4th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage
VAST (2003)
D. Arnold, A. Chalmers, F. Niccolucci (Editors)

New Approaches to Efficient Rendering of Complex Reconstructed Environments

S. Havemann,[1] D. W. Fellner,[1] A. M. Day[2] and D. B. Arnold[3]

[1] Braunschweig University of Technology, Germany
[2] University of East Anglia, UK, [3] University of Brighton, UK

Abstract

The creation of complete reconstructions of populated urban environments are technically difficult tasks primarily due to economic constraints in the modeling phase: complex models need to keep rendering aspects in mind in order to warrant interactive rendering speeds which makes this kind of work a labor-intensive task for highly skilled personel. Specialized modelling tools, which exploit knowledge of the types of object being modelled by working in the application domain, can be used to create appealing virtual reconstructions quickly. At the same time, the structural information from the modeller gives essential hints to the interactive renderer to determine efficient interactive display strategies through the use of level-of-detail and culling techniques. Even more important, only a shift in the modeling paradigm from "just in case" to "just in time" can solve the problem applications are faced in real-time rendering. In this paper we discuss the way in which polygonal and multi-resolution surface techniques can complement one another in the effective rendering of complex reconstructed environments. We also draw more general conclusions which apply to other software systems that share the same objectives.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Display algorithms, Viewing algorithms I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

The Charismatic research project [cha] is about the economic production and the interactive rendering of complete reconstructions of populated sites of historic and cultural interest, often referred to as Cultural Heritage Sites [BWW*01, WWDA01, FWB*01]. Technically this a difficult task because of the size and level of detail required of the reconstructions. Not only is the model creation difficult but the rendering of large numbers of houses, buildings and streets is also extremely demanding. Clearly the traditional approach for creating 3D content would not be satisfactory. This approach has been to create highly detailed, textured models with standard, all-purpose 3D modelling toolkits such as 3Dstudio Max (Kinetix) or Maya (Alias/Wavefront). The models are then exported to a general purpose rendering engine to interactively explore the virtual reconstruction. The main problems with this approach are that:

- the general purpose modelling tools mean that the system is much less able to exploit knowledge of the application domain to simplify the user interactions and to assist in the creation of typical building structures
- models created without exploiting domain knowledge are more difficult to optimise for real time rendering of complex scenes. In other words the person doing the modelling is very likely to create models that are inefficient to render and the system is less able to optimise these models if it has to assume that the user was creating completely general models.

More specialized modelling tools, which exploit knowledge of the types of object being modelled by working in the application domain, can be used to create appealing virtual reconstructions quickly. At the same time, the structural information from the modeller gives valuable hints to the renderer to efficiently optimise interactive display through the use of model/domain specific level-of-detail and culling techniques.

Even more important, only a shift in the modeling paradigm from "just in case" to "just in time" can solve the problem applications are faced in real-time rendering. This

means that instead of having tons of polygons in the model to provide the expected finest level of detail for a particular viewing situation (which might still not be enough) so called generative models provide the necessary surface detail dynamically and only when necessary.

Thus a modeller that knows the operator is creating houses can use this information to simplify the user interaction and guide the operator, but also to create models that build in optimisations to support real-time rendering. This general underlying idea was implemented with a full range of different techniques developed within Charismatic. In particular, two strands of research were pursued and combined:

1. Polygon and shell modelling [BWW*01]
2. Multi-resolution surface modelling [HF01].

Having presented the modeling aspects in [FWB*01, HF01] this paper focuses on the way in which these techniques complement one another in the efficient rendering of complex reconstructed environments.

2. Modelling Ingredients

2.1. Polygon and Shell based Modelling

In many virtual heritage scenes, the majority of the buildings will be typical domestic housing of the period, with slight variations, and often such buildings need to be produced in high detail and in large quantities to populate the scene [BC00, AvL98, dVJ98]. This presents problems both in human resources to create these objects, and in rendering the large quantities of high polygon models produced. Our approach to this problem is to model the houses at full detail using polygonal meshes, and build in level of detail techniques applied to the resulting model to reduce complexity when rendering at a distance.

