Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Graphics 28 (2004) 497–507
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Combining polygonal and subdivision surface approaches to modelling and rendering of urban environments

A.M. Day[a,*], D.B. Arnold[b], S. Havemann[c], D.W. Fellner[c]

[a] School of Computing Sciences, University of East Anglia, Norwich NR2 2DF, UK
[b] University of Brighton, UK
[c] Braunschweig University of Technology, Germany

Abstract

The economic production and the interactive rendering of complete reconstructions of populated urban environments are technically difficult tasks. 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 determine efficient interactive display strategies through the use of level-of-detail and culling techniques. Thus a modeller that knows the operator is creating houses can use this information to simplify the user interaction, guide the operator, and to create models that build in optimizations when attempting real-time rendering. In this paper we discuss the way in which polygonal and multi-resolution surface techniques can complement one another in the modelling of urban environments. We also draw more general conclusions which apply to other software systems that share the same objective.

Keywords: Display algorithms; Viewing algorithms; Curve, surface, solid and object representations

1. Introduction

The Charismatic research project [1] 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 [2–4].

Technically this is 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. The problem is then magnified when we need to include further object types such as people, plants and other items in the scenes.

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 optimize for real time

*Corresponding author. School of Computing Sciences, University of East Anglia, Norwich NR2 2DF, UK. Tel.: +44-1603-592604; fax: +44-1603-593344.
E-mail address: amd@cmp.uea.ac.uk (A.M. Day).

0097-8493/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2004.04.005

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 optimize 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 renderer, uses the structural information from the modeller to efficiently optimize interactive display through the use of model/domain specific level-of-detail and culling techniques. Thus a modeller that knows the operator is creating houses can use this information to simplify the user interaction and guide the operator, and to create models that build in optimizations when attempting 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 [2],
(2) Multi-resolution surface modelling [5].

In this paper we discuss the way in which these techniques complement one another in the modelling and rendering of urban environments.

2. Modelling approaches

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 large quantities to populate the scene [6–10]. We have aimed to render these houses in high detail with windows, doors, and also with interiors and furniture. 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. The generic 3D modeller approach, using mesh optimization where possible, and then applying one or more level of detail techniques to all or parts of the model often needs 'fine tuning' for optimum results. Initially this was attempted by writing plugins for 3D Studio Max and Realimation that provided methods for rapid creation of houses and the attachment of level of detail mechanisms to the results. In both cases this exercise highlighted the difficulty of gaining access to the modeller's geometry for manipulation at a sufficiently low level since only the modeller's exposed API was available to the plugin. It also made the implementation of other modelling speedup techniques such as the automatic creation of texture maps of the models for level of detail extremely complex and inefficient. To overcome these problems we created an 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 do not produce visible artifacts. The basic approach was to create a modelling program, the Shell Modeller, using the scenegraph. A shell is 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 combined 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. In a similar way additional modules were designed to operate together, each addressing the problems of modelling a specialized class of objects and using that information to improve user interface and the efficiency of the resulting models. In addition the method has been extended to additional classes of objects such as arches, windows and columns [4].

2.1.1. Modelling buildings in the shell modeller

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. Modelling starts by creating a single shell that is the root node of the model, and all other shells are attached as 'children' of this root shell. Shells attached to the upper or lower faces of the root shell form the extra floors or cellars of the house. The structures used to make the shell are Walls and Planes. So a shell with an interior will consist of six Walls, each with two Planes. The Plane is the lowest level object in the model that the user can modify to change the size or shape of a shell. Fig. 1 shows a root shell with another shell attached to one wall.

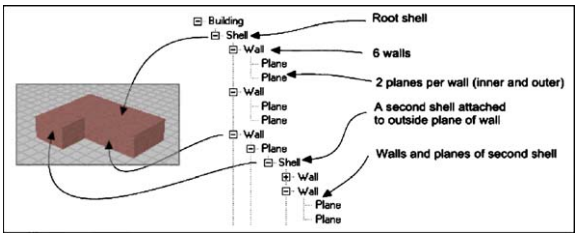

Fig. 1. Data structure layout of a root shell with another shell attached.

