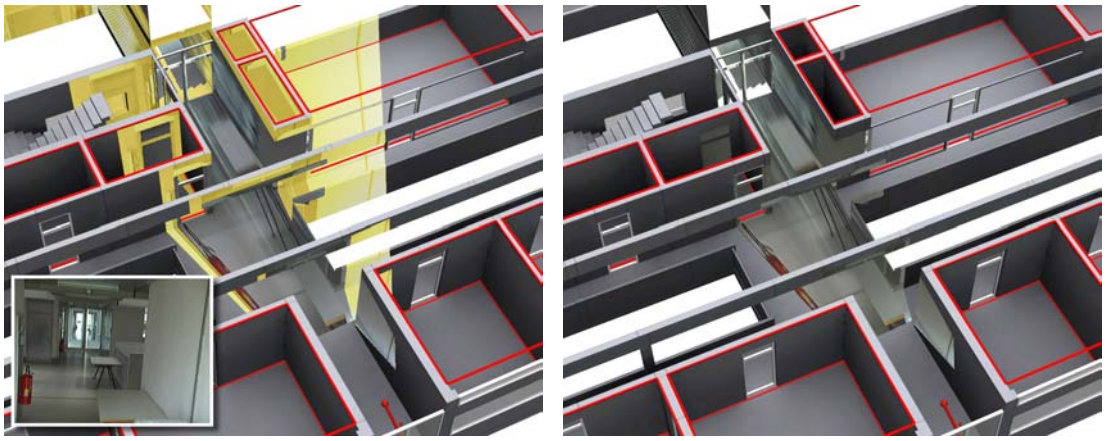


Fig. 5.25 The video image is projected back into the 3D model. The yellow areas in the left figure are in the camera frustum but occluded by closer objects. Thus they are not visible from the camera position. A modified shadow algorithm is used to remove the video image in these areas.

An important aspect is the modulation of the transparency of pixels. We included a fade effect for pixels which are far away from the camera. To reduce the distraction from distorted textures the projected image is also faded to transparent when viewed from an angle very different to the viewing direction of the camera.

5.2.2.7 Visualization of Detected Objects

The detection areas of the video image are added into the 3D scene as planar rectangles. These billboards are oriented to the respective video camera position and not to the viewer. It is assumed that detected objects are always connected to the ground plane that was defined in the configuration file for the video texture. From the position of the camera and with the parameters of the frustum we calculate a ray through the lower edge of the rectangle. By cutting the ground plane with this ray we get the position of the rectangle in 3D space. For some scenarios it may be necessary to use the upper edge of the rectangle because the lower part of the detection may be occluded in the video. In this case a default height of objects is used to estimate the position of the rectangle. The rectangles are placed right-angled to the ground plane.

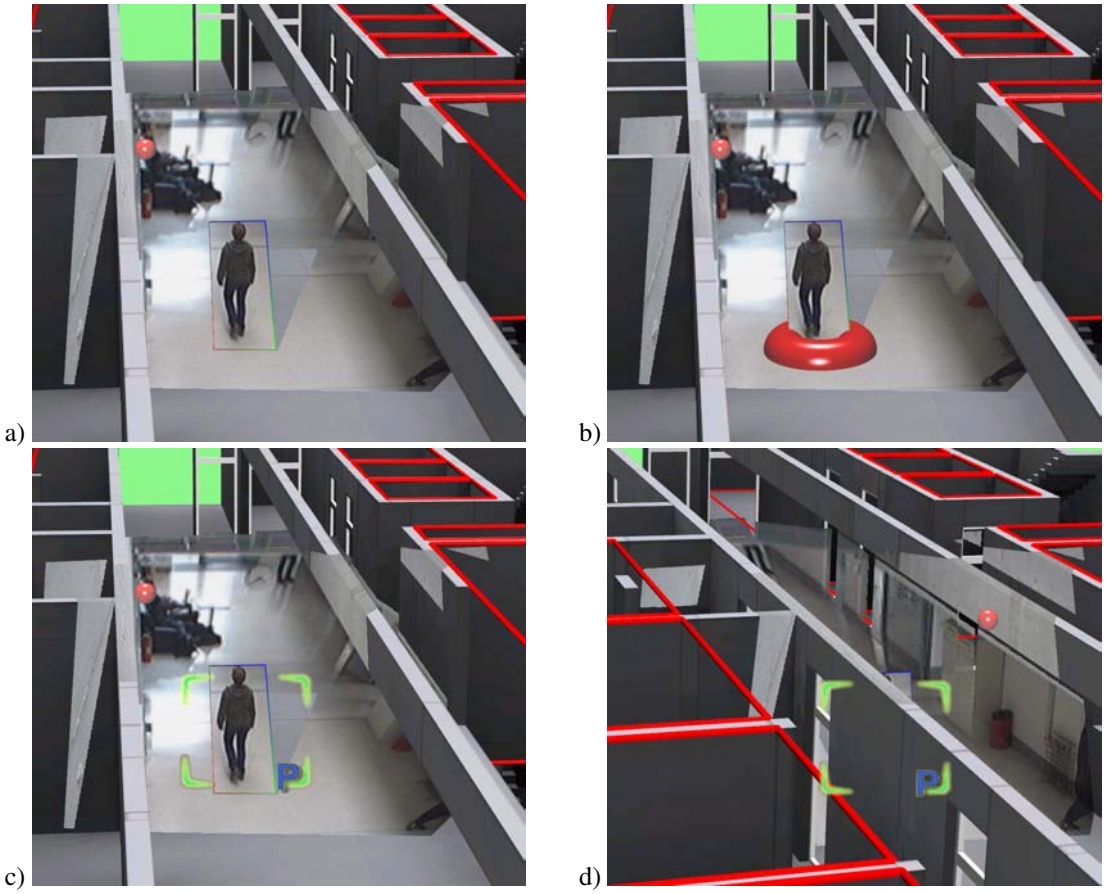


Fig. 5.26 Different representations of a person detection in 3D: (a) Billboard with a thin colored surrounding line, (b) additional geometry, (c, d) overlay marker which is not occluded by the scene.

The first implementation of the billboard technique revealed some drawbacks: 2D billboards are hardly visible from the side or from a top view position. Also in more complex 3D scenes the rectangles may be occluded by walls and other elements. To improve the visibility we added the possibility to connect GML generated geometry with the billboards. This includes shapes like spheres or boxes but also textual information. Overlaid renderings which are always visible help to signalize detections even if the rectangle is occluded by parts of the scene. See Figure 5.26 for some examples of enhanced billboard visualizations.

5.2.2.8 Insight

We propose a real-time visualization system for integrating image based 2D detections into a 3D scene. For visualization we show the natural inverse process of capturing a camera image: the projection of the image into the 3D scene. Optimized shading and transparency effects enhance quality and usability.

The combination of detection results with a 3D scene is very useful for surveillance systems with many cameras. An informal user study shows that watching a moving person over multiple video camera images is much easier in a spatial coherent 3D environment. Other interesting events or detections recognized by different input sensors can also be added to the system and displayed in the same space.

With the scriptable scene graph engine it is possible to create a large variety of applications. Scripting of geometry, materials and user interactions offer a comfortable and fast way of manipulating the application. Using the OpenSG framework we also support large tiled displays and even immersive systems like the DAVE.

5.3 Mobility

5.3.1 Project mPed+

Project leader: Österreichisches Forschungs- und Prüfzentrum Arsenal GmbH (AIT Mobility), partners: Fraunhofer Austria Research GmbH, Wiener Linien GmbH & Co KG, Evolit
This project is funded by BMVIT ways2go.

The increase in mobility demand especially within urban areas puts strategies for efficient transportation and resource management into the focus of transport operators. In order to answer manifold questions which arise within large-scale public transportation networks substantial and realistic pedestrian flow simulation becomes more important. However, currently available simulation programs only allow to model pedestrian flows using a pre-specified simulation model with a fixed level of detail. Therefore, the simulation of complex public transportation network faces at least one of the following two problems at all times:

1. A highly detailed model generates precise information, but it is not capable to calculate in reasonable time
2. A less detailed model requires lower performance in terms of calculation, though it cannot provide sufficient information

These two extremes can only be overcome if a simulation program is able to combine several simulation models with different levels of detail. No currently commercially available simulation program supports such an approach. Moreover, simply combining existing programs to simulate a public transportation network is also not feasible due to the involved complexity (with respect to data exchange), efficiency (with respect to computation performance) and high costs (several, expensive programs are needed). Furthermore, the exchange of passengers across the simulation boundaries of the various simulation models is still an open research field.

The goal of mPed+ is to close this gap by developing the prototype of a simulation program which allows to combine several different simulation models and to calculate them simultaneously based on a highly scalable system architecture. This high flexibility is achieved by loose coupling of the models that consequently allows exchanging single simulation models in order to use the right model for each region and hence to invest the computation power where it is really needed. In order to make the corresponding results easily accessible for the user mPed+ develops and prototypically implements a user-friendly 3D visualization for simulation results considering special requirements of the individual operator. This also involves a nontrivial unification of the simulation output of the various models. The visualization module is OpenSG-based and uses GML geometry for selected objects. Figure 5.27 shows a screen shot of the output. The test scenario is a model of a subway station in Vienna.

Using the simulation program from mPed+ public transportation operators and planers can simulate and visualize pedestrian flows with an adaptive level of detail in high complex large-scale public transportation networks for the first time. Accordingly, one application field for the simulation program is to simulate the whole subway network of a large city covering pedestrian

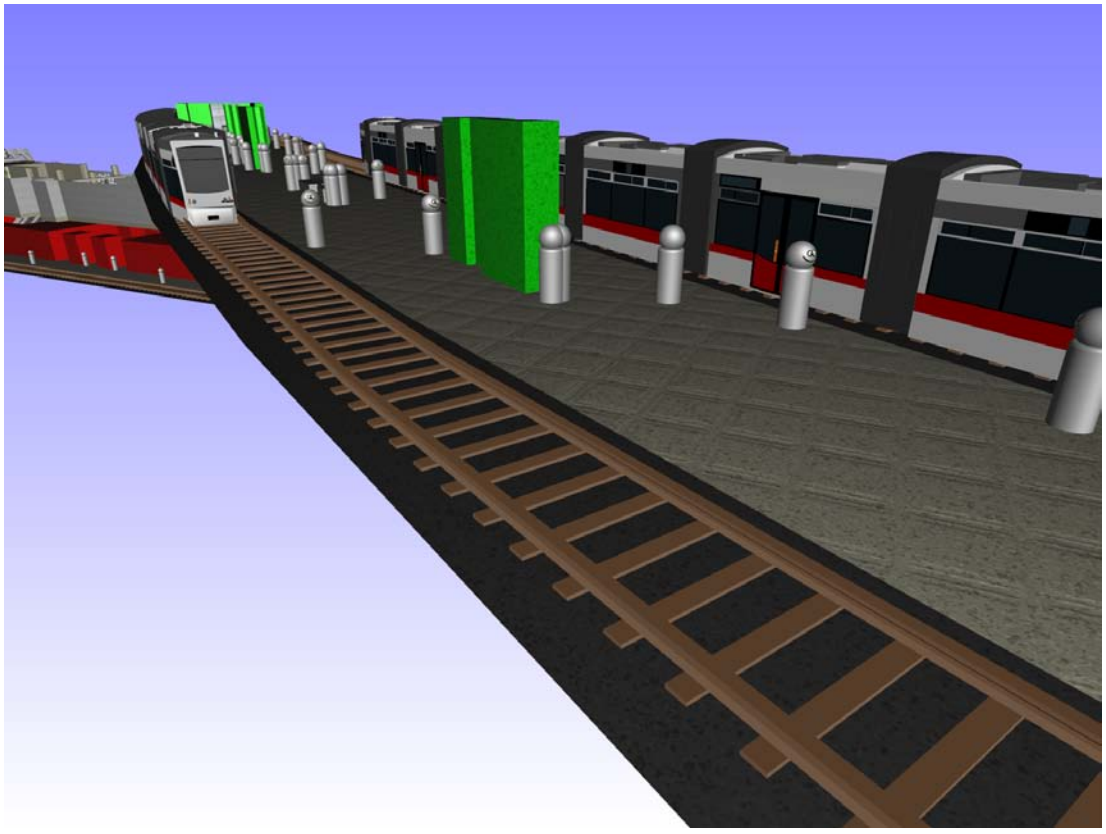


Fig. 5.27 A simulated subway station from the mPed+ project. The railway tracks are generated using GML, the visualization is based on OpenSG.

flows in all stations as well as in all means of transport such as trains connecting the stations. Furthermore, the simulation program allows determining and evaluating control measures for pedestrian flows and train intervals. The research is based on the results of the previous fundamental research project mPed, where a mathematical microscopic pedestrian simulation model was developed as well as calibrated and validated using empirical data from real world. Hence, mPed+ transforms previously fundamental research into a simulation program that is expected to have a mid-range time to market. Using the results of the simulation program transportation infrastructures and processes can be optimized leading to an increase in attractiveness and acceptance of public transportation.

