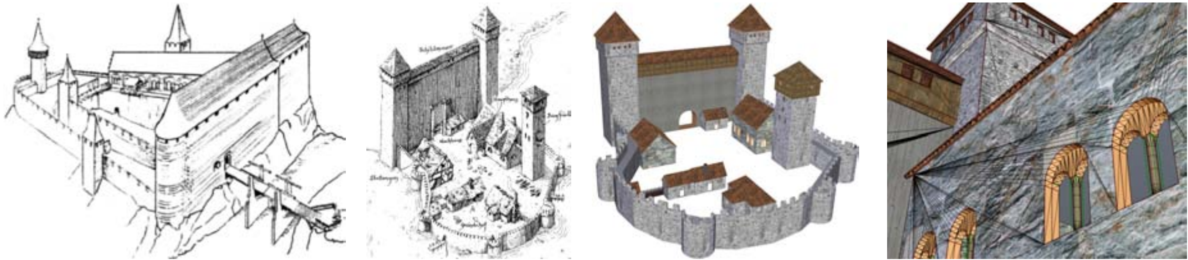
The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage
VAST (2005)
M. Mudge, N. Ryan, R. Scopigno (Editors)

3D Modeling for Non-Expert Users with the Castle Construction Kit v0.5

Björn Gerth[1] René Berndt[1] Sven Havemann[2] Dieter W. Fellner[2]

[1]Institut für ComputerGraphik, TU Braunschweig, Germany
[2]ComputerGraphik & WissensVisualisierung, TU Graz, Austria



Abstract

We present first results of a system for the ergonomic and economic production of three-dimensional interactive illustrations by non-expert users like average CH professionals. For this purpose we enter the realm of domain-dependent interactive modeling tools, in this case exemplified with the domain of medieval castles. Special emphasis is laid on creating generic modeling tools that increase the usability with a unified 3D user interface, as well as the efficiency of tool generation. On the technical level our system innovates by combining two powerful but previously separate approaches, the Generative Modeling Language *(GML) and the* OpenSG *scene graph engine.*

Categories and Subject Descriptors (according to ACM CCS): H.5.1 [Information Interfaces]: Multimedia Information Systems, I.3.6 [Computer Graphics]: Interaction Techniques

1. Introduction

Interactive three-dimensional illustrations have a huge potential for engaging the public audience in museums and exhibitions, as well as for the exchange of scientific hypothesis about the past among researchers. But expectations are high due to the ubiquitous use of 3D, which is slowly but steadily becoming a standard in the entertainment and edutainment sectors. 'Serious games' are expected to deliver the same level of quality as known from X-Box and Playstation, even if realized with only a fraction of the budget.

A central issue is the content authoring problem, in case of 3D also known as the *modeling bottleneck*. In principle there are two ways to create 3D objects, namely shape acquisition (3D scanning) and shape modeling. The focus of our paper is the latter, for which there are two main applications:

- **Virtual reconstructions** of destroyed cultural heritage sites to let users explore their ancient form at a certain point in history, or a hypothesis about it
- **3D Illustrations** as a straight generalization of the familiar 2D drawings in museums exhibitions, e.g., to emphasize the context of the findings and their surroundings

Archeologists, museum curators and art historians usually have a background in human sciences rather than engineering or computer science. These persons are untrained in 3D modeling, but they have distinct three-dimensional ideas about the appearance of historic sites and monuments. But is there a way to let them express their ideas in 3D? – It is a very demanding task to provide this particular community with easy-to-use tools that (i) require no/not much learning, but (ii) still guarantee high-quality results.

© The Eurographics Association 2005.

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

Figure 1: *Castles in recent computer games:* ***Stronghold2*** *is a castle and medieval warfare simulation (a), sophisticated placement tools of* ***The Settlers*** *(b), and the game map editor of* ***The Battle for Middle Earth*** *with integrated castle designer (c,d).*

1.1. Options for Solving the Modeling Bottleneck

Three-dimensional objects are usually created with sophisticated 3D modeling software, with procedural modelers like 3D Studio Max or Maya, or with parametric CAD software like AutoCAD or Catia. These tools require serious learning efforts, and since they are all-purpose tools they do not provide any particular support for creating CH content.

Custom tools exist for the architectural domain, for instance house planning software. It permits average users to create standard houses easily by using libraries of predefined intelligent 3D components such as staircases, walls, and windows, and catalogues of furniture. Drawbacks of house planning software are that (i) the components are for modern but not for ancient architecture and (ii) due to the proprietary format new intelligent components can not be added by normal users but only by programmers.