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. When a new shell is needed, it is created directly on a selected wall of an existing shell. It can be moved across the face of its parent wall, but not away from it (unless it is specifically detached from its parent wall). Shells can be resized, moved, and, in some cases, rotated on their parent wall (see Fig. 2). There are three types of shell that can be created in the modeller:

(1) *Shells with interiors*: These form the main structure of the house, with the interior shells being created automatically from the external dimensions and specified wall and ceiling thicknesses. By default, when a shell is connected to another shell the wall partitioning the shells is removed. This default is controllable (i.e. the partition can be 'enabled' if preferred without further modelling).
(2) *Empty shells*: These have no interiors and are useful for creating features such as timbering, chimneys, and other solid structures, and for creating roofs. In situations where it is certain that the inside of a building will not be required then the exterior shell could be modelled using an empty shell. Such empty shells could of course be replaced at a later date if the interior became important to have available.
(3) *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. The inside faces of the opening are created automatically between the inner and outer plane, and will adjust themselves to fit if the wall depth is changed.

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. If necessary other fine detail on the house can be achieved by importing and attaching geometry from third party software such as 3D Studio Max or from our Charismatic plugins to the Modeller (e.g. the Window and Column modellers [2]).

Imported geometry attaches to walls in the same way as shells, and can be manipulated using the same controls where applicable (see Fig. 3). Geometry imported from the Window Modeller also contains methods to create openings in the wall to which the window is attached, which are implemented automatically when the window is attached to a wall.

2.1.2. Level of detail

All imported geometry is automatically subject to level of detail control when it is imported into the modeller, and can be acted upon by the rendering scenegraph. The basic level of detail mechanism used is geometry switching to an allied texture map of the object, with a new method of blending the transition between the two states without perceivable 'popping' effects. Details of this are given in Section 3.3.1.

2.2. Multi-resolution surface modelling approach

As presented so far, Charismatic efficiently exploits the object semantics for creation of objects which are specific to the domain of Cultural Heritage [3]. This bottom up approach attempts to cover shapes which are most often used in reconstruction, but is limited in that it cannot be exhaustive. The remaining shapes must be imported from external sources. The most important, and in several respects, most problematic source for 3D models are textured polygon meshes. Meshes represent the smallest common denominator for 3D objects and can be exported by most commercial modellers. 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

Fig. 2. The reconstruction of a house (Wolfenbttel) demonstrates the flexibility of combining shells.

Fig. 3. A street in Wolfenbttel (left) and its reconstruction (right).

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. It offers tools to convert polygon meshes into multiresolution objects, i.e. objects that permit a change of model resolution at runtime.

The multiresolution toolkit provides two multiresolution object representations:

- progressive meshes,
- combined Breps.

Both methods are presented in the following from a user's point of view. More in-depth information can be found in the documentation that comes with the Multiresolution Toolkit [13].

2.2.1. *Reduction of detail: progressive meshes*

The basic device for reducing the triangle count of over-sampled 3D objects is mesh simplification. This is accomplished by means of a simplification algorithm, which rates all edges of a mesh according to an importance measure and sorts them in a queue [12]. The least important edge is removed using an edge collapse operation removing one vertex and two triangles at a time. Edges in the neighbourhood of the simplified region are re-evaluated, and the simplification algorithm iterates until no valid collapse remains in the queue.

A progressive mesh is the most simplified version of a mesh, the base mesh, together with the sequence of simplification operations, the split sequence [1]. Once the split sequence is computed, the level of detail can be freely adjusted in either direction issuing the appropriate vertex split (refinement) or edge collapse (coarsening) operations.

The multiresolution toolkit contains an authoring tool for progressive meshes. It basically proceeds by loading an object, computes the multiresolution hierarchy, and finally stores it in a .pm file which contains the base mesh together with the split sequence. Simplification is essentially an automatic process and requires no manual interaction. The simplification rate is actually quite high, about 10 K vertices per second on current PC hardware. The authoring tool also permits mesh smoothing [1], in order to remove noise from the surface. Noisy data are a problem with all kinds of 3D model acquisition procedures, such as 3D scanning and photogrammetry.

Progressive meshes work fairly well and are the method of choice for over-sampled 3D objects. Such models are typically found when automatic methods are applied, or when no care was taken to produce lean models, for instance when exporting from a general purpose modeller. The most serious drawback of progressive meshes is that they cannot respect the intended structure of a shape. Resulting distortions are naturally most notable with high reduction rates, typically if more than 80–90% of the geometry is removed (see Fig. 4).