Our approach uses a hierarchical scenegraph, enabling it to incorporate optimized structures for modelling and rendering houses and landscapes, and appropriate level of detail methods for houses and other objects in the scene, such that they no not produce visible artifacts. The main modelling tool is the *Shell Modeller* using the scenegraph, with a shell as the basic building unit which provides localized hierarchical structure to the modelled spaces consisting of rectangular parallelepipeds representing one room or a group of rooms. This program combines rapid creation of the main structure of the house, with low geometry counts, and integrated automatic level of detail mechanisms including the automatic creation of texture maps of all faces of the full detail model for distance viewing.

The building shell hierarchy is based on the concept of representing the major components of a house as a set of 'shells' with each shell delineating a room or a collection of rooms. Figure 1 shows a root shell with another shell attached to one wall.

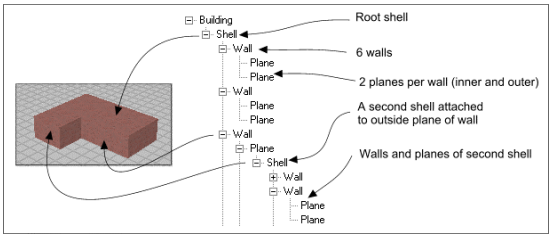

Figure 1: *Data structure layout of a root shell with another shell attached*

This shell structure is maintained when the model is exported and is used by the renderer for culling the internal model at distance. It can also be used for culling out parts of the building when close to or inside the model.

There are three types of shells: *Shells with interiors* (forming the main structure of the house, with the interior shells being created automatically from the external dimensions and specified wall and ceiling thicknesses), *Empty shells* (have no interiors and are useful for creating features such as timbering) and *Openings* (these have the same internal data structure as an empty shell, but are placed between the inner and outer planes of a wall and have stencils on their front and back faces to create the opening).

No other geometry is created in the Shell Modeller other than these three objects, which form the lowest level of geometric detail for a house that is rendered at distance, although provision is made to replace all geometry where possible using impostors.

Figure 2: *A street in Wolfenbüttel (top row) and its reconstruction (bottom row)*

2.2. Multi-Resolution Surface Modelling

Meshes represent the smallest common denominator for 3D objects and can be exported by most commercial mod-

ellers. They pose notorious problems for interactive display, because a priori they do not provide support for view-dependent rendering. This means that they are usually over-sampled when viewed from a distance when they cover just a few pixels, and they are under-sampled when used for close-up views. In order to improve the situation, we have implemented the Multiresolution Toolkit which supports objects that permit a change of model resolution at runtime. The Multiresolution Toolkit provides two multiresolution object representations:

- *Progressive Meshes* as presented in [GH97, Hop96] and
- *Combined Breps* as demostrated at the VAST'01 conference [HF01] (see Figure 3).

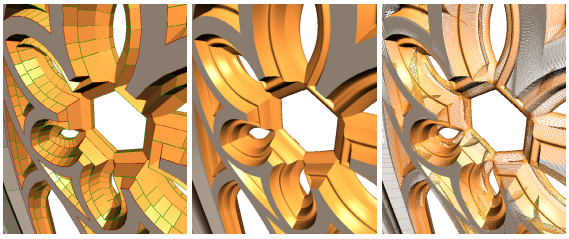

Figure 3: *Consecutive sharp edges form uniform B-Spline crease curves. Left: Combined BRep mesh, middle: resulting surface, right: wireframe of tesselated surface.*

3. Combination of Rendering Approaches

Only a *combination* of different realtime rendering strategies will yield satisfactory results. Detailed urban environments are especially demanding because of the variety of different objects and shape representation formats in the scene. The ultimate goal of realtime rendering is output sensitivity: To send only those geometric primitives to the graphics hardware that make a perceivable contribution to the image. But no single strategy can be expected to perform well in all possible situations. For this reason, we use a well-balanced mix of different methods in our realtime-renderer. These methods are of course in direct correspondence to the modeling tools and object representations explained earlier.