5.3.2 Project IMITATE

Project leader: Österreichisches Forschungs- und Prüfzentrum Arsenal GmbH (AIT Mobility), partners: Komobile Gmunden, Merten Management, im-plan-tat Reinberg und Partner, Fraunhofer Austria Research GmbH, ÖBB-Infrastruktur AG
This project is funded by BMVIT ways2go.

Modern transport infrastructures fulfill a variety of functions which make the design of guidance systems ever more complicated. In addition guidance systems are also increasingly used for crowd control measures in order to increase a given infrastructures capacity and ensure passenger safety. Static guidance systems cannot meet these increasing demands and cannot

be adjusted according to the needs of specific groups (the elderly, people with prams, tourists, sight-impaired people etc.). For many people disorientation and subjective insecurity, which are caused by the increasing complexity of traffic junctions, present a psychological barrier to mobility. As of late more and more dynamic, mobile guidance systems have been developed which could potentially be adjusted to specific circumstances and individual needs.

The technical preconditions for their implementation have been developed over the past years. While research and product management have focused on technical implementation, questions concerning usability and the acceptance by the groups addressed were however neglected. Adapting transport infrastructures for a new dynamic, mobile guidance system is very expensive hence testing different systems for their usability is difficult. Also, measuring the effects of a new guidance system under real conditions is nearly impossible due to underdeveloped measuring methods and the impossibility of controlling environmental conditions. These include e.g. the existing static guidance system, illumination and the number of people currently in the infrastructure. Hence IMITATE will develop a controlled test environment which makes it possible to control environmental conditions, to measure the interaction of a given test person with the infrastructure and other passengers and to adjust static, dynamic and even mobile guidance systems as needed. This test environment makes it possible for the first time to measure the effects of situation-specific personalized guidance systems, investigate the acceptance of components of the guidance system and consequently adapt these in a cost-efficient way.

The controlled test environment is based on the immersive virtual environment DAVE (see Section 4.2.3.1). IMITATE enriches the DAVE by including other pedestrians, typical noise conditions, realistic illumination and the Kinect navigation described in Section 4.5.3. Since the test person can move freely within the DAVE, it is - in contrast to head-mounted-displays – possible to use numerous mobile guidance systems, from a conventional map to mobile phones. Situations as experienced during traveling can be realistically reenacted by using e.g. suitcases.

To evaluate the significance of the results in the test environment, a user study compares the same task in the real world and in virtual reality. The test persons have to travel through a model of the Graz University of Technology area using the Kinect navigation method or through the real area (see Figure 5.28). Simple orientation tasks are used to investigate in which way the artificial scene alters the perception of the test persons and therefore influence the test results.

The ongoing project will culminate in the investigation of two specific applications in two case studies using the example of the currently planned Vienna central station. On the one hand a mobile guidance system for groups with special mobility needs will be investigated, which is currently being developed as part of the project ways4all. On the other hand the potential of guiding people by means of LED line lights will be examined. IMITATE will develop guidelines for the planning of guidance systems that will result from a combination of experiences within the project and comprehensive research. In addition IMITATE creates a controlled test environment which can be used as an essential planning tool for the development of guidance systems and as a scientific tool for the analysis of a variety of research questions.

One follow-up project, “MOVING”, has already started in October of 2012. MOVING will continue to create methods for the evaluation and planning of guidance systems. It will be focused on the evaluation of mobile navigation devices, and it will include the use of an eye tracker in the DAVE. Within the project the controlled test environment will be further improved. The continuing work with the DAVE will open many new development possibilities for various applications.

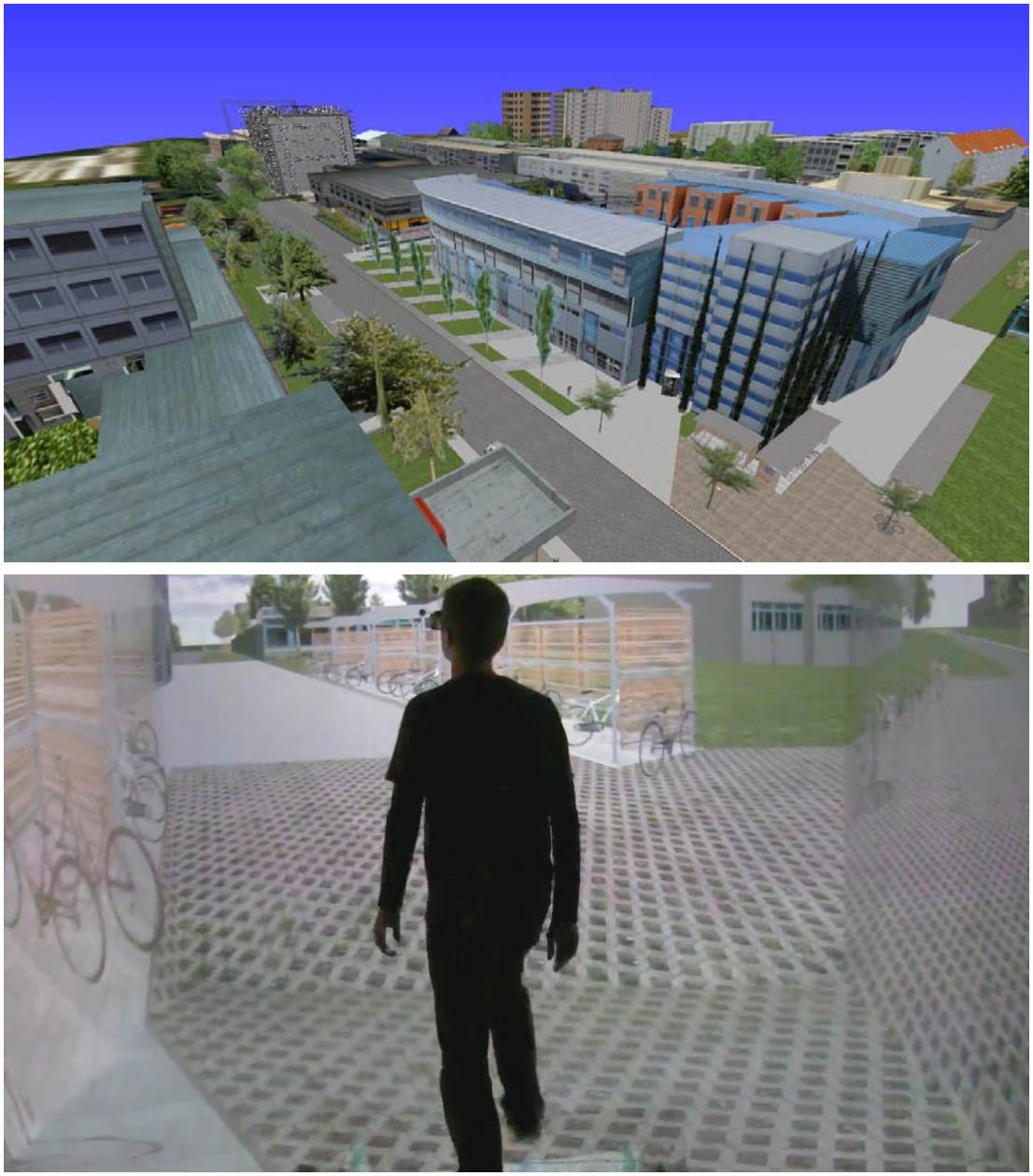


Fig. 5.28 Evaluation of the distance perception and orientation in the DAVE. The test person has to travel through a model of the Graz University of Technology area using the Kinect navigation method. He is asked to estimate the traveled distance.

Chapter 6
Conclusion

Within this thesis new ways of handling semantic data in interactive 3D visualizations are presented. The symbiosis of 3D content and semantic information is an important aspect for improving the usage of virtual environments in various fields of applications. The main goal of my thesis is to emphasize the importance of semantic enrichment for interactive visualizations.

6.1 Contribution and Benefit

The contributions cover the whole processing chain for 3D content: the **creation**, the **semantic enrichment** and in the end the **presentation** and **publication** respectively. My work is the basis of a number of ready-to-use **applications**.

Creation:
In Chapter 2 creation methods are presented and compared. It is pointed out that the creation process is a large field with many different data structures, file formats and creation work flows. A focal point of my thesis is the processing and presentation of digitalized artifacts. Suitable creation methods are presented, among others the automatic reconstruction using photos. By digitalizing plaster statues I describe the work flow from photos to 3D models using the ARC 3D web service. The solution is completely based on open source projects and freely available services making it affordable even for small museums. The entire work flow from 3D acquisition over markup and annotation to web delivery is published in *"A Publishing Work flow for Cultural Heritage Artifacts from 3D-Reconstruction to Internet Presentation"* [BBH*10].
Often the creation process involves the adjustment of existing 3D content. In Section 4.3 I describe common problems when dealing with unstructured data and propose suitable measures. The necessity of semantic information for a semi-automatic preparation of the 3D data is illustrated with example data from digital engineering and architectural origin. These valuable guide lines are published also in *"The Preparation of 3D-Content for Interactive Visualization"* [SEF12].

Semantic enrichment:
Semantic information adds additional value to the data and enhances the presentation of 3D content. In Chapter 3 I propose a classification scheme for semantic markup which will support further research in this area. The classification scheme is published in *"Semantic Enrichment for 3D Documents Techniques and Open Problems"* [USB10]. Furthermore I present an acquisition method to transfer semantic knowledge of higher order to unstruc-

tured, automatically created geometry, e.g., point clouds from a laser scanner. In this way the point cloud can be segmented and classified according to the meaning of its parts. Automatic methods for the acquisition of semantic data are very important as large amounts of 3D data are acquired already in an automatic way.
My colleagues and I discuss the requirements for the integration of 3D models into cultural heritage libraries [HSKF06]. In Section 3.4 two suggestions for a tight connection of 3D content and semantic data are proposed. With the help of XML attachments to scene graph nodes I developed a generic basis for annotation and presentation applications for cultural heritage [BHSF08]. Together with my co-authors I present a PDF-based solution for the sustainable publication of 3D content with semantic information [SBS*09]. Both suggestions for the combination of semantic data and 3D content are a sound foundation for future research. 3D models based on culture historical artifacts will be integrated into the wider context of knowledge sources. In the long run this could help to change 3D content from technical toys for the few to elements in the toolbox used by researchers, curators and the general public as part of their daily work with cultural information.

Presentation/Publication:
The contributions in the area of presentation and publication can be further divided into content oriented achievements and technical basics.
The content oriented contributions are dealing with cultural heritage. Sustainable linking of semantic data with 3D geometry is an important aspect for cultural heritage but also in other areas dealing with 3D computer graphics. The considerations made by my co-authors and me are helpful guidelines for further work in this area.
Technical basics are the contributions for the DAVE (see Section 4.2.3.1) and immersive environments in general. Novel navigation methods for immersive environments are presented in Section 4.5. Especially the affordable simulation of a walk experience is a valuable contribution for the evaluation of large pedestrian areas like train stations. The traveling method proposed in Section 4.5.3 helps to estimate traveled distances without installing expensive and large hardware. The perception of distances and walking time will be the focus of further investigations in this respect. This navigation builds the cornerstone for the development of a test lab. It has been successfully used in the project “IMITATE”. In the follow-up project “MOVING” which started in October 2012, it will be used to evaluate the effectiveness of smartphone based indoor navigation systems.
With the RESTful web service for OpenSG a useful alternative is proposed for controlling a scene graph and virtual reality applications at run time. The architecture proposed in Section 4.1.3 can be added to an existing OpenSG 2 application with minimal changes to the original code [SBS*10]. The web service interface opens new possibilities to modify the scene graph and to control the application. My work is the basis for a more general solution using a RESTful web service to manipulate arbitrary data of visualization applications.
Including the GML into OpenSG allows to use procedural models in the scene graph. The scripting functionality described in Section 4.1.2.2 has proved to be of value in different applications. It is the basis for the applications described in Chapter 5 but also for current research and projects like 3D-COFORM and V2me[1] [ZTH*12].
Furthermore, in Section 4.6 I propose a method that helps to integrate legacy software into tiled displays and other multi-projection systems. In this way it is possible to use novel environments also for every day tasks, e.g., Powerpoint-presentations, without modifications to such applications. We showed examples running on a tiled display with a high resolu-

[1] http://www.v2me.org/

tion, in an immersive environment and on a multi-touch table. With the possibility to freely transform application windows we can solve the orientation problem of tabletop screens. The system can also be used to manipulate the visual appearance of a desktop. User inputs can be mapped from various input devices to the common keyboard and mouse input events that the legacy software can process.