Modelers such as AutoCAD and Maya are extensible through their built-in scripting languages (AutoLisp and MEL) to let users add specialized modeling functionality by programming. This is a way to speed up the modeling, but it leads to another problem, the *separation of modeling from viewing*: The finished models are exported to some exchange file format (DXF, VRML) and loaded into a runtime viewer, but there any specialized modeling functionality is gone. The export throws away the high-level semantic modeling information, which precludes any on-line high-level editing – but the possiblity to change objects is a major asset when choosing interactive 3D graphics as a media form!

The perceived level of engaging dynamic interaction can be greatly enhanced using game engines. They provide fluent interaction, best possible rendering quality, animated characters (virtual enemies), event-driven interactions, and possibly even physical simulations. But there are some serious issues with using game engines, among others:

- **authoring cost:** the game industry counts in *man years*
- **sustainability:** short lifecycle of game technology
- **re-usablity:** proprietary game data structures are a dead end, no later re-use of models created
- **generality:** rendering is optimized for low-poly, multires-models, no custom renderers supported

The last issue is of great importance in the CH context (and especially for Epoch) where many sophisticated surface representations have been developed that need to be displayed together. This is witnessed by the publications of previous VAST conferences, just to mention densely sampled triangle meshes [BC*04], point clouds [DD*04], or high-quality BTF rendering [MMK04], e.g., for precious artifacts.

To summarize the requirements of a CH authoring system: An ***extensible*** set of ***CH specific*** modeling tools, represented in a ***non-proprietary***, sustainable format standard, combined with a CH presentation system, into which also ***non-standard rendering methods*** can be integrated.

2. Related Work on Domain-Dependent Modeling Tools

The problem of creating custom tailored modeling tools has not received much attention so far as a research topic of its own. Extensive texts on modeling like the *Handbook of CAGD* only mention the fact that domain-dependent tools are important for practical shape design, but do not explain ways to realize them [HJA02]. There is no clear picture or systematic survey, the aforementioned proprietary approaches (MEL, AutoLISP, etc) clearly dominate. Formats such as 3DS (3DStudioMax) or Collada [Col] allow for storing animations and for attaching some high-level information. – One lesson learned from VRML, however, is that any sort of non-trivial interaction requires programming, i.e., to resort to a *second* formalism, such as JavaScript.

The absence of any sort of standard format for domain-dependent, or *procedural*, modeling tools has also been identified as a major problem for the manufacturing industry: Models containing intelligent 3D parts ('*features*') are 'frozen' when they are transferred from one parametric modeler to another, e.g., from SolidWorks to Catia: Intelligence is lost. Some interesting background information about the fundamental obstacles experienced by the STEP consortium trying to solve this problem was given by Michael Pratt on SMI 2004 [Pra04]; a detailed discussion is part of [Hav05].

A thrilling new aspect was recently added to the whole subject by a new generation of computer games from the *simulation* genre. The title *Stronghold2* terms itself the ulti-

© The Eurographics Association 2005.

Figure 2: *User-defined world in **The Settlers**. High quality and great variety is obtained at the same time: The game provides about 60 different pre-defined building types, each with a number of configuration options on several levels of extension.*

Figure 3: *Several typical castles from [Koc00]. They are all composed of similar elements, but each has a different style.*

mate castle simulation [Str05]. The latest *Settlers* from Blue-Byte or the *Lord of the Rings* series from Electronic Arts play in a fantasy world that can be designed to a large extend by the user [Set05, Lor05]. Interesting with respect to domain-dependent modeling is that greatest emphasis is put on *usability*: Gamers are unpatient and not willing to spend precious game time on impractical modeling tools:

- generality is sacrificed for clarity of actions
- only few but very well-chosen options
- no WIMP-style GUI but freely floating graphical objects
- optimized with respect to mouse movements

Fig. 1 shows such streamlined modeling tools in action, the quality of a typical world can be judged in Fig. 2. Similar as with Lego vs. clay, modeling efficiency is obtained by radically reducing the degrees of freedom (DOF), or, more suitably, by exposing only the *essential DOFs* of the model. An important maxim for domain-dependent modeling tools is: Not everything can be changed, but everything that can be changed can be changed efficiently.

3. Technical Foundations: GML and OpenSG

Our approach combines the strengths of two technologies that were previously unrelated:

- The ***Generative Modeling Language*** (GML) does not have a scene graph, so all models exist in the same coordinate system, and there are no multiple shape instances.
- The ***OpenSG*** scene graph engine provides the 'hooks' to change all aspects of the scene at runtime, but it has no scripting language. Hitherto, all types of dynamic changes must be programmed in C++, i.e., defined at compile time.

It turns out that both technologies fit surprisingly well together. Their tight integration opens a number of very interesting options, some of which are sketched in this paper. But first a short introduction into the ingredients.