Progressive meshes are known to work comparatively badly with synthetic (that is not obtained from 3D scanning), highly regular, structured models. This is especially the case if the models are clean in a sense that not many vertices are in fact redundant. In order to test the applicability of progressive meshes, we have therefore carried out a number of tests with very clean models. The result was that with most of the models, a reduction of up to 50% of the data was practically imperceptible, and did not introduce at all unacceptable distortions. With the lamp post model in Fig. 4, the 50% reduction can practically only be seen with the sphere lamps that are flat-shaded and subject to environment mapping. Also with the column in Fig. 5, a 60% reduction makes hardly a perceivable difference. The column was created as a lean model optimized for Internet rendering.

In summary, progressive meshes have proven more powerful than expected but their use requires 'fine-tuning', as the reduction rate a given object permits may vary significantly. The best way to proceed is to import an object to the simplifier, and to examine it at different levels of detail. With a little bit of experience, a feasible reduction interval can quickly be determined. Another useful aspect is that a snapshot of a model at any chosen resolution can be saved as a mesh, from which a

Fig. 4. Viewpoint lamp post with 20%, 30%, 40%, 50%, 60%, and 100% quality.

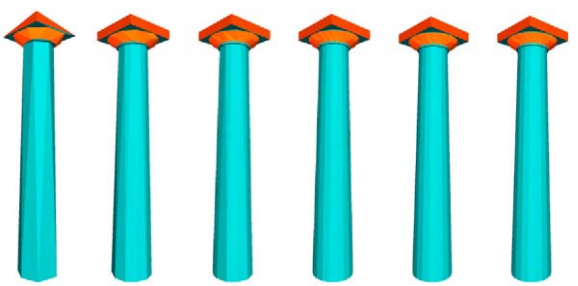

Fig. 5. Column with 10%, 20%, 30%, 40%, 50%, and 100% resolution.

progressive mesh can be computed anew (see Fig. 6). This is important if a mesh with 600 K vertices is to be used only with 100 K vertices max, for instance.

2.2.2. Extensions to the basic concept

Experiments have been carried out to use progressive meshes with data from actual laser range scanning.

However the problems with such models were overwhelming (see Fig. 7). After trying different methods for mesh repairing and cleaning, we finally got to the point where we needed a slightly more general simplification data structure to cope with highly non-manifold geometry. Although the problems and concepts for solving them have become clear now (non-manifold collapse), this is now the subject of future work [13].

2.2.3. Increasing the detail: subdivision surfaces & combined BRep meshes

A drawback of progressive meshes is that even at highest level of detail, the resolution cannot be higher than the resolution of the original mesh. This restriction is overcome by combined BRep meshes, where edges of a polygonal mesh can be made smooth. In such smooth regions, the mesh is refined, i.e. detail is synthesized, in a view-dependent way, approximating a smooth freeform surface. Technically, Catmull/Clark subdivision surfaces are used for smooth parts of the surface, and standard polygonal techniques otherwise. This way, combined BReps bridge the gap between free-form and polygonal surfaces. It is also important to note that with Combined BReps, the resolution can be increased beyond the actual resolution of the input model.

(a) *Catmull/Clark subdivision surfaces*: Subdivision surfaces represent a way of defining free-form geometry through a process of recursive subdivision of polygon meshes according to a set of pre-defined rules. In theory, this process creates a smooth curved limit surface if infinitely repeated. In practice, a few levels of recursion are sufficient, i.e. less than six (Fig. 8).

The great advantage of subdivision surfaces is that virtually any polygonal control mesh can be used as the control mesh of a free-form surface. The faces may be triangles, quadrangles, or they can in fact have any degree, and any number of edges can meet at a vertex. So, in particular any valid manifold mesh can be a control mesh.

(b) *Combined BRep meshes*: The true power of subdivision surfaces is revealed if they are used together with sharp edges, i.e. if every edge contains a sharpness flag which can be toggled to make this edge either a smooth or a sharp (crease) edge. The ability to create crease edges significantly adds to the expressiveness of the data structure, as they can be used in various ways.