Applications:

All the techniques and guidelines have led to several applications: With the software tool "Abstand" by T. Ullrich and me the overall work flow to produce high-quality visualizations of distances between surfaces has been reduced significantly [USF08]. Furthermore the application takes the latest results in human perception and visualization techniques into account (see Section 4.4.1).

Together with my co-authors I developed the 3D-Powerpoint application [HSLF07], which later on evolved to the EpochViewer [SBS*09]. We created a complete set of tools for sustainable markup and publication of 3D cultural heritage content. Additionally we improved the GML scripting for OpenSG and included our methods into an ActiveX module. With this module our annotation and presentation tool can be integrated into existing applications.

In Section 5.2.2 I propose a real-time visualization system for integrating image based 2D detections into a 3D scene [SLHF09]. For the visualization the inverse process of capturing a camera image is used: the projection of the image into the 3D scene. Optimized shading and transparency effects enhance quality and usability. With the proposed spatial coherent solution it is much easier to watch a moving person over multiple video camera images.

6.2 Future Work

The results of this thesis lay out the foundation for new challenges and ongoing research. In particular three aspects are of special interest:

Integrated creation of semantic data
Semantic information should be added as early as possible in the process of content creation. It is important to support designers and other creators in this step by adding functionality to the creation application. A semi-automatic augmentation with semantic knowledge is desirable. The integration of procedural modeling can also help to transfer the meaning of shape to 3D content. Tools for generative model creation have to be more accessible to designers and 3D artists.

Sustainable linking of semantic information with 3D content
Historic 3D content will become a common part of cultural heritage repositories. The definition of a general standard format for semantic data is an important issue which can only be realized in a large interdisciplinary community. The presented suggestions for sustainable semantic data linking require the approval of a larger group of real users in practical cultural heritage applications. To have XML support is fine, but much more important is to define how it is used: What are the best solutions for the problems of 3D markup and processing history, which metadata scheme is really useful, and which XML technology should be applied for which purpose.

Productivity of VR environments
For the usefulness of virtual reality environments and for cost reduction it is important to minimize the manual preparation of 3D content. Only in this way virtual engineering is a useful part in the development of new designs and improved products. Semantically enriched content helps to identify the importance level of parts in a large scene with many objects. Special environments like the CAVE^{TM} should generate valuable results that can be used directly within the working process. This can be achieved by keeping the knowledge of shape within the 3D content.
Another important aspect is the user interaction with the virtual world. Further improvements in the man-machine communication will help to increase the acceptance of novel virtual environments.

Appendix A
List of Publications

Adaptive Tesselation of Subdivision Surfaces
V. Settgast, K. Müller, C. Fünfzig and D. W. Fellner
(Computers & Graphics 2004)

On the Integration of 3D Models into Digital Cultural Heritage Libraries
S. Havemann, V. Settgast, H. Krottmaier and D. W. Fellner
(International Symposium on Virtual Reality, Archaeology and
Cultural Heritage, 2006)

3D-Powerpoint - Towards a design tool for digital exhibitions of cultural artifacts
S. Havemann, V. Settgast, M. Lancelle and D. W. Fellner
(International Symposium on Virtual Reality, Archaeology and
Cultural Heritage, 2007)

Distance Calculation between a Point and a Subdivision Surface
T. Ullrich, V. Settgast, U. Krispel, C. Fünfzig and D. W. Fellner
(Vision, Modeling and Visualization 2007)

Information Technology for Cultural Heritage
V. Settgast, T. Ullrich and D. W. Fellner
(IEEE Potentials, 2007)

Self-paced exploring of the Austrian National Library through thought
R. Leeb, V. Settgast, D. W. Fellner and G. Pfurtscheller
(Proceedings of the Neuromath Workshop 2007)

Semantic Fitting and Reconstruction
T. Ullrich, V. Settgast and D. W. Fellner
(Journal on Computing and Cultural Heritage, 2008)

Sustainable Markup and Annotation of 3D Geometry
R. Berndt, S. Havemann, V. Settgast and D. W. Fellner
(14th International Conference on Virtual Systems and Multimedia)

Intuitive Navigation in Virtual Environments
M. Steiner, P. Reiter, C. Ofenböck, V. Settgast, T. Ullrich,
M. Lancelle and D. W. Fellner
(Eurographics Symposium on Virtual Environments, 2008)

Abstand: Distance Visualization for Geometric Analysis
T. Ullrich, V. Settgast and D. W. Fellner
(International Conference on Virtual Systems and Multimedia, 2008)

The Arrigo Showcase Reloaded -
Towards a Sustainable Link between 3D and Semantics
S. Havemann, V. Settgast, R. Berndt, Ø. Eide and D. W. Fellner
(International Symposium on Virtual Reality, Archaeology and
Cultural Heritage, 2008)

Spatially Coherent Visualization of Image Detection Results using Video Textures
V. Settgast, M. Lancelle, S. Havemann and D. W. Fellner
(33rd Workshop of the Austrian Association for Pattern Recognition, 2009)

Desktop Integration in Graphics Environments
T. Ullrich, V. Settgast, C. Ofenböck and D. W. Fellner
(Joint Virtual Reality Conference of EGVE - ICAT - EuroVR 2009)

Publishing 3D Content as PDF in Cultural Heritage
M. Strobl, V. Settgast and R. Berndt, S. Havemann and D. W. Fellner
(International Symposium on Virtual Reality, Archaeology and
Cultural Heritage, 2009)

A Publishing Workflow for Cultural Heritage Artifacts
from 3D-Reconstruction to Internet Presentation
R. Berndt, G. Buchgraber, S. Havemann, V. Settgast and D. W. Fellner
(Digital Heritage, Third International Conference, Euromed 2010)

Semantic Enrichment for 3D Documents Techniques and Open Problems
T. Ullrich, V. Settgast and R. Berndt
(14th International Conference on Electronic Publishing (elPub), 2010)

Service-Oriented Scene Graph Manipulation
A. Schiefer, R. Berndt, T. Ullrich, V. Settgast and D. W. Fellner
(15th International Conference on 3D Web Technology, Web3D 2010)

Enlightened by the Web - A service-oriented architecture for
real-time photorealistic rendering
T. Schiffer, A. Schiefer, R. Berndt, T. Ullrich, V. Settgast and D. W. Fellner
(05. Kongress Multimediatechnik Wismar, 2010)

Next-Generation 3D Visualization for Visual Surveillance
P. M. Roth, V. Settgast, P. Widhalm, M. Lancelle, J. Birchbauer,
N. Brändle, S. Havemann and H. Bischof
(8th IEEE International Conference on Advanced Video and
Signal-Based Surveillancei (AVSS), 2011)

Hands-Free Navigation in Immersive Environments
V. Settgast, M. Lancelle, D. Bauer and D. W. Fellner
(9. Workshop der GI-Fachgruppe VR/AR, 2012)

The Preparation of 3D-Content for Interactive Visualization
V. Settgast, E. Eggeling, D. W. Fellner
(9. Fachtagung Digitales Engineering zum Planen, Testen und Betreiben technischer Systeme
(15. Fraunhofer IFF-Wissenschaftstage), 2012)

References

AB06. ARNAUD R., BARNES M. C.: *Collada: Sailing the Gulf of 3D Digital Content Creation*. AK Peters Ltd, 2006. See pages 15 and 185.

ABK98. AMENTA N., BERN M., KAMVYSSELIS M.: A New Voronoi-Based Surface Reconstruction Algorithm. *Computer Graphics 32* (1998), 415–421. doi:10.1145/280814.280947. See page 39.

ADS06. AUGSDÖRFER U. H., DODGSON N. A., SABIN M. A.: Tuning Subdivision by Minimising the Gaussian Curvature Variation Near Extraordinary Vertices. In *Computer Graphics Forum* (2006), vol. 25, pp. 263–272. doi:10.1111/*j*.1467−8659.2006.00945.*x*. See page 10.

AKMHP09. AMATRIAIN X., KUCHERA-MORIN J., HÖLLERER T., POPE S. T.: The allosphere: Immersive multimedia for scientific discovery and artistic exploration. *IEEE MultiMedia 16*, 2 (2009), 64–75. doi:10.1109/*MMUL*.2009.35. See page 99.

Arn06. ARNOLD D.: Procedural methods for 3D reconstruction. *Recording, Modeling and Visualization of Cultural Heritage 1* (2006), 355–359. See page 39.

AS05. ADOBE SYSTEMS I.: 3D Annotations Tutorial. *Adobe Solutions Network 7* (2005), 1–16. See page 32.

ASCE02. ASPERT N., SANTA-CRUZ D., EBRAHIMI T.: Mesh: Measuring error between surfaces using the hausdorff distance. *Proceedings of the IEEE International Conference on Multimedia and Expo 2002 (ICME) 1* (2002), 705–708. See page 122.

AVJ07. ADAMO-VILLANI N., JONES D.: Travel in sMILE: A study of two immersive motion control techniques. In *Proceedings of IADIS International Conference on Computer Graphics and Visualization 2007* (2007), IADIS, pp. 43–49. See page 131.

Bal04. BALLING J. S.: Visualization, Annotation, and Exploration of Three Dimensional Datasets Using an Existing 3D Rendering System. *BioEngineering Senior Design 1* (2004), 1–6. URL: `http://www.sci.utah.edu/~balling/TylerAndJanna/Janna/index_files/SeniorProject_JannaBalling.doc`. See page 33.

BB09. BI X., BALAKRISHNAN R.: Comparing Usage of a Large High-Resolution Display to Single or Dual Desktop Displays for Daily Work. *Conference on Human Factors in Computing Systems 27* (2009), 1005–1014. doi:10.1145/1518701.1518855. See page 150.

BBCS99. BERNARDINI F., BAJAJ C. L., CHEN J., SCHIKORE D. R.: Automatic Reconstruction of 3D CAD Models from Digital Scans. *International Journal on Computational Geometry and Applications 9*, 4-5 (1999), 327–369. See page 39.

BBH03. BUES M., BLACH R., HASELBERGER F.: Sensing surfaces: bringing the desktop into virtual environments. *Proceedings of the workshop on Virtual Environments (EGVE) 39* (2003), 303–304. doi:10.1145/769953.769989. See page 150.

BBH05. BECKHAUS S., BLOM K. J., HARINGER M.: Intuitive, Hands-free Travel Interfaces for Virtual Environments. *Proceedings of the 2005 IEEE Conference on Virtual Reality, Workshop on New Directions in 3D User Interfaces 1* (2005), 57–60. See page 132.

BBH*10. BERNDT R., BUCHGRABER G., HAVEMANN S., SETTGAST V., FELLNER D. W.: A publishing workflow for cultural heritage artifacts from 3d-reconstruction to internet presentation. In *Digital Heritage. Third International Conference, EuroMed 2010* (2010), Ioannides M., Fellner D. W., Georgopoulos A.,, Hadjimitsis D., (Eds.), vol. 6436, Springer, pp. 166–178. doi:10.1007/978−3−642−16873−4$_1$3. See pages 16 and 199.

BD07. BRUTZMAN D., DALY L.: *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann, 2007. See page 15.

BDA99. BOURDOT P., DROMIGNY M., ARNAL L.: Virtual navigation fully controlled by head tracking. *Proceedings of the 7th International Scientific Workshop on Virtual Reality and Prototyping 7* (1999), 1–9. URL: http://www-sop.inria.fr/epidaure/Collaborations/GT-RV/JT-GT-RV7/. See page 131.

BDR04. BEHR J., DÄHNE P., ROTH M.: Utilizing X3D for immersive environments. *Proceedings of the ninth international conference on 3D Web technology 9* (2004), 71–78. doi:10.1145/985040.985051. See page 90.

BF06. BEDER C., FÖRSTNER W.: Direct Solutions for Computing Cylinders from Minimal Sets of 3D Points. *Proceedings of the European Conference on Computer Vision 2006 3951* (2006), 135–146. doi:10.1007/11744023$_1$1. See page 45.

BF08. BARNES D. G., FLUKE C. J.: Incorporating interactive 3-dimensional graphics in astronomy research papers. *New Astronomy 13*, 8 (Nov. 2008), 599–605. doi:10.1016/*j.newast*.2008.03.008. See pages 16 and 69.