3.1. The Generative Modeling Language

The GML has been sucessfully employed in a CH context for the procedural construction of Gothic window tracery as shown on VAST 2004 [HF04]. Its important novel feature is that (i) it can encode procedural modeling tools and (ii) it contains a runtime engine to apply these tools interactively. So the GML bridges the gap between modeling and viewing, it can be seen as a viewer with integrated modeling capabilites. This permits for an extremely concise encoding for the web-based transmission of highly complex models of a procedural kind [BFH05]: Instead of 3D objects transmit only the modeling operations that create these 3D objects.

© The Eurographics Association 2005.

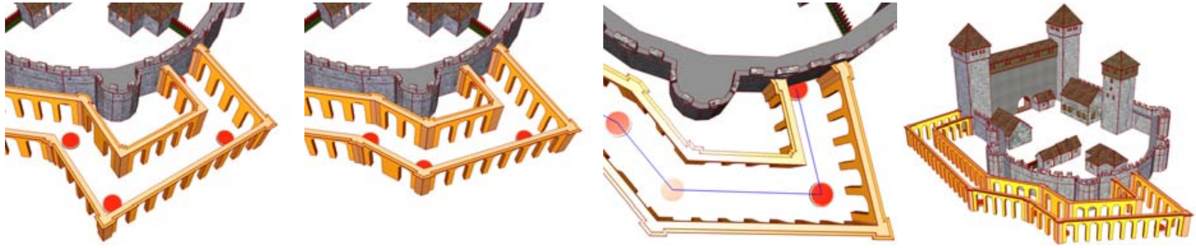


Figure 4: *Interactive dragging of Arkade CVs. It uses a customized version of the polygon editor with geometric error checks.*

The GML is a stack-based programming language, similar to Adobe's PostScript, which solves the nasty *code generation dilemma*: With procedural tools modeling becomes very similar to programming. But it is not tolerable to replace interactive shape design by literal programming using an ascii text editor; furthermore not all good artists are also good programmers. PostScript, on the other hand, is the 'invisible programming language'. Code can be – and is! – generated automatically with ease: Whenever printing a PostScript document the printer actually executes a computer program that produces the bitmap that is printed on paper – technically, as a side effect of program execution.

3.2. The OpenSG Scene Graph Engine

OpenSG is an open source scene graph system with built-in support for (i) multi-threading and (ii) cluster rendering. This feature will be of greatest importance in the near future to assure *scalability*: Parallel computing on multi-core CPUs (e.g., with *Hyperthreading*), is the only option to increase the processing power further when the clock speed comes close to physical barriers, which is the case already today. OpenSG clusters are driving transparently and efficiently tiled projection screens and multi-projector units, including several CAVE systems.

OpenSG can also digest dynamical changes to the scene graphs: All scene relevant data exist in several *aspects* that are replicated among the render clients. Changes are logged in a *change list* that helps to synchronize the data periodically, typically once per frame.

OpenSG uses very consistently the *node-field paradigm*: A ***single field*** is an atomic piece of data, e.g., an integer, float, string, or a reference, etc. A ***multi field*** is a (dynamic) uniform array of single fields, i.e., an array of integers, or of floats, or of strings etc. A ***field container*** is very much like a class in object oriented programming, or like a record in a relational database: It is just a list of several named fields, which can be single or multi fields. The scene graph itself is a tree made up of nodes. A ***node*** is a field container with fields parent, children, and core. The single field parent and the multi-field children are references to other nodes, and the single field core references a node core. A ***node core*** is a field container that contains actual data, e.g., a transformation matrix, or some geometry that is to be rendered. The separation between (lightweight) *node* and (potentially heavy) *node core* is important for multiple instancing of the same geometry for, e.g., all the identical chairs in a theatre.

```
 1 osg-getroot                  % push scene graph root
 2 /Transform osg-corednode     % create node+core
 3 (10,0,10) osg-translate      % change transform
 4 dup                          % stack = root trans trans
 5 /Cylinder osg-primitive      % create node+core
 6 dup begin                    % push cylinder scope
 7    /sides      20  def       % fake dict
 8    /height     15  def       % fake dict
 9    /botRadius  2   def       % fake dict
10 end                          % pop cylinder scope
11 osg-addchild                 % cylinder child of trans
12 osg-addchild                 % trans child of root
```

Figure 5: *Creation of a Cylinder primitive in GML and insertion into the OpenSG scene graph. The osg-translate operator can be called at any time to move or animate objects.*

3.3. The Combination of GML and OpenSG

The combination has two parts, (i) exposing the OpenSG API to the GML language, and (ii) integrating the GML runtime engine into the (larger) scene graph engine. Both parts are only quickly sketched here, for details see [Ger05].