3.1. Use of the Scenegraph

In order to minimise code maintenance, the same scenegraph structure has been used for the renderer as for the modelling applications. This means that all applications in the polygonal toolkit share the same code base and file formats, so data is easily exchanged between them. The scenegraph data structure was designed with both easy use as an API by the modelling tools and efficient rendering in mind. This is achieved by using a specialised data structure for each object type – tree, building, particle system etc. By implementing a special structure for each object type in this way, efficient specialised LOD techniques can be included to speed up rendering. The specialised object types supported by the scenegraph are Buildings, Trees, Particle Systems, Landscape and general Triangle meshes. All modelling in the tool kit is done at the highest level of detail so it is the rendering application's task to reduce the amount of polygonal detail as it is running. The next few sections describe the approaches to this.

3.2. Culling Techniques

View Frustum Culling The graphics library used is OpenGL, which provides most 3D functionality needed and has efficient drivers for various graphics cards which makes sure that accelerated 3D hardware is used, wherever the renderer is used. OpenGL provides a primitive version of view frustum culling at a late stage in the pipeline. It ensures that polygons do not extend beyond the limits of the screen, which would cause errors. However, because this method works late in the graphics pipeline, objects which will eventually be completely culled, are still sent to the graphics card, consuming valuable bandwidth of the AGP bus. Our polygonal renderer uses a quick bounding box check to determine if an object is outside the view frustum or not. Objects determined to be entirely outside the view are not sent to the graphics card for rendering, which saves bandwidth on the AGP bus.

Occlusion Culling Standard OpenGL also renders objects that are occluded behind others, even though they make no difference to the resulting image. The UEA polygonal renderer employs a cell-based occlusion against all objects before they are rendered. The check is performed by rendering occlusion shadows into the graphics frame buffer where each pixel relates to a cell in the scene. Once occlusion shadows have been rendered for all the closest objects in the scene, the z-buffer is read back into an array as an occlusion map. Each object can then be checked for occlusion against a pixel in the occlusion map. Objects determined to be occluded are not sent to the graphics card for rendering. This work was based on a number of approaches from the relevant literature [WS99, GJ01, KCO00].

Tree culling Trees can also benefit from having portions of their branches culled at a distance. When the view point is far from the tree, leaves tend to obscure the thinnest branches, whilst the trunk is almost always visible. The LOD system on the trees progressively removes outer branches as the view point retreats. The remaining branches also become less complex, starting at 16-sided cylinders, reducing to 4-sided ones at the furthest distance. This is an area in which there are significant further improvements to be made in rendering speeds.

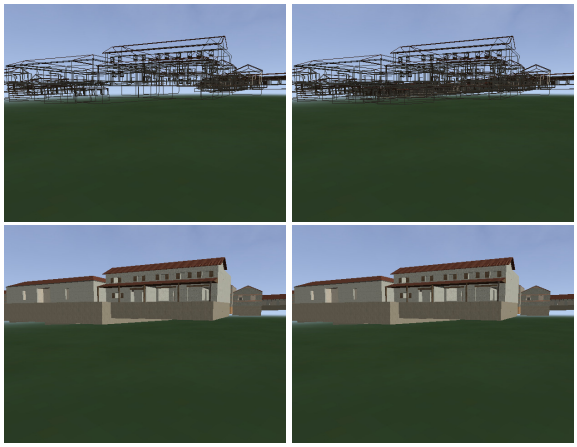

Figure 4: *OpenGL rendering with and without occlusion culling. Above Left: 16K triangles @ 13 FPS, Above Right: 45K triangles @ 6 FPS. Below: No difference in rendered scene.*

3.3. LOD Techniques

House LOD The building data structure supports a specialised LOD system that removes complex objects that are almost flat to the wall when the viewer is distant. These "almost flat" features, such as window frames and timber beams, can take up as much as 90% of the geometric complexity of a typical building. The "House LOD" system used in the polygonal renderer is special because the transition between full detail and low detail with these "external geometry" items removed is seamless – there are no popping artefacts.