BFH05. BERNDT R., FELLNER D. W., HAVEMANN S.: Generative 3D models: a key to more information within less bandwidth at higher quality. *Proceeding of the Tenth International Conference on 3D Web Technology 1* (2005), 111–121. doi:10.1145/1050508. See pages 12 and 60.

BGHF05. BERNDT R., GERTH B., HAVEMANN S., FELLNER D. W.: 3D Modeling for Non-Expert Users with the Castle Construction Kit. *Proceedings of ACM/EG VAST 2005 1* (2005), 1–9. doi:10.2312/*VAST*/*VAST*05/049−057. See page 29.

BGVGP06. BALTSAVIAS M., GRUEN A., VAN GOOL L., PATERAKI M.: *Recording, Modeling and Visualization of Cultural Heritage*. Taylor & Francis, 2006. See page 122.

BGVOM06. BILASCO I. M., GENSEL J., VILLANOVE-OLIVER M., MARTIN H.: An MPEG-7 framework enhancing the reuse of 3D models. *Proceedings of the International Conference on 3D web technology 11* (2006), 65–74. doi:10.1145/1122591.1122601. See page 35.

BHKN03. BIRBAUMER N., HINTERBERGER T., KBLER A., NEUMANN N.: The thought-translation device (ttd): neurobehavioral mechanisms and clinical outcome. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 11* (2003), 120–123. See page 142.

BHM*04. BOOTH D., HAAS H., MCCABE F., NEWCOMER E., CHAMPION M., FERRIS C., ORCHARD D.: Web Services Architecture. *World Wide Web Consortium 20040211* (2004), 1–98. See page 81.

BHSF08. BERNDT R., HAVEMANN S., SETTGAST V., FELLNER D. W.: Sustainable markup and annotation of 3d geometry. In *Proceedings of the 14th International Conference on Virtual Systems and Multimedia (VSMM 2008)* (2008), Ioannides M., (Ed.), International Society on Virtual Systems and MultiMedia, pp. 187–193. See page 200.

Bie87. BIEDERMAN I.: Recognition-by-Components: A Theory of Human Image Understanding. *Psychological Review 94*, 2 (1987), 115–147. See page 42.

BK98. BROWN N., KINDEL C.: Distributed Component Object Model Protocol – DCOM/1.0, 1998. See page 81.

BKH97. BOWMAN D. A., KOLLER D., HODGES L. F.: Travel in Immersive Virtual Environments: An Evaluation of Viewpoint Motion Control Techniques. *Proceedings of the 1997 Virtual Reality Annual International Symposium 5*, 7 (1997), 45–52. doi:10.1109/*VRAIS*.1997.583043. See page 131.

BKLP04. BOWMAN D. A., KRUIJFF E., LAVIOLA J. J., POUPYREV I.: *3D User Interfaces: Theory and Practice*. Addison-Wesley Professional, 2004. URL: http://dl.acm.org/citation.cfm?id=993837. See page 131.

BKSS07. BUSTOS B., KEIM D., SAUPE D., SCHRECK T.: Content-based 3D Object Retrieval. *IEEE Computer Graphics and Applications 27*, 4 (2007), 22–27. See page 42.

BKV*02. BENKO P., KÓS G., VÁRADY T., ANDOR L., MARTIN R.: Constrained Fitting in Reverse Engineering. *Computer Aided Geometric Design 19*, 3 (2002), 173 – 205. doi:10.1016/S0167 – 8396(01)00085 – 1. See page 39.

BLZ00. BIERMANN H., LEVIN A., ZORIN D.: Piecewise Smooth Subdivision Surfaces with Normal Control. In *Proceedings of SIGGRAPH 2000* (2000), pp. 113–120. See page 10.

BMB06. BORISOFF J. F., MASON S. G., BIRCH G. E.: Brain interface research for asynchronous control applications. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 14* (2006), 160–164. See page 142.

BMBB04. BORISOFF J. F., MASON S. G., BASHASHATI A., BIRCH G. E.: Brain-computer interface design for asynchronous control applications: improvements to the lf-asd asynchronous brain switch. *IEEE Transactions on Biomedical Engineering 51* (2004), 985–992. See page 148.

BMG08. BACU V., MURESAN L., GORGAN D.: Cluster Based Modeling and Remote Visualization of Virtual Geographical Space. *Proceedings of the 2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing 10* (2008), 416–421. doi:10.1109/SYNASC.2008.54. See page 90.

BMWB06. BASHASHATI A., MASON S., WARD R. K., BIRCH G. E.: An improved asynchronous brain interface: making use of the temporal history of the lf-asd feature vectors. *J Neural Eng 3* (2006), 87–94. See page 148.

BMZ04. BOIER-MARTIN I., ZORIN D.: Differentiable Parameterization of Catmull-Clark Subdivision Surfaces. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 71* (2004), 155 – 164. doi:10.1145/1057432.1057453. See page 9.

BR02. BERNARDINI F., RUSHMEIER H.: The 3D Model Acquisition Pipeline. *Computer Graphics Forum 21*, 2 (2002), 149–172. doi:10.1111/1467 – 8659.00574. See page 19.

Bri51. BRIET S.: *Qu'est-ce que la documentation?* ÉDIT - éditions documentaires industrielles et techniques, 1951. URL: `http://people.ischool.berkeley.edu/~buckland/briet.html`. See page 32.

BRT95. BERGMAN L. D., ROGOWITZ B. E., TREINISH L. A.: A rule-based tool for assisting colormap selection. *Proceedings of the 6th Conference on Visualization 6* (1995), 118–125. doi:10.1109/VISUAL.1995.480803. See page 122.

BSD08. BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 22:1–22:1. doi:10.1145/1401032.1401061. See page 116.

BSly. BBC, SILICON GRAPHICS (SGI): Mummy: Inside story, 2004 July. Exhibition at the British Museum London, http://www.thebritishmuseum.ac.uk. See page 173.

BTI07. BORLAND D., TAYLOR II R. M.: Rainbow color map (still) considered harmful. *IEEE Computer Graphics and Applications 27*, 2 (2007), 14–17. doi:10.1109/MCG.2007.323435. See page 122.

Buc97. BUCKLAND M. K.: What is a "document"? *Journal of American Society of Information Science 48*, 9 (1997), 804–809. URL: `http://people.ischool.berkeley.edu/~buckland/whatdoc.html`. See page 32.

Bux09. BUXTON B.: Multi-Touch Systems that I Have Known and Loved. online, 2009. URL: `http://www.billbuxton.com/multitouchOverview.html`. See page 150.

CBC*01. CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and Representation of 3D Objects with Radial Basis Functions. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques 28* (2001), 67 – 76. doi:10.1145/383259.383266. See page 39.

CC78. CATMULL E., CLARK J.: Recursively Generated B-spline Surfaces On Arbitrary Topological Meshes. *Computer Aided Design 10* (1978), 350–355. See pages 8 and 9.

CCC*08. CIGNONI P., CALLIERI M., CORSINI M., DELLEPIANE M., GANOVELLI F., RANZUGLIA G.: Meshlab: an open-source mesh processing tool. In *Sixth Eurographics Italian Chapter Conference* (2008), pp. 129–136. URL: http://vcg.isti.cnr.it/Publications/2008/CCCDGR08. See page 68.

CCMW01. CHRISTENSEN E., CURBERA F., MEREDITH G., WEERAWARANA S.: *Web Service Definition Language (WSDL)*. Tech. Rep. NOTE-wsdl-20010315, World Wide Web Consortium, March 2001. URL: http://www.w3.org/TR/wsdl. See page 81.

CDG*05. CROFTS N., DOERR M., GILL T., STEAD S., STIFF M.: *Definition of the CIDOC Conceptual Reference Model*, version 4.2 ed. CIDOC Documentation Standards Working Group, June 2005. also ISO/PRF 21127 (almost). cidoc.ics.forth.gr. See pages 36 and 185.

CGVT03. CIGER J., GUTIERREZ M., VEXO F., THALMANN D.: The Magic Wand. *Proceedings of 2003 Spring Conference on Computer Graphics 19* (2003), 132–138. doi:10.1145/984952.984972. See page 131.

CLK*00. COLLINS R., LIPTON A., KANADE T., FUJIYOSHI H., DUGGINS D., TSIN Y., TOLLIVER D., ENOMOTO N., HASEGAWA O.: *A System for Video Surveillance and Monitoring*. Tech. Rep. CMU-RI-TR-00-12, Robotics Institute, Pittsburgh, PA, May 2000. See page 189.

CMFP02. COSTAGLIOLA G., MARTINO S. D., FERRUCCI F., PITTARELLO F.: An approach for authoring 3D cultural heritage exhibitions on the web. In *SEKE '02: Proc. 14th int'l conference on Software and knowledge engineering* (New York, NY, USA, 2002), ACM Press, pp. 601–608. doi:10.1145/568760.568865. See page 172.

CNSD*92. CRUZ-NEIRA C., SANDIN D. J., DEFANTI T. A., KENYON R. V., HART J. C.: The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM 35* (1992), 64–72. doi:10.1145/129888.129892. See page 99.

CoC99. COMMITTEE ON CATALOGING D. . A.: Task Force on Metadata - Summary Report. *American Library Association 4* (1999), 1–16. URL: http://www.libraries.psu.edu/tas/jca/ccda/tf-meta3.html. See page 32.

Cor09. CORPORATION M.: *Remote Desktop Protocol: Basic Connectivity and Graphics Remote Specification (Version 9.0)*. Microsoft Corporation, 2009. URL: http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/%5BMS-RDPBCGR%5D.pdf. See page 150.

CPCS06. CALLIERI M., PONCHIO F., CIGNONI P., SCOPIGNO R.: Easy Access to Huge 3D Models of Works of Art. In *Fourth Eurographics Italian Chapter 2006* (Graz, Austria, 2006), Fellner D., (Ed.), Graz University of Technology, Eurographics Association, pp. 29–36. Catania, 22-24 February 2006. URL: http://vcg.isti.cnr.it/Publications/2006/CPCS06. See pages 54, 172, and 182.

CPCS08. CALLIERI M., PONCHIO F., CIGNONI P., SCOPIGNO R.: Virtual Inspector: a flexible visualizer for dense 3D scanned models. *IEEE Computer Graphics and Applications 28*, 1 (Jan.-Febr. 2008), 44–55. URL: http://vcg.isti.cnr.it/Publications/2008/CPCS08. See pages 54, 172, and 182.

Cro06. CROCKFORD D.: Rfc4627: Javascript object notation, 2006. See page 81.

CRS98. CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174. See page 122.

CSAD04. COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Transactions on Graphics, Proceedings of the 2004 SIGGRAPH 23*, 3 (2004), 905 – 914. doi:10.1145/1015706.1015817. See page 40.

CSVP07. CIUALA A., SPENCE P., VIEIRA J. M., POUPEAU G.: Expressing complex associations in medieval historical documents: The Henry III fine rolls project. In *Digital Humanities 2007. Conference abstracts* (2007). See page 185.

CTSO03. CHEN D.-Y., TIAN X.-P., SHEN Y.-T., OUHYOUNG M.: On Visual Similarity Based 3D Model Retrieval. *Computer Graphics Forum 22*, 3 (2003), 223–232. See page 42.

CWQ*04. CHENG K.-S. D., WANG W., QIN H., WONG K.-Y. K., YANG H., LIU Y.: Fitting Subdivision Surfaces to Unorganized Point Data using SDM. *Proceedings of 12th Pacific Conference on Computer Graphics and Applications 1* (2004), 16 – 24. doi:10.1109/*PCCGA*.2004.1348330. See page 39.

CWQ*07. CHENG K.-S., WANG W., QIN H., WONG K.-Y. K., YANG H., LIU Y.: Design and Analysis of Optimization Methods for Subdivision Surface Fitting. *IEEE Transactions on Visualization and Computer Graphics 13*, 5 (2007), 878–890. doi:10.1109/*TVCG*.2007.1064. See page 39.

DAA*11. DEFANTI T. A., ACEVEDO D., AINSWORTH R. A., BROWN M. D., CUTCHIN S., DAWE G., DOERR K.-U., JOHNSON A., KNOX C., KOOIMA R., KUESTER F., LEIGH J., LONG L., OTTO P., PETROVIC V., PONTO K., PRUDHOMME A., RAO R., RENAMBOT L., SANDIN D. J., SCHULZE J. P., SMARR L., SRINIVASAN M., WEBER P., WICKHAM G.: The future of the CAVE. *Central European Journal of Engineering 1* (2011), 16–37. doi:10.2478/*s*13531 − 010 − 0002 − 5. See page 99.