All functionality in the GML comes from operators. Only two higher-level data structures are built in, (heterogenous) arrays and dictionaries. A dictionary is a list of (name,token) pairs, where the token can either be an atomic value, or refer to an array or another dictionary. Note the striking similarity to the node-field paradigm in OpenSG from above. So the idea was to treat OpenSG field containers in the GML as ***fake dictionaries***. This could even be realized in a generic way with the *reflectivity* capability of OpenSG: Every field container type registers itself at startup with a ***field container factory*** that records the type, numerus (single/multi), and name of the fields in the container. The VRML-like field

© The Eurographics Association 2005.

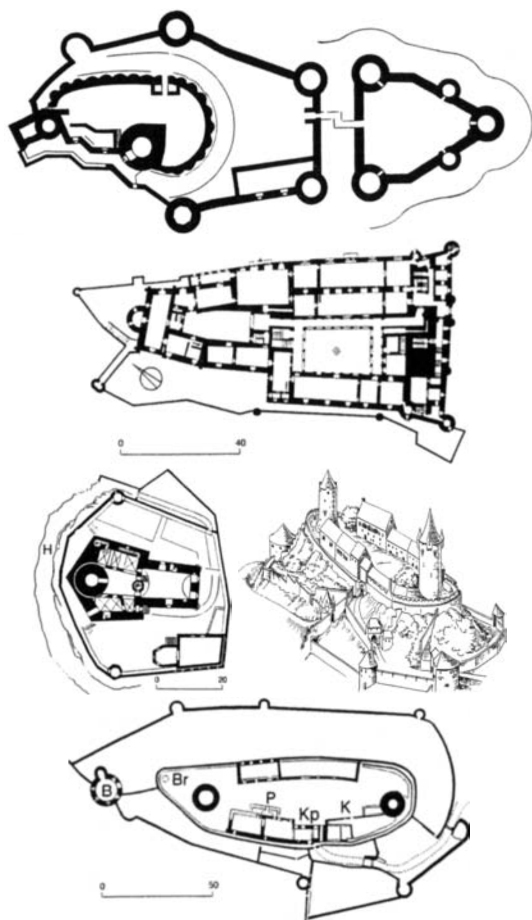


Figure 6: *Ground layout of typical castles adapting to the local ground topography. Examples from [Koc00].*

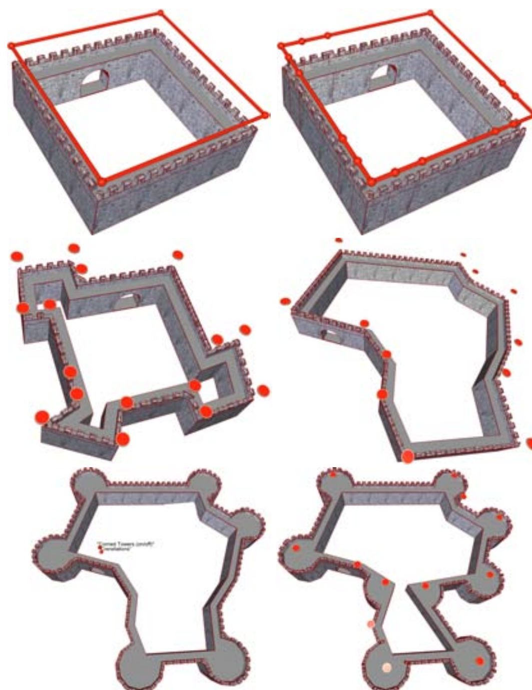


Figure 7: *Castle wall editor. In* ***insertion mode*** *corner points can be added by clicking on the red polygon gizmo (1a,b). In* ***move mode*** *they are represented by red discs that can be interactively dragged until the desired shape is reached (2a,b). In* ***tower mode****, towers are automatically inserted so that all portions of the wall can be defended well (3a,b).*

container type Cylinder for instance, a node core, can be understood as a GML fake dictionary with entries sides, height, and radius. It can now be created and integrated in the scene graph with a GML code snippet as shown in Fig. 5. Note that this technique makes it possible to translate VRML files *fully automatically* to GML+OpenSG; but this will be described in detail in a separate paper.