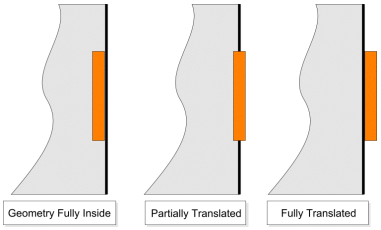

Figure 5: *Gradual transition of window geometry through a wall*

The seamless transition is achieved by representing the "missing" geometry in the texture of the remaining wall, so shutters, doors and windows remain "painted on" the wall at all times. The special textures for the walls are automatically generated when buildings are loaded into the renderer, so that each wall has available two textures; the "Base Texture", which just the wall material, with no external geometry and the "LOD Texture" which is the wall material with projections of the external geometry overlaid. When the viewer is close to a building, the base textures are used on all walls, and external geometry is displayed at full detail. When the user starts to retreat from the building, the external geometry objects begin to slide back into the walls that they are attached to and the texture is switched to the LOD texture. As the user retreats further, the external geometry disappears into the walls and is culled, leaving only the LOD texture to represent them.

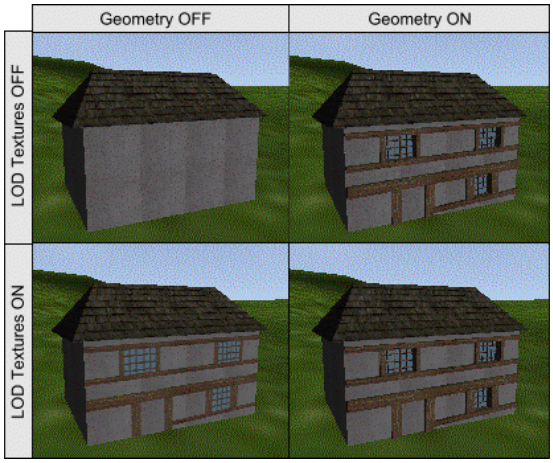

Figure 6: *Bottom Row shows full geometry vs. geometry represented by texture*

Impostors House LOD works well for reducing triangle geometry in buildings only. It cannot be easily applied to trees or arbitrary meshes which have no regular structure. The polygonal renderer uses another LOD type on buildings, trees and triangle meshes at a far distance called object impostors. The impostor system reduces triangle geometry by replacing individual objects with a billboard when they are further than a user-specified distance away. The billboard mesh consists only of 2 triangles – making a square. The billboard is textured with an RGBA texture especially generated to make the difference between an impostor representation and the full triangle mesh indistinguishable. The use of impostors falls into the LOD category of 'image caching' because the generated impostor textures can be used over a number of frames, so each object is not re-rendered every frame, as it is in a traditional rendering model. This is particularly useful in scenes when there is high frame-to-frame coherence, such as city walkthroughs, because the image calculations can be reused over a number of frames. The pictures below show a comparison of a typical scene.

The right Figure 7 shows an incorrect view of the above scene to illustrate where the 102,000 triangle saving is made. Most of the buildings in the bottom left hand corners are rendered normally, but the rest of the scene is represented by impostors. The particular example needs 4MB of texture space for all its impostors, whereas the AGP bandwith saved from 102,000 triangles is 3.6MB per frame.

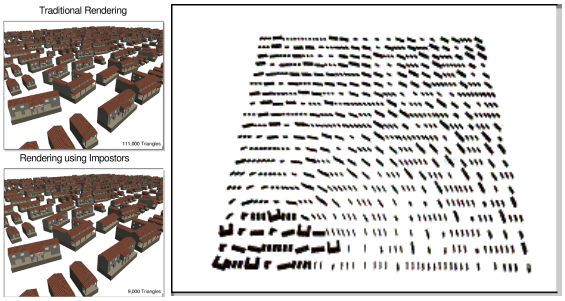

Figure 7: *Comparison of Impostors vs. Polygon Rendering: Traditional Rendering (111,000 Triangles) and Rendering using Impostors (9,000 Triangles)*

ROAM A *Realtime Optimally Adaptive Mesh* [DWS*97] is used on the landscape height map to provide enough LOD performance that the source data can be very detailed. ROAM works by producing a variance tree (in a binary tree) from the height map and using the data structure to base fast decisions on which areas of the landscape to triangulate to high details and which areas to leave as large flat triangles.