DDSD03. DÉCORET X., DURAND F., SILLION F., DORSEY J.: Billboard Clouds for Extreme Model Simplification. *Proceedings of the ACM Siggraph 2003 22*, 3 (2003), 689–696. doi:10.1145/882262.882326. See page 40.

Del04. DELIGIANNIDIS L.: Precise Navigation in Virtual Environments without Compromising Interaction. *Proceedings of the International Conference on Imaging, Science, and Technology 3* (2004), 272–278. See page 131.

DKT98. DEROSE T., KASS M., TRUONG T.: Subdivision Surfaces in Character Animation. In *Proceedings of SIGGRAPH 98* (1998), pp. 85–94. See page 10.

DL01. DIETZ P., LEIGH D.: DiamondTouch: a multi-user touch technology. *Proceedings of the 14th annual ACM symposium on User interface software and technology 14* (2001), 219 – 226. doi:10.1145/502348.502389. See page 150.

Doe05. DOERR M.: The CIDOC CRM, a standard for the integration of cultural information. (online), Nuremberg, 2005. Tutorial slides. URL: `http://cidoc.ics.forth.gr/docs/crm_for_nurnmberg.ppt`. See page 35.

DSW00. DONCHIN E., SPENCER K. M., WIJESINGHE R.: The mental prosthesis: assessing the speed of a p300-based brain-computer interface. *IEEE Transactions on Rehabilitation Engineering 8*, 2 (2000), 174–179. See page 142.

dub04. Dublin core metadata element set, version 1.1. ISO ANSI, 2004. ISO/IEC 15836:2002, available from dublincore.org. See page 35.

Dyk93. DYKSTRA P.: X11 in virtual environments. *Proceedings of IEEE 1993 Symposium on Research Frontiers in Virtual Reality 9* (1993), 118–119. doi:10.1109/*VRAIS*.1993.378255. See page 150.

Eng06. ENGEL K. D.: *Real-time volume graphics*. Peters, Wellesley, Mass., 2006. See page 4.

Epo04. Epoch – the european network of excellence on ict applications to cultural heritage, 2004. IST-2002-507382. See page 164.

FB81. FISCHLER M. A., BOLLES R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (1981), 381–395. doi:10.1145/358669.358692. See page 40.

FBBS06. FLECK S., BUSCH F., BIBER P., STRABER W.: 3d surveillance a distributed network of smart cameras for real-time tracking and its visualization in 3d. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop* (Washington, DC, USA, 2006), IEEE Computer Society, p. 118. doi:10.1109/*CVPRW*.2006.6. See page 189.

Fel01. FELLNER D. W.: Graphics content in digital libraries: Old problems, recent solutions, future demands. *Journal of Universal Computer Science 7* (2001), 400 – 409. doi:10.3217/*jucs* − 007 − 05 − 0400. See page 39.

FGM*99. FIELDING R., GETTYS J., MOGUL J., FRYSTYK H., MASINTER L., LEACH P., BERNERS-LEE T.: Hypertext Transfer Protocol – HTTP/1.1. RFC Editor, 1999. See page 84.

FH05. FELLNER D. W., HAVEMANN S.: Striving for an adequate vocabulary: Next generation metadata. *Proc. 29th Annual Conf. of the German Classification Society 29* (2005), 13 – 20. See page 12.

FHH03a. FARIN G., HAMANN B., HAGEN H.: *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer, 2003. See page 122.

FHH03b. FELLNER D. W., HAVEMANN S., HOPP A.: Dave – eine neue technologie zur preiswerten und hochqualitativen immersiven 3d-darstellung. In *Proc. 8. Workshop: Sichtsysteme – Visualisierung in der Simulationstechnik* (Bremen, nov 2003), Möller R., (Ed.), Shaker Verlag, pp. 77–87. See page 99.

FHH07. FELLNER D. W., HAVEMANN S., HOPP A.: A Single Chip DLP Projector for stereoscopic images of high color quality and resolution. In *Proc. IPT-EGVE 2007: 13th Eurographics Symposium on Virtual Environments, 10th Immersive Projection Technology Workshop* (Weimar, Germany, July 2007). See pages 39 and 99.

Fie00. FIELDING R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Univeristy of California, Irvine, 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. See pages 81 and 82.

Fis02. FISHER R. B.: Applying knowledge to reverse engineering problems. *Proceedings of Geometric Modeling and Processing 1* (2002), 149–155. doi:10.1109/*GMAP*.2002.1027506. See page 43.

FKC*05. FELS S., KINOSHITA Y., CHEN T.-P. G., TAKAMA Y., YOHANAN S., TAKAHASHI S., GADD A., FUNAHASHI K.: Swimming Across the Pacific: A VR Swimming Interface. *IEEE Computer Graphics and Applications 25*, 1 (2005), 24– 31. doi:10.1109/*MCG*.2005.20. See page 132.

FLL*93. FAIRCHILD K. M., LEE B. H., LOO J., NG H., SERRA L.: The heaven and earth virtual reality: Designing applications fornovice users. *Proceedings of 1993 IEEE Virtual Reality Annual International Symposium 1* (1993), 47–53. doi:10.1109/*VRAIS*.1993.380799. See pages 101 and 132.

FRE03. FERNANDES K. J., RAJA V., EYRE J.: Cybersphere: the fully immersive spherical projection system. *Communications of the ACM 46*, 9 (2003), 141 – 146. doi:10.1145/903893.903929. See page 132.

FSG98. FUHRMANN A., SCHMALSTIEG D., GERVAUTZ M.: Strolling through Cyberspace with Your Hands in Your Pockets: Head Directed Navigation in Virtual Environments. *Proceedings of the 4th EUROGRAPHICS Workshop on Virtual Environments 4* (1998), 216–227. doi:10.1007/978 – 3 – 7091 – 7519 – $4_2$1. See page 131.

GBHF05. GERTH B., BERNDT R., HAVEMANN S., FELLNER D. W.: 3D Modeling for Non-Expert Users with the Castle Construction Kit v0.5. In *Proc. VAST 2005* (Nov 2005), Mudge, Ryan,, Scopigno, (Eds.), Int. Symp. on VR, Arch. and Intelligent Cult. Heritage, Eurographics / ACM Siggraph, Eurographics Press, pp. 49–57. See pages 59 and 77.

GBY91. GONNET G. H., BAEZA-YATES R.: *Handbook of Algorithms and Data Structures*. Addison-Wesley Publishing, 1991. See page 122.

GG04. GELFAND N., GUIBAS L. J.: Shape Segmentation Using Local Slippage Analysis. *Proceedings of Symposium on Geometry Processing 1* (2004), 219–228. doi:10.2312/*SGP*/*SGP*04/219 – 228. See page 40.

GH97. GARLAND M., HECKBERT P.: Surface Simplification Using Quadric Error Metrics. *Proceedings of 1997 ACM Siggraph 31* (1997), 209 – 216. doi:10.1145/258734.258849. See page 113.

GHM*03. GUDGIN M., HADLEY M., MENDELSOHN N., MOREAU J.-J., NIELSEN H. F.: Soap version 1.2 part 1: Messaging framework. W3C Recommendation, June 2003. URL: http://www.w3.org/TR/soap12-part1/. See page 81.

GJ02. GIESEN J., JOHN M.: Surface reconstruction based on a dynamical system. *Computer Graphics Forum 21* (2002), 363–371. doi:10.1111/1467 – 8659.00596. See page 39.

GK07. GANSTER B., KLEIN R.: An Integrated Framework for Procedural Modeling. *Proceedings of Spring Conference on Computer Graphics 2007 (SCCG 2007) 23* (2007), 150–157. URL: http://cg.cs.uni-bonn.de/publications/publication.asp?id=293&language=de. See page 26.

GKM86. GETTYS J., KARLTON P. L., MCGREGOR S.: The X Window System. *ACM Transactions on Graphics 5* (1986), 79–109. doi:10.1145/22949.24053. See page 150.

GMHP04. GROCHOW K., MARTIN S. L., HERTZMANN A., POPOVIĆ Z.: Style-based inverse kinematics. *Proceedings of the 2004 SIGGRAPH Conference, ACM Transactions on Graphics 23*, 3 (2004), 522 – 531. doi:10.1145/1015706.1015755. See page 132.

Gro03. GROUP C. C. S. I.: *Definition of the CIDOC Conceptual Reference Model*. ICOM/CIDOC Documentation Standards Group, 2003. See page 35.

Gru93. GRUBER T. R.: A translation approach to portable ontology specifications. *Knowledge Acquisition - Special issue: Current issues in knowledge modeling 5*, 2 (1993), 199–220. doi:10.1006/*knac*.1993.1008. See page 32.

GSN*01. GUGER C., SCHLGL A., NEUPER C., WALTERSPACHER D., STREIN T., PFURTSCHELLER G.: Rapid prototyping of an eeg-based brain-computer interface (bci). *IEEE Transactions on Neural Systems and Rehabilitation Engineering 9* (2001), 49–58. See page 142.

GWM01. GUMHOLD S., WANG X., MACLEOD R.: Feature Extraction from Point Clouds. *Proceedings of 10th Int. Meshing Roundtable 1* (2001), 1–13. See page 39.

Han05. HAN J. Y.: Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2005), ACM, pp. 115–118. doi:10.1145/1095034.1095054. See pages 97 and 130.

Hav05. HAVEMANN S.: *PhD-Thesis: Generative Mesh Modeling*. Technische Universität Braunschweig, 2005. URL: http://www.digibib.tu-bs.de/?docid=00000008. See pages 11, 26, 57, and 175.

HD07. HAGEDORN B., DÖLLNER J.: High-level web service for 3D building information visualization and analysis. *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems 15* (2007), 8:1–8. doi:10.1145/1341012.1341023. See page 90.

HDD*92. HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *Proceedings of the 19th annual conference on Computer graphics and interactive techniques 1* (1992), 71 – 78. doi:10.1145/142920.134011. See pages 25 and 39.

HF01. HAVEMANN S., FELLNER D. W.: A versatile 3D model representation for cultural reconstruction. *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage 1* (2001), 205 – 212. doi:10.1145/584993.585025. See page 12.

HF03. HAVEMANN S., FELLNER D. W.: Generative Mesh Modeling. *Technical Report 1* (2003), 1–11. URL: http://www.cgv.tugraz.at/CGVold/DigitalLibrary/publications/TechnicalReports/bs/TR-tubs-cg-2003-01.pdf. See page 39.

HF04. HAVEMANN S., FELLNER D. W.: Generative parametric design of gothic window tracery. *Proc. of VAST 2004 Intl. Symp. 1* (2004), 193 – 201. doi:10.1109/*SMI*.2004.37. See pages 12 and 29.

HF07. HAVEMANN S., FELLNER D. W.: Seven research challenges of generalized 3D documents. *IEEE Computer Graphics and Applications 27*, 3 (2007), 70–76. (special issue on 3D documents). doi:*http*://*dx.doi.org*/10.1109/*MCG*.2007.67. See pages 58 and 185.

HKD93. HALSTEAD H., KASS M., DEROSE T.: Efficient, Fair Interpolation using Catmull–Clark Surfaces. In *Proceedings of COMPUTER GRAPHICS 1993* (1993), pp. 35–44. See page 10.

HOP*05. HOFER M., ODEHNAL B., POTTMANN H., STEINER T., WALLNER J.: 3D shape recognition and reconstruction based on line element geometry. *Proceedings of the Tenth IEEE International Conference on Computer Vision 2* (2005), 1532–1538. doi:10.1109/*ICCV*.2005.2. See page 40.

HP02. HEGE H.-C., POLTHIER K.: *Mathematical Visualization: Algorithms, Applications and Numerics*. Springer, 2002. See page 122.

HSB*08. HAVEMANN S., SETTGAST V., BERND R., EIDE O., FELLNER D. W.: The arrigo showcase reloaded - towards a sustainable link between 3d and semantics. *Proceedings of the Symposium on Virtual Reality, Archaeology and Cultural Heritage 1* (2008), 125–132. doi:10.1145/1551676.1551680. See pages 67 and 69.

HSKF06. HAVEMANN S., SETTGAST V., KROTTMAIER H., FELLNER D.: On the integration of 3D models into digital cultural heritage libraries. In *VAST 2006 Project Papers* (Nicosia, Cyprus, Nov 2006), Ioannides, Arnold, Niccolucci,, Mania, (Eds.), Int. Symp. on VR, Archae. and Cultural Heritage, Epoch publication, pp. 161–169. See pages 173, 185, and 200.

HSKK01. HILAGA M., SHINAGAWA Y., KOHMURA T., KUNII T. L.: Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques 28* (2001), 203–212. See page 42.