The main GML shape representation are ***combined B-reps***. They can represent both polygonal and smooth free-form shapes in the same data structure using one *sharp/smooth* flag per mesh edge: Faces with a smooth edge are rendered as Catmull/Clark subdivision surfaces [HF01]. The GML interpreter operates only on a single mesh. In principle this is not a limitation since a mesh can contain an arbitrary number of connected components (3D objects). But all these objects live in the same global coordinate space, so the only way to *move* one object is to modify the positions of its vertices. – This problem has been solved by allowing the GML to switch the current mesh. An OpenSG scene graph can therefore now contain any number of separate cB-rep meshes, stored in geometry node cores of type BRepCombinedFC (field container). Any such node can be made current, and may be interactively modeled, at runtime. The low-level integration of combined B-reps into OpenSG asserts that all mesh changes are even propagated through a *cluster* of render client. So all client PCs compute and display a consistent tesselation of the visible combined B-rep meshes even if they are modified.

4. The Castle Construction Kit

The new possibilities from the last section are a perfect match to the technical requirements from section 1.1 for authoring/presentation software for CH content. Domain-dependent modeling tools are defined with respect to a domain. As example domain we chose *medieval castles*, and our *Castle Construction Kit* (GML-CCK) was realized as part of another diploma thesis [Ber05]. – First we had to understand the domain. We found valuable material on castles in literature on architecture. Very helpful was the book from Koch on *construction forms*, where all the building elements and components from different era and styles are ex-

© The Eurographics Association 2005.

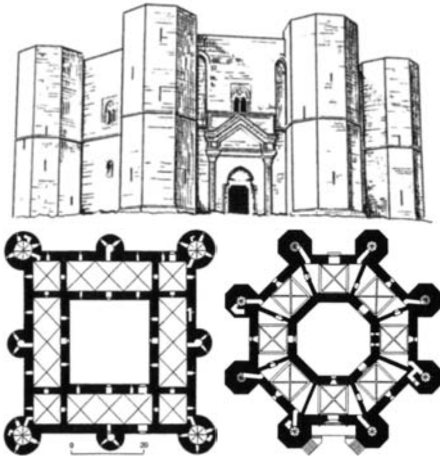


Figure 8: *Staufer castles, e.g., in Sicilia, have a very clean structure. Images are from [Koc00].*

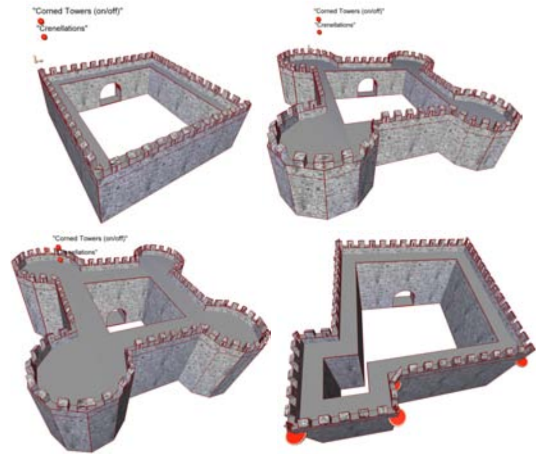


Figure 9: *Very simple Staufer castle with a few mouse clicks.*

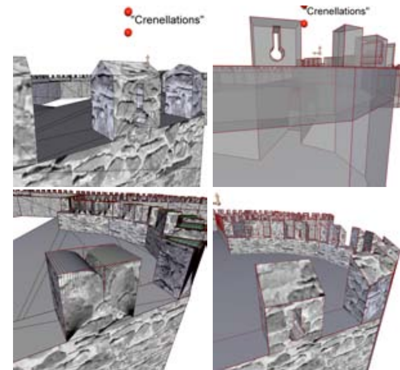


Figure 10: *The **crenellation** style is a wall attribute.*

plained [Koc00]. Some of its pencil illustrations are reproduced here to show the input material we had: Perspective drawings (Fig. 3) and ground plans (Fig. 6).

Our idea was to create modeling tools that permit to quickly produce 3D look-alikes that are not necessarily precise, but that contain the same structure as the original. Interestingly, interest on castle (re-)construction was also expressed from the *fantasy role playing* community. The workflow is depicted in the cover illustration on the first page: (a) historical original castle (Scharfeneck from [Koc00]), (b) adaptation for role playing from [Rad98], (c) 3D look-alike, and (d) tesselation detail. Historically, castles have developed out of simple wooden ring walls on a small hill (*Motte* in German, see Fig. 14).

Most important for the shape of a castle is the shape of the landscape. The location can greatly support a fortification. Height isolines can often be recognized in the ground plan of a castle adapting to a hill, see images Fig. 6 (3b,4a) in row 3 (right) and row 4. But at first, to keep things simple, we chose to start with castles in the plane.

4.1. Castle Wall Editing

As suggested by ground layout examples (Fig. 6) a good first representation of the castle walls is a closed simple *polygon*: A sequence of straight wall segments is connected with elements such as corners, towers, semicircular towers (with open gorge), that are placed in the polygon vertices. Which connection to build depends on the rules of warfare: For the defenders it must be possible to strike every portion of the wall, e.g., by bow and arrow. This is in fact a geometrical problem, which we have built into our wall editor. A typical wall editing session is shown in Fig. 7.