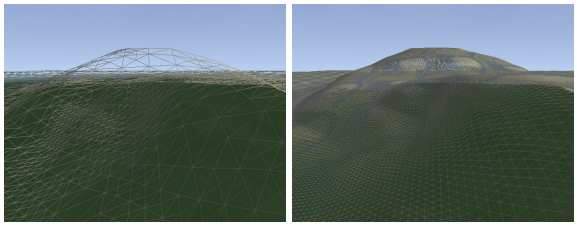

Figure 8: *Comparison of landscape with/without ROAM. Left example: 4K triangles @ 42 FPS, Right: 381K Triangles @ 2 FPS*

Efficient GL coding Another aim was to produce houses with interiors, using walls with depth, and to have openings through the walls. However, houses with openings in the walls created in existing modelling packages normally have many triangles in the wall to create the opening(s). These triangles are not necessary for the rendering of the wall itself (and in fact can cause rendering artefacts). In addition, the triangulated wall requires simplification if the openings are to be removed when rendering at distance. Models with many openings will therefore have high geometry counts, creating frame rate problems in real time rendering in scenes where many houses may be visible at one time.

To overcome this the Shell Modeller creates all openings as stencils, leaving the wall as a single quad, even if there are multiple openings on a wall. With rectangular openings four quads replace the exposed space between the inner and outer walls. This technique yields up to a 2/3 reduction (for each wall $n+1$ quads for n openings using stencils compared to $3n+1$ otherwise) in the geometric complexity of a typical house, and allows the parts that require complex geometry to be handled separately from the main structure. Therefore that openings can, if desired, be removed from walls at distance, without having to re-triangulate the wall or even change the texture on it.

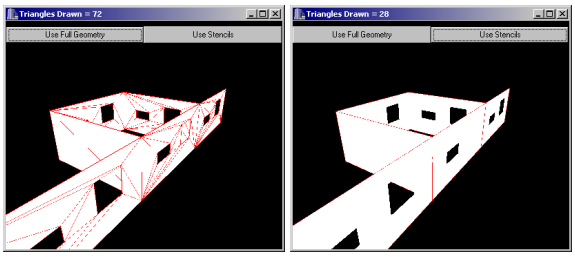

Figure 9: *When not using stencils, faces with holes need to be triangulated. Right: Same building using stencil masks.*

The stencil buffer allows the masking of areas of the frame buffer, and is now supported in hardware by modern graphics cards.

OpenGL Extensions There are a number of places in which graphics extensions have been used to speed up rendering or improve visual quality. Once such example is the use of register combiners for dynamic lighting of the landscape. Because the landscape in the renderer has ROAM LOD applied, it is not practical to use standard OpenGL vertex lighting, because the vertices are continuously being added and removed. The solution in the polygonal renderer was to generate a 'normal map' for the landscape and shade each pixel according to the dot product of the landscape's normal vector and the vector of light from the sun. Using the hardware register combiners on the graphics card means that this operation is done per pixel, and so smooth lighting is guaranteed.

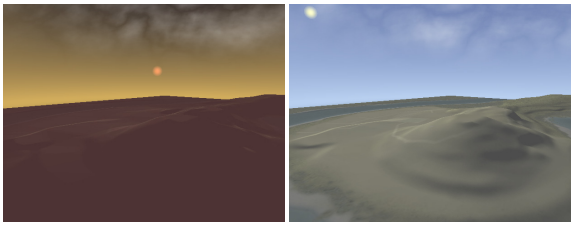

Figure 10: *Same landscape under different lighting conditions*

Another extension employed is the use of fast primitive rendering methods for triangle meshes. These are handled using NV_Draw_Range_Elements because it significantly speeds up the transfer of vector data to the GPU.

Combination of LOD Methods The techniques used in the polygonal renderer have been carefully selected to work together. Occlusion culling and Impostors work well together, since they handle different situations – occlusion is high when the user is close to the ground whiles Impostors are good for scenes where the user can fly over a city, viewing a large portion of it at a distance. House LOD and Impostors work well in tandem because they are used at different distance ranges, and ROAM works well because it maintains the silhouette of the landscape at all distances. The combination of all the listed techniques when applied to an example model (approximately 550,000 polygons) comprising both buildings and terrain gave frame rates of about 36 FPS compared to 1.6 FPS without any (using a Pentium 3, 733 Mhz, 512MB RAM, GeForce3 graphics card).