The idea is basically that in smooth surface regions, i.e. with only smooth edges, Catmull/Clark subdivision is applied. In sharp regions, the mesh is treated completely like a polygonal mesh. The interesting applications, however, are those where both methods are used together. In Fig. 9, an ornamental object is depicted which is essentially a rounded object, but some edges are marked sharp. It can be clearly seen how a crease curve in the surface follows the red edges.

Rounded shapes can be generated from very coarse input meshes through subdivision surfaces. They are therefore well suited for ornamental design. This was demonstrated e.g. with some architectural models at the VAST conference [5]. Historic buildings often contain free-form elements in distinct places, which reveal their complexity with close-up views during interactive display. This technique also blends nicely together with

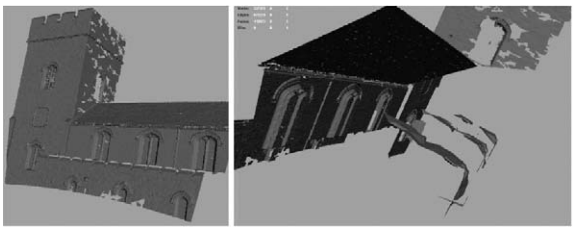

Fig. 6. Triangulation of a point-cloud obtained from laser-range-scanning.

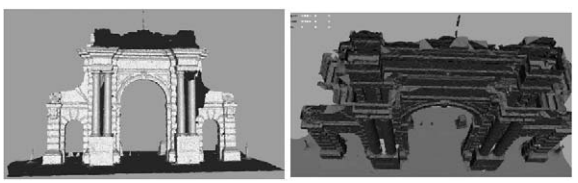

Fig. 7. Occlusion problems, reversed surface parts, and unconnected regions.

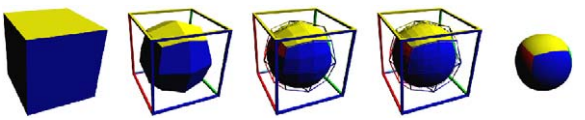

Fig. 8. A cube with three levels of recursive subdivision, final object uses normals per vertex instead of face normals.

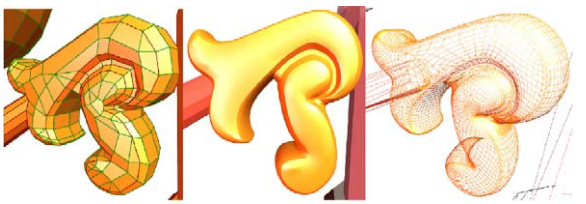

Fig. 9. Consecutive sharp edges form uniform BSpline crease curves.

shell-based building design and the level of detail technique employed in the Charismatic renderer.

The marking of edges can make a big difference, and requires some understanding of how subdivision proceeds. The rules are simple:

- Vertices are either smooth, creased, or corners, depending on whether less than 2, exactly 2, or more than 2 of their edges are sharp;
- A face that has any smooth edges gets subdivided;
- A face with only sharp edges may have crease vertices (called a sharp face);
- A face with a ring cannot have smooth edges.

It turns out that the most important thing to learn when working with combined BReps is how the mesh topology and the sharpness flags act together to define the shape of the resulting surface.

However, users quickly get acquainted with the ways that different effects can be obtained, due to the immediate response from the authoring tool. Fig. 10 shows an undesired effect when all control edges are made smooth. Especially the u-shaped front face of the object creates problems, because it has a non-convex shape, and subdividing it creates undesired self-intersections in the surface. But note that subdivision still works even in such cases. The whole process is very stable and virtually never crashes with manifold meshes.

Pitfalls such as that in Fig. 10 can be remedied if the user starts from a completely polygonal model, where all edges are sharp (i.e. red), see Fig. 11. When gradually adding smooth edges, the shape changes can be readily tracked. Especially with the horizontal edges in the top of the ornament, a nice rounded profile is created. Impressive shiny effects are possible with reflective materials which use environment mapping (see Fig. 12). A more extensive collection of examples for what effects can be achieved with BReps, demonstrating their usefulness for the architectural domain can be found in [5].