The wall editor uses gizmos to clearly indicate the different editing modes. A ***gizmo*** is a 3D object that is artificially set into a 3D scene to represent a certain operation. The wall editor provides an insertion gizmo (red closed polygon), a motion gizmo (points as discs), and an option gizmo that presents a 3D menu with captions floating space, using balls as switches (Fig. 7, 3a).

4.2. Re-usable Interactive Tools

As it turns out a whole number of objects can be suitably represented by polygons, e.g., arkades, houses, and palisades. So it is sensible to further develop and provide more sophisticated polygon editing modes, as all entities that are based on polygons will immediately benefit from improvements in the polygon editor. From the user's point of view the re-use of existing 3D gizmos/editors is highly desirable as well, since it reduces the learning effort for the user interface.

So the challenge is to design re-usable gizmos/editors in a uniform but flexible, customizable fashion, suitable for many different elements: Care must be taken to avoid erro-

© The Eurographics Association 2005.

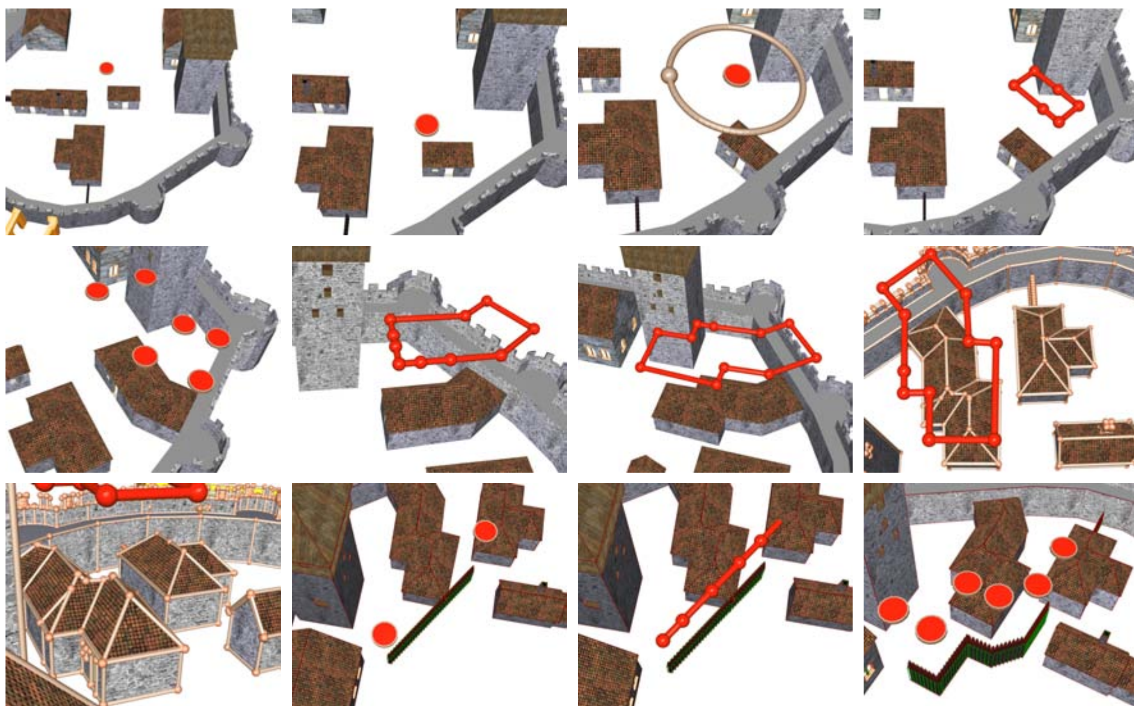


Figure 11: *Creation of houses and palisades using variants of the polygon editor. A new house is created from a list of available house types (1a). It can be moved (1b) and rotated (1c) as a whole prior to editing the outline polygon (1d). Row 2: Adaptation of the ground polygon by alternating insertion and movement actions. A palisade is created very much like an arkade (3b-d).*

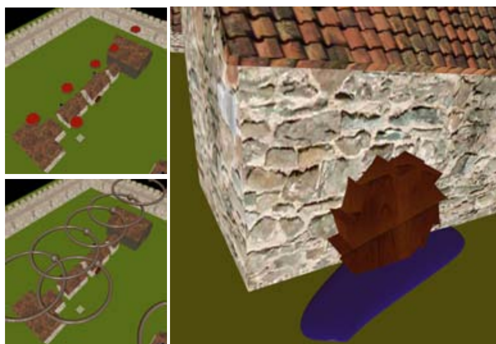


Figure 12: *Movements become efficient only with a scene graph. It permits to animate elements like a water wheel.*

neous states during editing and, therewith, user frustration. Any editor for simple polygons should always avoid self-intersections (8-shape). The arkade gizmo in Fig. 4 additionally needs to avoid very acute angles and very short edges, i.e., the freedom to move the points must be suitably limited. The wall editor in Fig. 7 must additionally take into account the wall width, and the feasible wall length depends on whether it has no, one, or two towers at the ends.