3.4. Rendering Progressive Meshes

A progressive mesh [Hop96] consists of a very coarse base mesh and the split sequence. In order to display it, the base mesh and the split sequence are loaded, and at least one instance is created. An instance consists of a copy of the base mesh and its current LOD value, 0 at the beginning. In order to refine (or coarsen) the instance, split (or collapse) operations are executed until the desired level of detail is reached. The execution of these elementary operations is very fast and reaches a rate of 200K vertices per second on up-todate PC hardware. Once a Progressive Mesh is loaded, an arbitrary number of instances can be created. This is useful if the same object appears multiple times in the scene. As each instance can be rendered multiple times in a frame, for instance 44 columns can appear, which are shown at either of three different levels of detail, using only three instances. This is shown in Figure 11, with extremely simplified columns in the back, for demonstration. A simple distance heuristic is applied to choose from the three available resolutions.

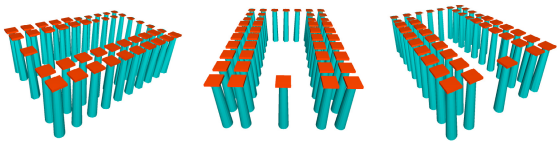

Figure 11: *Three instances of the multiresolution column (5%, 15%, and 100%), with each instance rendered multiple times*

3.5. Rendering Combined BReps

The problem with subdivision surfaces is that the number of faces grows by a factor of four with each subdivision. A cube, three times subdivided, already results in 384 quads, as can be seen in Figure 12. This problem is overcome by a scheme for tessellation on the fly. The basic idea is to allocate empty chunks of memory for the tessellation, and to execute the actual subdivision only on demand, if a patch is visible. Once the memory is filled with computed points and normals, switching between different levels of detail can be accomplished at virtually no cost. The reason for this is that triangle strip indices are pre-computed on a per-batch basis for all possible subdivision combinations. To switch between resolutions then simply means to use a different index list.

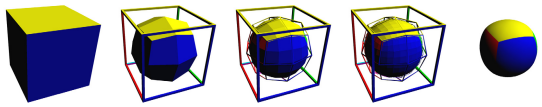

Figure 12: *A cube with three levels of recursive subdivision, rightmost object uses normals per vertex instead of face normals.*

An automatic frame rate control can adjust the resolution of all visible patches by adjusting a global quality parameter, which is a number between 0.0 and 1.0. It attempts to keep the frame rate between 20 and 40 FPS by changing the quality parameter, based on the time the last frame took to render. Figure 13 shows a typical effect of an increase in quality; no popping artefacts are introduced. The technical details of the implementation can be found in [Hav02].

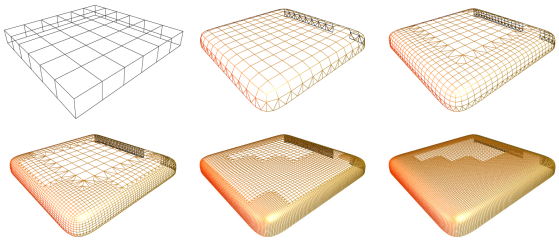

Figure 13: *View-dependent tessellation with increasing level of detail, the* 6×6 *quad faces on the top are subdivided into* 24×24 *patches.*

The mesh representation underlying Combined BReps is a half-edge data structure with rings. This gives a fair amount of freedom to modelling free-form openings for instance. The first image in Figure 14 shows a complicated sharp white face that is a single BRep face with eight holes. In addition to this, most of the vertices at its border are crease vertices, so when tessellated, it has a nice BSpline border. The next image gives an idea of the number of triangles that are created when necessary.

The last image shows the purpose of this effort: The best possible quality is delivered at close-up views. This quality would not have been possible with the OpenGL stencil technique as used by the UEA for rectangular windows and doors. The reason is that stencils are essentially bitmaps. A stencil for a curved opening would necessarily reveal grid artefacts.