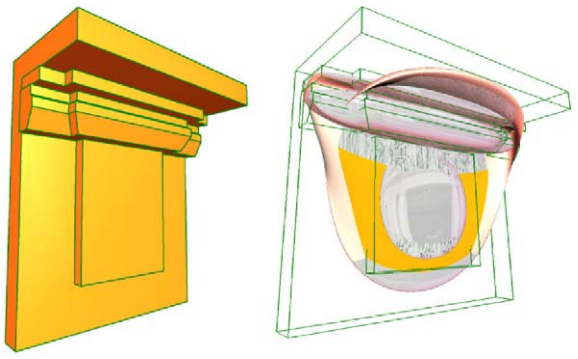

Fig. 10. All edges made smooth.

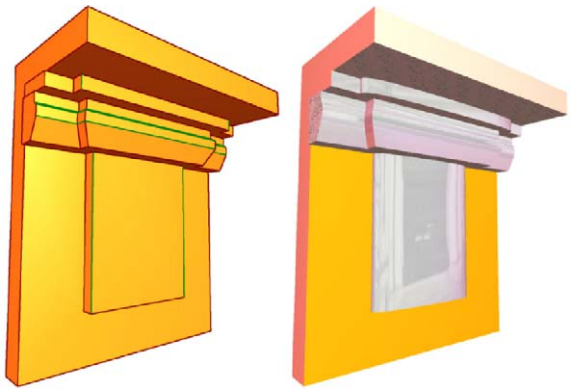

Fig. 11. Just a few smooth edges in an otherwise polygonal model.

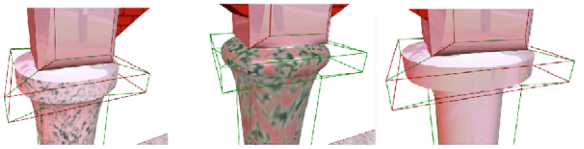

Fig. 12. Different effects when changing sharpness flags of just a few edges.

2.2.4. *Generative modelling*

The interactive modelling approaches that have been presented still lack one vital ingredient to significantly increase efficiency and that is automation.

Ironically, the highly engineered domain of 3D modelling still requires enormous amounts of manual intervention to create appealing objects. The predominant trend is to create more elaborate, specialized and highly configurable tools to perform specific modelling tasks in more convenient, thus more time-efficient ways. In spite of this, the production of virtual worlds still is enormously cost-intensive. One goal of Charismatic is to develop concepts which allow cost efficiency to be improved, in order to make virtual reconstructions more economic in general. The Charismatic Shell Modeller is a great example for this evolution of tools. The problem remaining also with great tools is that they must still be used: the tool is interactively called, dialogue boxes must be filled, objects need to be picked etc. The basic idea to give support for the automation of GUI applications is to provide an interactive application with a 'geometric spreadsheet calculator'. For this purpose, the generative modelling language (GML) has been developed as a formally well-defined back-end. It is a very simple, but nevertheless complete programming language, i.e. it supports loops and conditional decisions. It is in fact so simple in syntax that the code generation can be hidden from the user and happens in background. The analogy

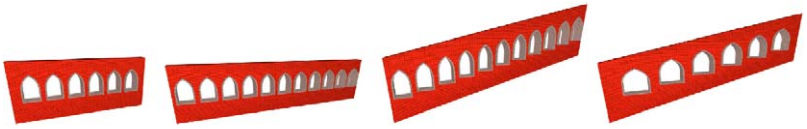

Fig. 13. GML example: with automatic construction, changing a few parameters such as window width or the distance between them, or the slant of a wall, can have a significant effect.

to spreadsheets comes from the possibility to use variables instead of concrete values in dialogue boxes, and these variables can be computed from other variables. A dialogue to create a 3D box (or shell) after pushing the OK button may internally issue any of these equivalent commands, the second of which depends only on a single parameter 'size' and actually creates a cube as a special case of a box, so it is effectively a specialized new 'cube' tool, created from a more general, basic 'box' tool.

The GML has been provided with an interface to the combined BRep mesh representation for automated alterations to the control mesh. This facilitates the creation of macros for tedious manual tasks such as marking edges to make them sharp or smooth, e.g. to switch between the different column capitals in Fig. 12, or for different versions of a wall with gothic windows as in Fig. 13.