The house and palisade editors are shown in action in Fig. 11. Although the house appears to be classical rectangular, its ground polygon can in fact be freely edited. The roof is constructed fully automatically (2d,3a) from the *straight line skeleton* of the house polygon [AA96]. Interestingly, the roof is created with the same extrudestable operator as the window tracery in [HF04]. The house editor adds two more operations, move and rotate, to the polygon editor that made not much sense for the wall polygon. The move operation is also customized, since it checks whether a house is going to penetrate a wall, in which case the house snaps to the wall and aligns with it.

Note the great benefit of using OpenSG at this point: The move and rotate operations are realized only on the scene graph level, which means that also large portions of the geometry can be moved without overhead. We have used the scene graph also on the object level to add a constantly turning water wheel to a mill, see Fig. 12.

4.3. GML Dictionaries as Element Classes

Recall that our original goal was an *extensible* architecture for domain-dependent modeling tools. The danger in a collaborative environment, especially in an *open CH* context, is that a huge bunch of incompatible tools may result since

© The Eurographics Association 2005.

```
 1 dict dup !wall begin
 2     /polygon            :polygon     def
 3     /cornedTowers       0            def
 4     /rotateMidpoint     (0,0,5)      def
 5     /crenellationStyle  /style-4     def
 6     /wallWidth          2            def
 7     /wallHeight         5.0          def
 8
 9     /gizmo              { }          def
10     /model    CastleConstructionKit
11                /castle_wall get      def
12     /model-update       {
13         /construction clearmacro
14         model gizmo
15     } def
16
17     Model begin
18         /current-wall :wall def
19     end
20
21     /construction newmacro def
22     model gizmo
23 end
```

Figure 13: *Castle element class in GML: Semantic information about a wall is collected in a dictionary that contains static data as well as functions to create, modify, and update the wall. The 3D geometry of the wall and the gizmos is created in line 22 by generic functions operating on the local wall data thanks to* begin, end *in lines 1, 23.*

the tool developer community is so heterogeneous. This can only be avoided with *interface standards*, 'interface' understood in the API sense, with respect to GML code.

We have solved the interface problem by using a unique feature of the GML, namely the fact that it is a functional language that can not only store literal data, but also functions in a dictionary. As mentioned before a GML dictionary can behave like a *class object* in the OOP sense, even as a class with a dynamic set of members. – A generic API for interactive procedural elements was quickly found; in the simplest case it consists of only the three functions gizmo, model, and model-update shown for the wall in Fig. 13. This function also enters the wall dictionary as the current-wall in the Model dictionary, where it is globally visible (line 18).

Note that the wall is in fact a special case, since it has no single gizmo, but a mode dependent gizmo. The gizmo member of the wall is by default set to the empty gizmo: The interaction mode function can dynamically replace the empty gizmo function by the gizmo function for the chosen interaction mode! – Examples are shown in Fig. 11!

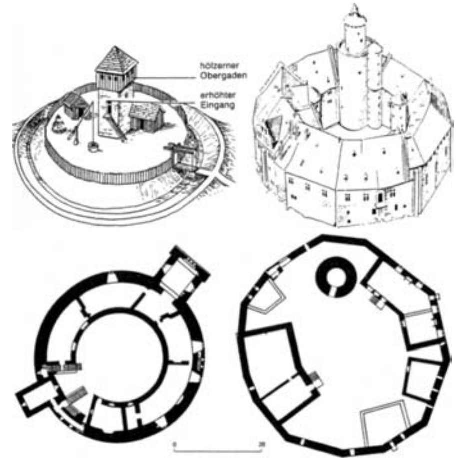


Figure 14: *Origin of castles, from [Koc00]: Wooden ring wall that was gradually transformed into a stone castle.*

5. Conclusion and Future Work

We have presented the first working prototype of our *Castle Construction Kit*. It is very rudimentary, especially compared to what is possible in *Stronghold2*, but we believe that it exhibits already all the features that are essential to achieve what is *not* possible using games technology, namely to create ***open libraries*** of ***CH specific*** modeling tools that bring ***full modeling capabilites*** to a CH 3D-presentation, but can still be rendered with multi-threading ***on a cluster*** with a rendering engine that supports ***custom render node types***.