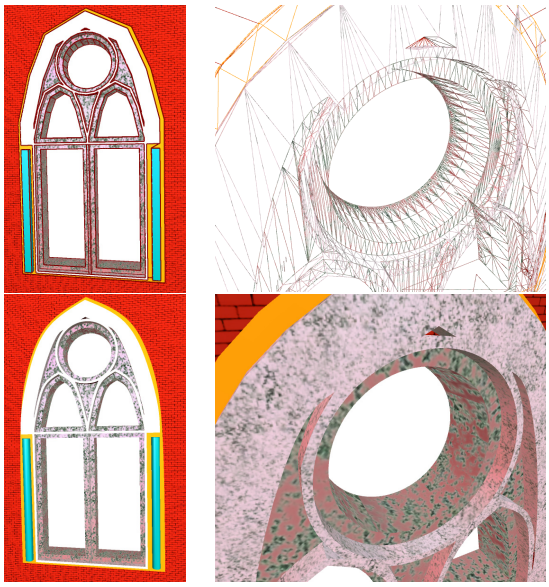

Figure 14: *Rounded shapes and openings are not for free*

3.5.1. Combination of Both Multiresolution Techniques

The Multiresolution toolkit provides two model representations: Progressive Meshes provide the simplification up to a radical degree, and Combined BReps can refine models by orders of magnitude. Both techniques can complement one another in an ideal way. To use them together, distance ranges must be defined for the two quality parameters.

- In a distance range $[0, d_0]$, Combined BReps are used. The tessellation quality decreases from 1.0 at close distance to 0.0 at distance d_0.
- At distance d_0, the rendering is switched to polygon rendering
- At distance $> d_0$, Progressive Meshes are used with a quality 1.0 at d_0, falling off to some minimal quality level $q_{\min} < 1.0$ at a distance d_1.

Experiments with this technique have revealed that the crucial point is the selection of the different parameter ranges. Great effects can be achieved though if they are carefully chosen. Figure 15 gives an impression of the appearance of the combination. As a last resort for distant objects, the rendering of Progressive Meshes can also be nicely combined with the impostor technique from the UEA.

3.6. Summary of Rendering Approaches

A comparison of the rendering approaches reveals the true power of Charismatic. In both strands of development, the underlying concepts serve to one predominant goal: To isolate only the essential bits of information necessary to represent a virtual reconstruction, and to start rendering from this minimal set of given data. This point of view also reveals that modelling and rendering of a virtual city are tightly related: The same sort of efficient encoding that makes the modelling very easy equally forms the basis for optimised interactive display. A shell is the basic building block for creating whole buildings. Yet a single shell can be described with just a few bytes of data (except for the texture). But as described above, its usefulness goes beyond that, as it is an efficient occluder, and it can be used for 'swallowing' attached geometry in imperceptible ways. Finally, a single textured quad can efficiently represent it at a distance.

Figure 16: *Subdivision Surfaces integrated with Polygonal Renderer*

Combined BReps are also a major step into the same direction, as really massive amounts of geometry are created from comparably lean control meshes on demand. The parent shell of a freeform element triggers its level of detail. At a distance, polygon rendering is sufficient, then the control mesh itself is simplified, and finally the attached geometry is completely replaced by an impostor.

4. Further work

The concepts developed present various possibilities for extension and generalization. As already outlined before, the shell concept has further potential, which equally applies to rendering and to modeling. Shells are good occluders, because they are opaque and have very low polygon count. With hierarchical shells, a single shell at a distance could replace a conglomeration of shells that represents a complex house.

It is even possible to provide the buildings with complex interiors. The concept of interior shell potentially frees the renderer from the burden of displaying the rest of the city. To make this effective more work needs to be put into handling the rendering of openings within a shell – for example the views from the window, or the views into a lit room from outside a building at night. These cases are likely to draw on the experiences of using cells and portals. The released resources can be used to instantly generate high-quality meshes for furniture, staircases, and household items. Ideally, the respective control meshes will be also generated on the fly, through evaluation of a procedural description for these objects. Ideally, the respective control meshes will be also generated on the fly, through evaluation of a procedural description for these objects. A first prototype of such an procedural shape description language for online evaluation,

Figure 15: *Blending from Combined BReps (quality 1.0 and 0.8) to Progressive Meshes (quality 100%, 80%, 50%, 30%)*

the *Generative Modeling Language (GML)*, has already been realized. Examples and test programs can be found on our website on www.generative-modeling.org. Evaluating a concise procedural model description only on demand would then completely decouple the size of a virtual city, measured in raw triangles, from the size needed to store the model in a file.