HSLF07. HAVEMANN S., SETTGAST V., LANCELLE M., FELLNER D.: 3d-powerpoint - towards a design tool for digital exhibitions of cultural artifacts. In *VAST 2007* (Brighton, UK, Nov 2007), Arnold, Niccolucci,, Chalmers, (Eds.), 8th Inter. Symp. on Virtual Reality, Archaeology and Cultural Heritage, Eurographics Association, pp. 39–46. See pages 39, 64, 131, 185, 189, and 201.

HT07. HUTTERER P., THOMAS B. H.: Groupware Support in the Windowing System. *Proceedings of the eight Australasian conference on User interface 64* (2007), 39–46. URL: http://wearables.unisa.edu.au/mpx. See page 150.

Hut06. HUTTERER P.: The Multi-Pointer X Server. online: http://wearables.unisa.edu.au/mpx, 2006. See page 150.

HZ04. HARTLEY R., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. ISBN 0521540518. URL: http://www.bookpool.com/sm/0521540518. See page 44.

IG03. ISENBURG M., GUMHOLD S.: Out-of-core compression for gigantic polygon meshes. *ACM Transactions on Graphics 22*, 3 (July 2003), 935–942. See page 6.

Ini95. INITIATIVE D. C. M.: Dublin Core Metadata Initiative. http://dublincore.org/, 1995. URL: http://dublincore.org/. See page 35.

ISO05. ISO 19005-1:2005: *Document management – Electronic document file format for long-term preservation – Part 1: Use of PDF 1.4 (PDF/A-1)*. ISO, Geneva, Switzerland, 2005. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38920. See page 69.

ISO08. ISO 32000-1:2008: *Document management – Portable document format – Part 1: PDF 1.7*. ISO, Geneva, Switzerland, 2008. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502. See page 69.

IYFN05. IWATA H., YANO H., FUKUSHIMA H., NOMA H.: CirculaFloor. *IEEE Computer Graphics and Applications 25*, 1 (2005), 64 – 67. doi:10.1109/*MCG*.2005.5. See page 132.

Jas58. JASPER H. H.: The ten-twenty electrode system of the international federation. *Electroencephalography and Clinical Neurophysiology 10* (1958), 370–375. See page 144.

JGD02. JUNG T., GROSS M. D., DO E. Y.-L.: Annotating and Sketching on 3D Web models. *Proceedings of the International Conference on Intelligent User Interfaces 7* (2002), 95–102. URL: http://portal.acm.org/citation.cfm?id=502733. See page 32.

JKGA06. JORDÀ S., KALTENBRUNNER M., GEIGER G., ALONSO M.: The reacTable – A Tangible Tabletop Musical Instrument and Collaborative Workbench. *International Conference on Computer Graphics and Interactive Techniques archive ACM SIGGRAPH 2006 Sketches 25* (2006), 91. See page 150.

JKS11. JUNG T., KROHN S., SCHMIDT P.: Ein Natural User Interface zur Interaktion in einem CAVE Automatic Virtual Environment basierend auf optischem Tracking. In *Workshop 3D-NordOst* (2011), pp. 93–102. See page 132.

JLRLC05. JACOBSON J., LE RENARD M., LUGRIN J.-L., CAVAZZA M.: The caveut system: immersive entertainment based on a game engine. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology* (New York, NY, USA, 2005), ACE '05, ACM, pp. 184–187. doi:10.1145/1178477.1178503. See page 94.

Jos07. JOSUTTIS N. M.: *SOA in Practice: The Art of Distributed System Design*. O'Reilly, Beijing, 2007. See page 81.

JSB10. JUAREZ A., SCHONENBERG B., BARTNECK C.: Implementing a low-cost cave system using the cryengine2. *Entertainment Computing 1*, 3-4 (2010), 157–164. See page 93.

Kad05. KADOBAYASHI R.: Automatic 3d blogging to support the collaborative experience of 3d digital archives. In *ICADL* (2005), Fox E. A., Neuhold E. J., Premsmit P.,, Wuwongse V., (Eds.), vol. 3815 of *Lecture Notes in Computer Science*, Springer, pp. 109–118. See page 69.

Kap10. KAPLANYAN A.: Real-time Diffuse Global Illuminationin CryENGINE 3. In *ACM SIGGRAPH 2010 Global Illumination across Industries Course* (Los Angeles, CA, 2010). See pages 93 and 117.

KBBC05. KALTENBRUNNER M., BOVERMANN T., BENCINA R., CONSTANZA E.: TUIO: A Protocol for Table-Top Tangible User Interfaces. *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation 6* (2005), 1–5. See page 150.

Ken. KENNER C.: Glove programmable input emulator (glovepie). URL: http://carl.kenner.googlepages.com/glovepie. See page 176.

KLS96. KLEIN R., LIEBICH G., STRASSER W.: Mesh Reduction with Error Control. *Proceedings of IEEE Visualization '96 1* (1996), 311–318. See page 122.

KSKP03. KRAUSZ G., SCHERER R., KORISEK G., PFURTSCHELLER G.: Critical decision-speed and information transfer in the "graz brain-computer interface". *Appl. Psychophysiol Biofeedback 28* (2003), 233–240. See page 146.

Lan11. LANCELLE M.: *Visual Computing in Virtual Environments*. PhD thesis, TU Graz, Diss., 2011, 2011. 228 p. See page 97.

Lee09. LEEB R.: *Brain computer communication: the motivation, aim, and impact of virtual feedback*. PhD thesis, 2009. See page 149.

LF11. LANCELLE M., FELLNER D. W.: Soft edge and soft corner blending. In *Workshop Virtuelle & Erweiterte Realitt (VR/AR)* (2011), pp. 63–71. See page 98.

LFKZ01. LAVIOLA J. J. J., FELIZ D. A., KEEFE D. F., ZELEZNIK R. C.: Hands-free multi-scale navigation in virtual environments. *Proceedings of the 2001 Symposium on Interactive 3D Graphics 1* (2001), 9–15. doi:10.1145/364338.364339. See page 131.

LFMP*07. LEEB R., FRIEDMAN D., MLLER-PUTZ G., SCHERER R., SLATER M., PFURTSCHELLER G.: Self-paced (asynchronous) bci control of a wheelchair in virtual environments: A case study with a tetraplegic. *Computational Intelligence and Neuroscience, special issue: "Brain-Computer Interfaces: Towards Practical Implementations and Potential Applications"* (2007), 1–8. See page 142.

LH92. LEVKOWITZ H., HERMAN G. T.: Color scales for image data. *IEEE Computer Graphics and Applications 12*, 1 (1992), 72–80. doi:*http://doi.ieeecomputersociety.org*/10.1109/38.135886. See page 122.

LH94. LOUGHLIN M. M., HUGHES J. F.: An annotation system for 3D fluid flow visualization. *Proceedings of the IEEE Conference on Visualization 5* (94), 273 – 279. URL: http://portal.acm.org/citation.cfm?id=951137. See page 33.

LLK*07. LEEB R., LEE F., KEINRATH C., SCHERER R., BISCHOF H., PFURTSCHELLER G.: Brain-computer communication: Motivation, aim and impact of exploring a virtual apartment. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 15* (2007), 473–482. See page 145.

LN03. LEE S. C., NEVATIA R.: Interactive 3D Building Modeling Using a Hierarchical Representation. *Proceedings of IEEE Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis 1* (2003), 58– 65. doi:10.1109/*HLK*.2003.1240859. See page 26.

Lon06. LONDON CHARTER INITIATIVE (HUGH DENARD): The london charter, June 2006. URL: http://www.londoncharter.org. See page 185.

LPZ03. LEOPOLDSEDER S., POTTMANN H., ZHAO H.: The d^2-tree: A hierarchical representation of the squared distance function. *Tech.Rep., Institut of Geometry, Vienna University of Technology 101* (2003), 1–12. See page 44.

LSF08. LANCELLE M., SETTGAST V., FELLNER D. W.: Definitely affordable virtual environment. *Virtual Reality Conference 1*, 2 (2008), 1–1. See page 103.

LSFP07. LEEB R., SETTGAST V., FELLNER D., PFURTSCHELLER G.: Self-paced exploration of the Austrian National Library through thought. In *International Journal of Bioelectromagnetism* (2007), vol. 9. See page 141.

Ma05. MA W.: Subdivision surfaces for CAD - an overview. *Computer-Aided Design 37*, 7 (2005), 693–709. URL: http://www.sciencedirect.com/science/journal/00104485. See page 8.

MB00. MASON S. G., BIRCH G. E.: A brain-controlled switch for asynchronous control applications. *IEEE Transactions on Biomedical Engineering 47* (2000), 1297–1307. See page 142.

MB06. MÜLLER K. R., BLANKERTZ B.: Toward noninvasive brain computer interfaces. *IEEE Signal Processing Magazine 23* (2006), 125–128. See page 148.

MFR*10. MÜLLER K., FÜNFZIG C., REUSCHE L., HANSFORD D., FARIN G. E., HAGEN H.: Dinus: Double insertion, nonuniform, stationary subdivision surfaces. *ACM Trans. Graph. 29*, 3 (2010). doi:10.1145/1805964.1805969. See page 10.

MH00. MÜLLER K., HAVEMANN S.: Subdivision Surface Tesselation on the Fly using a versatile Mesh Data Structure. In *Proceedings of Eurographics 2000* (2000), vol. 19, pp. 151–159. See page 9.

Mic10. MICROSOFT: Direct3d 11 graphics, 2010. URL: http://msdn.microsoft.com/en-us/library/ee415158(VS.85).aspx. See page 75.

Mil96. MILLER P.: Metadata for the masses. Ariadne 5 (online), 1996. URL: http://www.ariadne.ac.uk/issue5/metadata-masses. See page 35.

Min95. MINE M. R.: Virtual Environment Interaction Techniques. *University of North Carolina at Chapel Hill, Technical Report TR95-018 18* (1995), 1–18. See page 131.

MM03. MILLAN J., MOURINO J.: Asynchronous bci and local neural classifiers: an overview of the adaptive brain interface project. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 11*, 2 (June 2003), 159–161. See page 148.

MMTP04. MA W., MA X., TSO S.-K., PAN Z.: A direct approach for subdivision surface fitting from a dense triangle mesh. *Computer-Aided Design 36*, 6 (2004), 525–536. See page 39.

MMY06. MARONNA R. A., MARTIN D. R., YOHAI V. J.: *Robust Statistics: Theory and Methods*. Wiley Series in Probability and Statistics. John Wiley and Sons, 2006. See page 46.

Moo91. MOORE A. W.: An introductory tutorial on kd-trees. *Technical Report, Computer Laboratory, University of Cambridge 209* (1991), 1–20. See page 122.

MPB05. MARVIE J.-E., PERRET J., BOUATOUCH K.: The FL-system: a functional L-system for procedural geometric modeling. *The Visual Computer 21* (2005), 329–339. doi:10.1007/$s00371-005-0289-z$. See page 26.

MPSBP05. MÜLLER-PUTZ G. R., SCHERER R., BRAUNEIS C., PFURTSCHELLER G.: Steady-state visual evoked potential (ssvep)-based communication: impact of harmonic frequency components. *Journal of Neural Engineering 2* (2005), 123–130. See page 142.

MPSP07. MÜLLER-PUTZ G., SCHERER R., PFURTSCHELLER G.: Control of a two-axis artificial limb by means of a pulse width modulated. *Challenges for assistive Technology - AAATE '07 (European Conference for the Advancement of Assistive Technology)* (2007), 888–892. See pages 145 and 147.

MPSPR05. MÜLLER-PUTZ G. R., SCHERER R., PFURTSCHELLER G., RUPP R.: Eeg-based neuroprosthesis control: a step towards clinical practice. *Neuroscience Letters 382* (2005), 169–174. See page 148.

MSF07. MARINI S., SPAGNUOLO M., FALCIDIENO B.: Structural Shape Prototypes for Automatic Classification of 3D Objects. *IEEE Computer Graphics and Applications 27*, 4 (2007), 28–37. See page 42.

MSHS06. MARTINET A., SOLER C., HOLZSCHUCH N., SILLION F.: Accurate Detection of Symmetries in 3D Shapes. *ACM Transactions on Graphics 25* (2006), 439 – 464. doi:10.1145/1138450.1138462. See page 40.

MVUVG06. MUELLER P., VEREENOOGHE T., ULMER A., VAN GOOL L.: Automatic Reconstruction of Roman Housing Architecture. *Recording, Modeling and Visualization of Cultural Heritage 1* (2006), 287–298. URL: http://www.vision.ee.ethz.ch/publications/get_abstract.cgi?procs=387&mode=&lang=en. See page 43.