With our architecture models *and* modeling tools can be exchanged, in fact our models contain their own customized 3D modeler. These fundaments laid directly lead to a long wish-list of improvements, only to mention:

- Differentiate attributes per sub-element (wall, tower)
- Integration of terrain heightfields
- Better appearance, so far only shape was in focus
- Systematically reduce number of authoring mouse clicks
- High-precision construction tools for advanced users, including AutoCAD-like tools for polygon editing.

There are also several long-term goals we will pursue. First, we would like to understand the castle domain better and come up with an *exhaustive* list of sophisticated intelligent building elements that permit a much better match between reality and look-alike. Second, we would like to develop methods to match the model to given data automatically, or at least semi-automatically. Our third goal is particularly fancy: We would like to *drag* a complete castle *interactively* over a hilly landscape, and the castle shall immediately adapt to its new location so that the 'spirit' of the original castle is preserved, but also the laws of medieval warfare are respected, so that a plausible castle results in each time step.

© The Eurographics Association 2005.

References

[AA96] AICHHOLZER O., AURENHAMMER F.: Straight skeletons for general polygonal figures in the plane. *Proc. 2nd Annu. Internat. Conf. Computing and Combinatorics* (1996), 117–126. 7

[BC*04] BALZANI M., CALLIERI M., ET AL.: Digital representation and multimodal presentation of archeological graffiti at Pompei. In *Proc. VAST 2004* (Brussels, Belgium, 2004), Chrysanthou Y. et al., (Eds.), Eurographics Association, pp. 93–103. 2

[Ber05] BERNDT R.: *Automatische Codegenerierung mit der GML (in German)*. Master's thesis, Institute of Computer Graphics, Braunschweig Technical University, Germany, 2005. 5

[BFH05] BERNDT R., FELLNER D. W., HAVEMANN S.: Generative 3d models: A key to more information within less bandwidth at higher quality. In *Proc. Web3D '05* (Bangor, UK, 2005), ACM Press, pp. 111–121. 3

[Col] Collada project website. collada.org. 2

[DD*04] DUGUET F., DRETTAKIS G., ET AL.: A point-based approach for capture, display and illustration of very complex archeological artefacts. In *Proc. VAST 2004* (Brussels, Belgium, 2004), Cain K. et al., (Eds.), Eurographics, pp. 105–114. 2

[Ger05] GERTH B.: *Generative Scene Manipulation in OpenSG*. Master's thesis, Institute of Computer Graphics, Braunschweig Technical University, Germany, 2005. 4

[Hav05] HAVEMANN S.: *Generative Mesh Modeling*. PhD thesis, TU Braunschweig, Germany, 2005. 2

[HF01] HAVEMANN S., FELLNER D. W.: A versatile 3d model representation for cultural reconstruction. In *Proc. VAST '01* (New York, USA, 2001), ACM Press, pp. 205–212. 5

[HF04] HAVEMANN S., FELLNER D. W.: Generative parametric design of gothic window tracery. In *Proc. VAST 2004* (Brussels, Belgium, 2004), Cain K. et al., (Eds.), Eurographics, pp. 193–201. 3, 7

[HJA02] HOFFMANN C., JOAN-ARINYO R.: Parametric modeling. In *Handbook of Computer Aided Geometric Design*. Elsevier, 2002, ch. 21, pp. 519–541. 2

[Koc00] KOCH W.: *Baustilkunde : das Standardwerk zur europäischen Baukunst von der Antike bis zur Gegenwart*, 22 ed. Bertelsmann, Gütersloh, 2000. 3, 5, 6, 8

[Lor05] Lord of the rings: The battle for middle earth. Electronic Arts Inc, 2005. www.eagames.com. 3

[MMK04] MÜLLER G., MESETH J., KLEIN R.: Fast environmental lighting for local-pca encoded btfs. In *Proc. CGI 2004* (June 2004), IEEE, pp. 198–205. 2

[Pra04] PRATT M.: Extension of iso 10303, the step standard, for the exchange of procedural shape models. In *Proc. SMI04* (Genova, Italy, June 2004), pp. 317–326. 2

[Rad98] RADDATZ J.: *Armorium Ardariticum, eine DSA-Spielhilfe*. Fantasy Productions, Erkrath, 1998. 6

[Set05] The settlers: Heritage of kings. Ubisoft Entertainment (BlueByte), March 2005. www.ubi.com. 3

[Str05] Stronghold2: The ultimate castle sim. Firefly Studios, 2005. www.fireflyworld.com. 3

© The Eurographics Association 2005.