Besides further developing the core technology, other options on our agenda include:

- content distribution over the Internet.
- a standardized scene graph engine for the commercial distribution of Cultural Heritage content to third-party software companies, e.g. at www.opensg.org
- truly immersive historic experiences through cluster-based stereoscopic rendering.

5. Conclusions

The key to success in rendering complex reconstructions is to find the right way to represent as much structural information as possible. The underlying idea and the source of efficiency, is that just a few, well chosen and structured data are sufficient to describe a virtual reconstruction. The renderer can make efficient use of these data and choose the appropriate representation at runtime and generate derived data on the fly.

The most important achievement of this work was to identify and implement the concepts to prove the efficiency of this general approach. The net gain from reducing the data required is simply that the model complexity can be tremendously increased. Rendering is a daunting task only if one is confronted with millions of unstructured polygons or other inappropriate representations.

This research has also shown that there are ways of thinking about the modelling problem which provide substantial improvements to the rendering – i.e. start with a suitable representation for modelling rather than try and optimise rendering of bad models.

The concepts we have developed and presented here demonstrate an underpinning approach and provide the basis for succeeding in realising fully modelled and highly complex reconstructions.

References

[AvL98] ACHTEN H. H., VAN LEEUWEN J. P.: A feature-based technique for designing processes: A case study. In *Proceedings of the 4th Conference on Design and Decision Support Systems in Architecture and Urban Planning* (1998).

[BC00] BRIDGES A., CHARITOS D.: The architectural design of virtual environments. In *Proceedings of the 7th Conference on Computer Aided Architectural Design Futures* (Munich, Germany, 2000), vol. 719–732, Kluwer Academic Publishers.

[BWW*01] BROWNE S. P., WILLMOTT J., WRIGHT L. I., DAY A. M., ARNOLD D. B.: Latest approaches to modelling and rendering large urban environments. In *Proceedings of Eurographics* (UK, 2001).

[cha] Charismatic website. http://www.charismatic-project.com.

[dVJ98] DE VRIES B., JESSURUN A. J.: Features and constraints in architectural design. In *Proceedings of DETC'98* (1998), 1998 ASME Design Engineering Technical Conference.

[DWS*97] DUCHAINEAU M. A., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: Roaming terrain: Real-time optimally adapting meshes. In *IEEE Visualization* (1997), pp. 81–88.

[FWB*01] FLACK P. A., WILLMOTT J., BROWNE S. P., DAY A. M., ARNOLD D. B.: Scene assembly for large scale reconstructions. In *Proceedings of VAST* (Glyfada, Greece, 2001).

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97* (Los Angeles, California, August 1997), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, pp. 209–216.

[GJ01] GERMS R., JANSEN F.: Geometric simplification for efficient occlusion culling in urban scenes. In *Proceedings of WSCG* (Pilsen, Czech Republic, February 2001).

[Hav02] HAVEMANN S.: Interactive rendering of catmull/clark surfaces with crease edges. *The Visual Computer 18*, 6 (2002), 286–298.

[HF01] HAVEMANN S., FELLNER D.: A versatile 3d model representation for cultural reconstruction. In *Proceedings of VAST* (Glyfada, Greece, 2001).

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of SIGGRAPH 96* (New Orleans, Louisiana, August 1996), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, pp. 99–108.

[KCO00] KOLTUN V., COHEN-OR D.: Selecting effective occluders for visibility culling. In *Proceedings of Eurographics* (Interlaken, Switzerland, 2000).

[WS99] WONKA P., SCHMALSTIEG D.: Occluder shadows for fast walkthroughs of urban environments. In *Proceedings of Eurographics* (Milan, Italy, 1999).

[WWDA01] WRIGHT L. I., WILLMOTT J., DAY A. M., ARNOLD D. B.: Rendering of large and complex urban environments for real time heritage reconstruction. In *Proceedings of VAST* (Glyfada, Greece, 2001).