MWH*06. MÜLLER P., WONKA P., HAEGLER S., A. U., VAN GOOL L.: Procedural Modeling of Buildings. *Proceedings of ACM SIGGRAPH 2006 25*, 3 (2006), 614–623. doi:10.1145/1141911.1141931. See pages 11 and 26.

MZWVG07. MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based Procedural Modeling of Facades. *ACM Transactions on Graphics 28*, 3 (2007), 1–9. See page 40.

ND06. NICCOLUCCI F., D'ANDREA A.: An Ontology for 3D Cultural Objects. In *Int. Symp. on VR, Archaeology and Intelligent Cultural Heritage* (Nicosia, Cyprus, 2006), Ioannides, Arnold, Niccolucci,, Mania, (Eds.), Eurographics Association, pp. 203–210. doi:10.2312/*VAST*/*VAST*06/203 – 210. See pages 37, 57, and 185.

NLMD12. NIESSNER M., LOOP C., MEYER M., DEROSE T.: Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces. *ACM Trans. Graph. 31*, 1 (Feb. 2012), 6:1–6:11. doi:10.1145/2077341.2077347. See page 11.

NMKT05. NAKASHIMA K., MACHIDA T., KIYOKAWA K., TAKEMURA H.: A 2D-3D integrated environment for cooperative work. *Proceedings of the ACM symposium on Virtual Reality Software and Technology 12* (2005), 16–22. doi:10.1145/1101616.1101621. See page 150.

NSS*06. NI T., SCHMIDT G. S., STAADT O. G., LIVINGSTON M. A., BALL R., MAY R.: A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. *Proceedings of the IEEE Conference on Virtual Reality 14* (2006), 223–236. doi:10.1109/*VR*.2006.20. See page 150.

NYH*03. NEUMANN U., YOU S., HU J., JIANG B., LEE J.: Augmented virtual environments (ave): Dynamic fusion of imagery and 3d models. In *VR '03: Proceedings of the IEEE Virtual Reality 2003* (Washington, DC, USA, 2003), IEEE Computer Society, p. 61. See page 189.

NYN03. NIEH J., YANG J. S., NOVIK N.: Measuring Thin-Client Performance Using Slow-Motion Benchmarking. *ACM Transactions on Computer Systems 21*, 1 (2003), 87 – 115. doi:10.1145/592637.592640. See page 151.

Obj. OBJECT MANAGEMENT GROUP: Common Object Request Broker Architecture: Core Specification. URL: http://www.omg.org/technology/documents/corba_spec_catalog.htm. See page 81.

OF08. OFFEN L., FELLNER D. W.: BioBrowser – Visualization of and Access to Macro-Molecular Structures. *Mathematics and Visualization – Visualization in Medicine and Life Sciences 5* (2008), 257–273. doi:10.1007/978 – 3 – 540 – 72630 – $2_1$5. See page 33.

OMS06. OSMAN K., MALRIC F., SHIRMOHAMMADI S.: A 3D Annotation Interface Using DIVINE Visual Display. *Proceeding of the IEEE International Workshop on Haptic Audio Visual Environments and their Applications 5* (2006), 5–9. doi:10.1109/*HAVE*.2006.283798. See page 32.

Ous98. OUSTERHOUT J. K.: Scripting: Higher Level Programming for the 21st Century. *IEEE Computer Magazine 31*, 3 (1998), 23–30. See pages 77 and 189.

Pet04. PETERNELL M.: Developable Surface Fitting to Point Clouds. *Computer Aided Geometric Design 21* (2004), 785–803. doi:10.1016/*j.cagd*.2004.07.008. See page 40.

PFC*05. PINGI P., FASANO A., CIGNONI P., MONTANI C., SCOPIGNO R.: Exploiting the scanning sequence for automatic registration of large sets of range maps. *Computer Graphics Forum 24*, 3 (2005), 517–526. URL: http://vcg.isti.cnr.it/Publications/2005/PFCMS05/. See page 18.

PH03. POTTMANN H., HOFER M.: Geometry of the squared distance function to curves and surfaces. *H.C. Hege and K. Polthier, editors, Visualization and Mathematics III 3* (2003), 223–244. See page 44.

Pin03. PINZARI G. F.: Introduction to NX technology. *NoMachine Technical Report 309* (2003), 1–7. URL: http://www.nomachine.com/. See page 150.

PKG03. PAULY M., KEISER R., GROSS M.: Multi-scale Feature Extraction on Point-Sampled Surfaces. *Computer Graphics Forum 22* (2003), 281–289. doi:10.1111/1467 – 8659.00675. See page 40.

PLM09. PENG H., LONG F., MYERS E. W.: VANO: a volume-object image annotation system. *Bioinformatics 25*, 5 (2009), 695–697. doi:10.1093/*bioinformatics*/*btp*046. See page 33.

PM01. PARISH Y., MUELLER P.: Procedural Modeling of Cities. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques 28* (2001), 301–308. doi:10.1145/383259.383292. See page 26.

PMPS*07. PFURTSCHELLER G., MLLER-PUTZ G., SCHLGL A., GRAIMANN B., SCHERER R., LEEB R., BRUNNER C., KEINRATH C., TOWNSEND G., VIDAURRE C., NAEEM M., LEE F., WRIESSNEGGER S., ZIMMERMANN D., HFLER E., NEUPER C.: *Toward Brain-Computer Interfacing*. MIT Press, 2007. See pages 143, 144, and 145.

PN01. PFURTSCHELLER G., NEUPER C.: Motor imagery and direct brain-computer communication. *Proceedings of the IEEE 89* (2001), 1123–1134. See page 142.

PNB05. PFURTSCHELLER G., NEUPER C., BIRBAUMER N.: Human brain-computer interface. *Motor cortex in voluntary movements: a distributed system for distributed functions. Series: Methods and New Frontiers in Neuroscience* (2005), 367–401. See page 142.

PPV95. PAOLUZZI A., PASCUCCI V., VICENTINO M.: Geometric programming: a programming approach to geometric design. *ACM Transactions on Graphics 14* (1995), 266 – 306. doi:10.1145/212332.212349. See page 26.

Pra04. PRATT M.: Extension of iso 10303, the step standard, for the exchange of procedural shape models. In *Proc. International Conference on Shape Modeling and Applications (SMI04)* (Genova, Italy, June 2004), pp. 317–326. See page 162.

PSH12. PAN X., SCHIFFER T., HECHER M.: A Scalable Repository Infrastructure for CH Digital Object Management. In *18th International Conference of Virtual Systems and Multimedia* (2012). See page 38.

PTK05. PAL P., TIGGA A. M., KUMAR A.: Feature extraction from large CAD databases using genetic algorithm. *Computer-Aided Design 37*, 5 (2005), 545–558. doi:10.1016/*j.cad*.2004.08.002. See page 40.

PWL01. POTTMANN H., WALLNER J., LEOPOLDSEDER S.: Kinematical methods for the classification, reconstruction and inspection of surfaces. *Comptes rendus du Congrs national de mathmatiques appliques et industrielles 1* (2001), 51– 60. See page 40.

PZL08. PAUTASSO C., ZIMMERMANN O., LEYMANN F.: Restful web services vs. "big"' web services: making the right architectural decision. *Proceeding of the 17th international conference on World Wide Web 17* (2008), 805–814. doi:10.1145/1367497.1367606. See page 90.

RA99. RAMAMOORTHI R., ARVO J.: Creating Generative Models from Range Images. *Proceedings of SIGGRAPH 99 1* (1999), 195–204. doi:10.1145/311535.311557. See page 41.

RB08. ROTH P. M., BISCHOF H.: *Conservative Learning for Object Detectors*. Springer, 2008, ch. Conservative Learning for Object Detectors, pp. 139–158. See page 189.

RBW01. REGENBRECHT H., BARATOFF G., WAGNER M.: A tangible AR desktop environment. *Computers & Graphics 25*, 5 (2001), 755–763. doi:*doi* : 10.1016/*S*0097 – 8493(01)00118 – 2. See page 150.

RLKG*09. ROST R. J., LICEA-KANE B., GINSBURG D., KESSENICH J. M., LICHTENBELT B., MALAN H., WEIBLEN M.: *OpenGL Shading Language*, 3rd ed. Addison-Wesley Professional, 2009. See page 117.

Rog01. ROGERS D. F.: *An introduction to NURBS: with historical perspective*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. See page 8.

Ros97. ROSSIGNAC J.: The 3D Revolution: CAD Access for All! In *SMA '97: Proc. International Conference on Shape Modeling and Applications (SMA '97)* (Washington, DC, USA, 1997), IEEE Computer Society, p. 64. See page 55.

RSC87. REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM, pp. 283–291. doi:10.1145/37401.37435. See page 189.

RSFWH98. RICHARDSON T., STAFFORD-FRASER Q., WOOD K. R., HOPPER A.: Virtual network computing. *IEEE Internet Computing 2* (1998), 33–38. doi:10.1.1.112.4716. See page 150.

RTB96. ROGOWITZ B. E., TREINISH L. A., BRYSON S.: How NOT to lie with visualization. *Computers in Physics 10* (1996), 268 – 273. URL: http://www.research.ibm.com/dx/proceedings/pravda/truevis.htm. See page 122.

RVB02. REINERS D., VOSS G., BEHR J.: OpenSG - basic concepts. In *Proceedings of OpenSG Symposium 2002* (2002). See pages 76 and 151.

RvdH04. RABBANI T., VAN DEN HEUVEL F.: Methods for Fitting CSG Models to Point Clouds and their Comparison. *Proceedings of Computer Graphics and Imaging - 2004 1* (2004), 1–6. URL: http://www.actapress.com/PaperInfo.aspx?PaperID=16940. See page 39.

SAK*02. SAWHNEY H. S., ARPA A., KUMAR R., SAMARASEKERA S., AGGARWAL M., HSU S., NISTER D., HANNA K.: Video flashlights: real time rendering of multiple videos for immersive model visualization. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 157–168. See page 189.

SB04. SHREINER D., BOARD O. A. R.: *OpenGL reference manual : the official reference document to OpenGL, version 1.4*. Addison-Wesley, 2004. See page 75.

SBS*09. STROBL M., BERNDT R., SETTGAST V., HAVEMANN S., FELLNER D. W.: Publishing 3D Content as PDF in Cultural Heritage. *Proceedings of the 10th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage (VAST) 6* (2009), 117–124. doi:10.2312/*VAST*/*VAST*09/117 – 124. See pages 16, 200, and 201.

SBS*10. SCHIEFER A., BERNDT R., SETTGAST V., ULLRICH T., FELLNER D. W.: Service-oriented scene graph manipulation. In *Proceedings Web3D 2010, Proceedings of the 15th International Conference on Web 3D Technology* (2010), pp. 55–62. doi:10.1145/1836049.1836057. See page 200.

SBW02. SUGIMOTO S., BAKER T., WEIBEL S. L.: Dublin Core: Process and Principles. *Lecture Notes in Computer Science – Digital Libraries: People, Knowledge, and Technology 2555/2010* (2002), 25–35. doi:10.1007/3 – 540 – 36227 – 4$_3$. See page 34.

SEF12. SETTGAST V., EVA E., FELLNER D.: The preparation of 3d-content for interactive visualization. In *9. Fachtagung "Digitales Engineering zum Planen, Testen und Betreiben technischer Systeme" (15. Fraunhofer IFF-Wissenschaftstage)* (6 2012), Fraunhofer IFF, pp. 185–192. See page 199.

SFWV07. SHEN C., FORLINES C., WIGDOR D., VERNIER F.: Open Letter to OS Designers from the Tabletop Research Community. online: http://www.diamondspace.merl.com/papers/2007_open_letter_to_OS_developers.pdf, 2007. See page 106.

Sha98. SHAKARJI C. M.: Least-Squares Fitting Algorithms of the NIST Algorithm Testing System. *Journal of Research of the National Institute of Standards and Technology 106*, 6 (1998), 633–641. See page 40.

SHYN03. SEBE I. O., HU J., YOU S., NEUMANN U.: 3d video surveillance with augmented virtual environments. In *IWVS '03: First ACM SIGMM international workshop on Video surveil-*

lance (New York, NY, USA, 2003), ACM, pp. 107–112. doi:10.1145/982452.982466. See page 189.

SK92. SNYDER J. M., KAJIYA J. T.: Generative modeling: a symbolic system for geometric modeling. *Proceedings of SIGGRAPH 1992 1* (1992), 369–378. doi:10.1145/142920.134094. See pages 11 and 26.

SK98. SHINAGAWA Y., KUNII T. L.: Unconstrained Automatic Image Matching Using Multiresolutional Critical-Point Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 9 (1998), 994–1010. doi:10.1109/34.713364. See page 21.