2.3. Modelling approaches combined

A simple way of regarding the combination of both approaches would be to say that it makes no sense to attempt to create arbitrary shapes solely through the use of shells. At the same time, it is not reasonable to represent entire buildings through 'anonymous' meshes, be they progressive or combined or whatever, simply because the represented objects completely lose their semantics. Therefore, the approaches are complementary rather than competing. The truth is that the potential of both approaches working together goes beyond just adding features lists.

The shell approach has proven to have far more potential than just to represent an agglomeration of rectangular boxes. Shells can be distorted in various ways and combined in different angles. A set of basic but powerful constraints helps to maintain consistency. When shells are attached to each other, they still remain editable and can effectively be adapted to changes in the scene. There is also indication that the shell approach offers potential that goes beyond the already realized functionality. In a more abstract way, they can be seen as a general device to efficiently represent spatial relationships between cells, such as containment and attachment. Following this approach, one can think of facilities for hierarchical grouping of shells, where containment and attachment mechanisms are propagated throughout an entire shell hierarchy.

As shells give an effective structure to buildings, they are basically the 'bones', while the 'flesh' comes from all the irregularly shaped geometry that is needed to create beautiful facades and ornaments. Many items, such as churches, fountains, bridges, stairways, have structural properties that make them very different from domestic buildings. The combined BRep meshes with their inherent multiresolution property are an ideal platform for all kinds of complicated detailed constructions that are small in extent but are of a distinct aesthetical importance. Still, they work best when used in a 'local' fashion, so they need some type of 'guiding' structure to trigger level-of-detail, visibility, etc. In other words they unfold their potential best when used as attachments to shells.

Progressive meshes on the other hand are the method of choice when 'anonymous' real-world data are to be used. They are the ideal devices to handle massive amounts of 3D data. Given the recent development in laser-range scanning and photogrammetric methods [11], it can be assumed that such input will drastically increase in near future. Progressive meshes offer the possibility to show today's findings with a virtual reconstruction of how things looked like in the past side by side.

Generative Modelling finally offers the potential to automate many tasks in the modeller, to procedurally create desired detail on the fly (e.g. ornamental structures), to use variables to relate quantities and to perform actions based on user-defined calculations. Their built-in interface to the mesh data structure can be used to create combined BReps directly within the Charismatic modeller.

3. Combination of rendering approaches

Only a combination of different real time rendering strategies will yield satisfactory results. Detailed urban environments are especially demanding because of the variety of different objects and shape representations in the scene.

3.1. Techniques

An important goal of real-time rendering is output sensitivity: to send only those geometric primitives to the

graphics hardware that make a perceivable contribution to the image. However 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 real time renderer. These methods are of course in direct correspondence to the modelling tools and object representation explained earlier. They are as follows:

- *Use of the scenegraph*: In order to minimize 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.
- *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. Our polygonal renderer uses a quick bounding box check to determine if an object is outside the view frustum or not.
- *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 [1–3].
- *House LOD*: The building data structure supports a specialized 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.

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.

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.

- *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 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.

- *ROAM*: A realtime optimally adaptive mesh (ROAM) [15] 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.
- *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. With the landscape in the renderer having 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.

3.1.1. Combination of techniques

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 while 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 III, 733 MHz, 512 MB RAM, GeForce3 graphics card).

3.2. Rendering progressive meshes

A progressive mesh 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 200 K vertices per second on up-to-date PC hardware, thus it is much faster than the computation of the simplification itself.

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.

3.3. 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 Fig. 8. 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 [14]. Fig. 14 shows a typical effect of an increase in quality; no popping artefacts are introduced.

3.3.1. 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.

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. Fig. 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.

3.4. 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

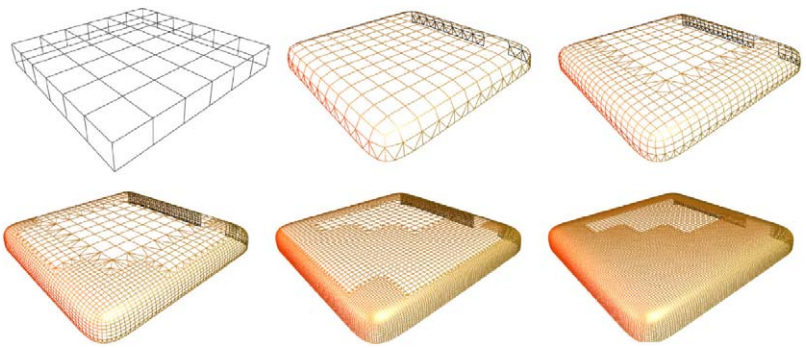

Fig. 14. View-dependent tessellation with increasing level of detail, the 66 quad faces on the top are subdivided into 2424 patches.