SKvW*92. SEGAL M., KOROBKIN C., VAN WIDENFELT R., FORAN J., HAEBERLI P.: Fast shadows and lighting effects using texture mapping. *SIGGRAPH Comput. Graph. 26*, 2 (1992), 249–252. doi:10.1145/142920.134071. See page 189.

SLHF09. SETTGAST V., LANCELLE M., HAVEMANN S., FELLNER D. W.: Spatially coherent visualization of image detection results using video textures. *Proceeding of the 33th Workshop of the Austrian Association for Pattern Recognition (AAPR/OAGM) 1* (2009), 13–23. See page 201.

SLS*08. SCHERER R., LEE F., SCHLGL A., LEEB R., BISCHOF H., PFURTSCHELLER G.: Toward self-paced braincomputer communication: Navigation through virtual worlds. *IEEE Transactions on Biomedical Engineering 55* (2008), 675–682. See pages 142 and 148.

SMFF04. SETTGAST V., MÜLLER K., FÜNFZIG C., FELLNER D.: Adaptive Tesselation of Subdivision Surfaces. *Computers & Graphics 28* (2004), 73–78. See page 10.

SMM*08. SCHOOR W., MASIK S., MECKE R., SEIFFERT U., SCHENK M.: VR Based Visualization and Exploration of Barley Grain Models with the Immersive Laser Projection System - Elbe Dom. In *Proc. of 10th Virtual Reality International Conference* (Laval, France, 2008), pp. 217–224. See page 106.

Sny92. SNYDER J. M.: *Generative modeling for computer graphics and CAD*. Academic Press Professional, Inc., 1992. URL: http://dl.acm.org/citation.cfm?id=130368. See page 38.

Sri95. SRINIVASAN R.: Remote Procedure Call Protocol Specification Version 2, 1995. See page 81.

SST10. STROBL M., SCHINKO C., TORSTEN U.: Euclides – a javascript to postscript translator. In *Proceedings of the International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (Computation Tools)* (2010), pp. 14–21. See page 29.

SSUF11. SCHINKO C., STROBL M., ULLRICH T., FELLNER D. W.: Scripting Technology for Generative Modeling. *International Journal On Advances in Software 4* (2011), 308–326. See page 29.

Sta98. STAM J.: Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques 1* (1998), 395 – 404. doi:10.1145/280814.280945. See page 9.

Ste99. STEWART C. V.: Robust Parameter Estimation in Computer Vision. *SIAM Review 41*, 3 (1999), 513 – 537. doi:10.1137/S0036144598345802. See page 46.

Sto03. STONE M.: *A Field Guide to Digital Color*. AK Peters, Ltd., 2003. See page 122.

STU07. SCHWAIGER M., THÜMMEL T., ULBRICH H.: Cyberwalk: Implementation of a Ball Bearing Platform for Humans. *Lecture Notes in Computer Science 4551* (2007), 926–935. doi:10.1007/978 – 3 – 540 – 73107 – 8$_1$02. See page 132.

SUF07. SETTGAST V., ULLRICH T., FELLNER D. W.: Information technology for cultural heritage. *IEEE Potentials 26*, 4 (2007), 38–43. doi:10.1109/*MP*.2007.4280332. See pages 38 and 144.

Sun04. SUN MICROSYSTEMS INC.: Java Remote Method Invocation Specification. http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf, 2004. URL: http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf. See page 81.

SWK07. SCHNABEL R., WAHL R., KLEIN R.: Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum 26*, 2 (2007), 214–226. See page 45.

SWWK08. SCHNABEL R., WESSEL R., WAHL R., KLEIN R.: Shape recognition in 3d point-clouds. In *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008* (Feb. 2008), Skala V., (Ed.), UNION Agency-Science Press. See pages 45 and 56.

SZZK05. SORMANN M., ZACH C., ZEBEDIN L., KARNER K.: High quality 3d reconstruction of complex cultural objects. *Proc of the CIPA (International Scientific Committee for Documentation and Architectural Photogrammetry) International Symposium* (2005), 1–6. See page 144.

Tat06. TATARCHUK N.: Dynamic parallax occlusion mapping with approximate soft shadows. In *In SIGGRAPH 06: ACM SIGGRAPH 2006 Courses, ACM* (2006), Press, pp. 63–69. See page 6.

TGP04. TOWNSEND G., GRAIMANN B., PFURTSCHELLER G.: Continuous EEG classification during motor imagery–simulation of an asynchronous BCI. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 12* (2004), 258–265. doi:10.1109/*TNSRE*.2004.827220. See pages 144 and 148.

Tuo06. TUOHY C.: Topic maps and tei — using topic maps as a tool for presenting tei documents. In *TEI Day in Kyoto 2006* (2006), pp. 85–99. URL: `http://coe21.zinbun.kyoto-u.ac.jp/tei-day/TEIDayKyoto2006.pdf`. See page 185.

UD12. UPCHURCH P., DESBRUN M.: Tightening the precision of perspective rendering. *Journal of Graphics Tools 16*, 1 (2012), 40–56. doi:10.1080/2151237*X*.2012.649987. See page 117.

UF11. ULLRICH T., FELLNER D. W.: Linear Algorithms in Sublinear Time – a tutorial on statistical estimation. *IEEE Computer Graphics and Applications 31* (2011), 58–66. doi:10.1109/*MCG*.2010.21. See page 26.

UI00. ULLMER B., ISHII H.: Emerging frameworks for tangible user interfaces. *IBM Systems Journal 39* (2000), 915–931. doi:10.1147/*sj*.393.0915. See page 150.

Ull11. ULLRICH T.: *Reconstructive Geometry*. PhD thesis, TU Graz, Diss., 2011, 2011. 322 p. URL: `http://www.cgv.tugraz.at/CGV/People/people/ullrich/Dissertation/Dissertation_Ullrich.pdf`. See page 54.

USB10. ULLRICH T., SETTGAST V., BERNDT R.: Semantic enrichment for 3d documents techniques and open problems. In *ELPUB 2010 - Publishing in the networked world: transforming the nature of communication* (2010), pp. 79–88. See page 199.

USF08. ULLRICH T., SETTGAST V., FELLNER D. W.: Semantic fitting and reconstuction. *Journal on Computing and Cultural Heritage 1*, 2 (2008), 1201–1220. doi:10.1145/1434763.1434769. See pages 38 and 201.

VBRR02. VOSS G., BEHR J., REINERS D., ROTH M.: A multi-thread safe foundation for scene graphs and its extension to clusters. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization* (2002), EGPGV '02, pp. 33–37. URL: `http://dl.acm.org/citation.cfm?id=569673.569679`. See page 77.

VG06. VERGAUWEN M., GOOL L. V.: Web-based 3d reconstruction service. *Machine Vision and Applications 17*, 6 (2006), 411–426. See pages 23 and 174.

VGSR04. VOSSELMAN G., GORTE B. G. H., SITHOLE G., RABBANI T.: Recognizing Structure in Laser Scanner Point Clouds. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 46*, 8 (2004), 33–38. See page 40.

Vis09. VISUAL TECHNOLOGY SERVICES: PDF3D, 2009. URL: `http://www.pdf3d.co.uk`. See page 68.

VMJ97. VRADI T., MARTIN R., J. C.: Reverse engineering of geometric models - an introduction. *Computer-Aided Design 29*, 4 (1997), 255–268. See page 39.

VRM97. VRML, The Virtual Reality Modeling Language. Web3D Consortium, 1997. Language Specification Version 2.0, ISO/IEC IS 14772-1, available from www.web3D.org. See pages 14 and 163.

WBM*02. WOLPAW J. R., BIRBAUMER N., MCFARLAND D. J., PFURTSCHELLER G., VAUGHAN T. M.: Brain-computer interfaces for communication and control. *Clinical Neurophysiology 113* (2002), 767–791. See page 142.

WCL*05. WEERAWARANA S., CURBERA F., LEYMANN F., STOREY T., FERGUSON D.: *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, march 2005. URL: http://dl.acm.org/citation.cfm?id=1051879. See page 90.

Wel91. WELLNER P.: The digitaldesk calculator: tangible manipulation on a desk top display. In *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology* (1991), pp. 27–33. doi:10.1145/120782.120785. See page 129.

WGK05. WAHL R., GUTHE M., KLEIN R.: Identifying Planes in Point-Clouds for Efficient Hybrid Rendering. *Proceedings of The 13th Pacific Conference on Computer Graphics and Applications 1* (2005), 1–8. URL: http://cg.cs.uni-bonn.de/aigaion2root/attachments/wahl-2005-hybrid.pdf. See page 40.

Whi06. WHITE C.: King kong: the building of 1933 new york city. In *ACM SIGGRAPH 2006 Sketches* (New York, NY, USA, 2006), SIGGRAPH '06, ACM. doi:10.1145/1179849.1179969. See page 29.

WK05. WU J., KOBBELT L.: Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum 24*, 3 (2005), 277–284. doi:10.1111/*j*.1467 – 8659.2005.00852.*x*. See page 40.

WKCB07. WANG Y., KRUM D. M., COELHO E. M., BOWMAN D. A.: Contextualized videos: Combining videos with environment models to support situational understanding. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1568–1575. doi:10.1109/*TVCG*.2007.70544. See page 189.

WMW06. WINKELBACH S., MOLKENSTRUCK S., WAHL F. M.: Low-Cost Laser Range Scanner and Fast Surface Registration Approach. In *28th Annual Symposium of the German Association for Pattern Recognition* (2006), Franke K., Müller K.-R., Nickolay B.,, Schäfer R., (Eds.), vol. 4174 of *Lecture Notes in Computer Science*, Springer, pp. 718–728. doi:10.1007/11861898$_7$2. See page 19.

WO90. WARE C., OSBORNE S.: Exploration and virtual camera control in virtual three dimensional environments. *Proceedings of the 1990 Symposium on Interactive 3D Graphics 1* (1990), 175 – 183. doi:10.1145/91385.91442. See page 131.

WPA97. WELLS M. J., PETERSON B. N., ATEN J.: The Virtual Motion Controller: A Sufficient-Motion Walking Simulator. *Proceedings of 1997 Virtual Reality Annual International Symposium 1* (1997), 1–8. URL: http://www.hitl.washington.edu/publications/r-96-4/. See page 131.

WPL04. WANG W., POTTMANN H., LIU Y.: Fitting B-spline curves to point clouds by curvature-based squared distance minimization. *ACM Transactions on Graphics 25* (2004), 214 – 238. doi:10.1145/1138450.1138453. See page 40.

WW08. WATSON B., WONKA P.: Procedural Methods for Urban Modeling. *IEEE Computer Graphics and Applications 28*, 3 (2008), 16–17. doi:10.1109/*MCG*.2008.57. See page 26.

WWSR03. WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant Architecture. *International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2003 22*, 3 (2003), 669 – 677. doi:10.1145/1201775.882324. See pages 11 and 26.

ZG08. ZHANG X., GRACANIN D.: Streaming web services for 3D portal applications. *Proceedings of the 13th international symposium on 3D web technology 13* (2008), 23–26. doi:10.1145/1394209.1394216. See page 90.

Zha97. ZHANG Z.: Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting. *Image and Vision Computing Journal 15*, 1 (1997), 59–76. doi:10.1016/*S*0262 – 8856(96)01112 – 2. See page 46.

Zor00. ZORIN D.: A method for analysis of C1-continuity of subdivision surfaces. *SIAM Journal of Numerical Analysis 37*, 5 (2000), 1677–1708. doi:10.1137/*S*003614299834263*X*. See page 8.

ZSD*00. ZORIN D., SCHRÖDER P., DEROSE T., KOBBELT L., LEVIN A., SWELDENS W.: Subdivision for Modeling and Animation. *SIGGRAPH 2000 Course Notes 1* (2000), 1–116. URL: http://mrl.nyu.edu/~dzorin/sig00course/. See page 9.

ZTH*12. ZMUGG R., THALLER W., HECHER M., SCHIFFER T., HAVEMANN S., FELLNER D. W.: Authoring Animated Interactive 3D Museum Exhibits using a Digital Repository. *Procedings of the 13th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage (VAST)* (2012), 73–80. doi:10.2312/*VAST*/*VAST*12/073 – 080. See page 200.

ZWG*07. ZHANG D., WANG Y., GAO X., HONG B., GAO S.: An algorithm for idle-state detection in motor-imagery-based brain-computer interface. *Computational Intelligence and Neuroscience, special issue: "Brain-Computer Interfaces: Towards Practical Implementations and Potential Applications"* (2007), 1–9. doi:10.1155/2007/39714. See page 148.