Fig. 15. Blending from combined BReps (quality 1.0 and 0.8) to progressive meshes (quality 100%, 80%, 50%, 30%).

sort of efficient encoding that makes the modelling very easy equally forms the basis for optimized 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). However 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.

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 also applies to the rendering. 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.

As another extension the concept of a shell as a box could be generalized to free-form shells: some buildings have curved features, such as windmills or towers of a castle. For such special cases, the box geometry could be entirely replaced by a combined BRep, instead of just attaching it. The set of constraints would have to be extended to retain the advantages of constrained attachment of other shells to a free-form shell.

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, by executing the appropriate GML code when the house is entered. This 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 for further work are:

- content distribution over the Internet,
- A *standardized* scene graph engine (e.g. http://www.opensg.org) for the commercial distribution of cultural heritage content to third-party software companies,
- truly immersive historic experiences through cluster-based stereoscopic rendering.

5. Conclusions

The key to success in rendering complex reconstructions is to find the best way to represent as much structural information as possible. The underlying idea in both approaches, 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 size 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 the modelling and representation rather than try to optimize bad models.

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

Acknowledgements

The support of D. Fellner and S. Havemann from the German Research Foundation (DFG) under grant numbers Fe 431/4-1 and Fe 431/4-3 is greatfully acknowledged. We would like to thank our colleague Chris Mills for his help in producing this paper.

References

[1] http://www.charismatic-project.com.
[2] Browne SP, Willmott J, Wright LI, Day AM, Arnold DB. Latest Approaches to Modelling and Rendering Large Urban Environments, Eurographics, UK, 2001.

[3] Wright LI, Willmott J, Day AM, Arnold DB. Rendering of large and complex urban environments for real time heritage reconstruction. Proceedings of VAST 2001, Athens.
[4] Flack PA, Willmott J, Browne SP, Day AM, Arnold DB. Scene assembly for large scale reconstructions. Proceedings of VAST 2001, Athens.
[5] Havemann S, Fellner D. A versatile 3D model representation for cultural reconstruction. Proceedings of VAST 2001, Athens.
[6] Bridges A, Charitos D. The architectural design of virtual environments. Proceedings of the Seventh Conference on Computer Aided Architectural Design Futures. Munich, Germany: Kluwer Academic Publishers; p. 719–32.
[7] van Leeuwen JP, Wagter H, Oxman RM. A feature based approach to modelling architectural information. Proceedings of the CIB W78 workshop, 1995.
[8] Achten HH, van Leeuwen JP. A feature-based technique for designing processes, a case study. Proceedings of the Fourth Conference on Design and Decision Support Systems in Architecture and Urban Planning, 1998.
[9] de Vries B, Jessurun AJ. Features and constraints in architectural design. Proceedings of DETC'98, 1998 ASME Design Engineering Technical Conference.
[10] de Vries B, Jessurun AJ, Kelleners RHMC. Using 3D geometric constraints in architectural design support systems. Proceedings of the Eighth International Conference in Central Europa on Computer Graphics, Visualisation, and Interactive Digital Media, 2000.
[11] Garland M, Heckbert PS. Surface simplification using quadric error metrics. Computer Graphics 1997;31:209–16.
[12] Hoppe H. Progressive meshes. Computer Graphics 1996;30:99–108.
[13] Technical Report TUBSCG-2002-02.pdf available from http://www.cg.cs.tu-bs.de/PubArc/tr.
[14] Havemann S. Interactive rendering of Catmull/Clark surfaces with crease edges. The visual computer 2002;18(5/6): 286–98.
[15] Duchaineau MA, Wolinsky M, Sigeti DE, Miller MC, Aldrich C, Mineev-Weinstein MB. Roaming terrain: real-time optimally adapting meshes. IEEE Visualization, 1997; 81–8.