Advances in Computer and Information Sciences'98
U. Güdükbay et al. (Eds.)
IOS Press, 1998

A Survey of Interactive Realistic Walkthrough Techniques in Complex Graphical Environments

Alper Selçuk, Uğur Güdükbay and Bülent Özgüç

Bilkent University
Department of Computer Engineering and Information Science,
06533 Bilkent Ankara Turkey
e-mail: {aselcuk, gudukbay, ozguc}@cs.bilkent.edu.tr

Abstract. One of the biggest problems in computer graphics is displaying huge geometric models in interactive frame-rates. Such models exceed limits of best graphics workstations. A lot of work has been done for achieving the required frame-rates in architecture, simulation, computer-aided design (CAD) and entertainment applications. A survey of methods that enable walkthrough of complex graphical environments is presented in this paper. Methods that enable walkthrough of complex graphical environments vary from very simple ideas, such as using texture mapping simple geometric objects to very complex algorithms for level-of-detail management. They can be used together to get combined advantage of each other.

1 Introduction

In recent years, the use of computers in design has become extremely important. As the price of computers and peripherals decrease, more and more companies have started to buy powerful workstations for their design process. This trend led many software companies to develop serious and complicated programs for CAD. Although CAD software helps engineers a lot in designing and viewing individual components, engineers need to view combined shots of their design in order to verify the correctness of design. Only combined shots are also sometimes not enough, walkthrough of the entire model is required. Considering the size of a very complex model containing millions of polygons, only one single camera shot can require days of time to render even if state-of-the-art computers are used [3].

Architects today use computers to design and view buildings. The size of such architectural models can also be huge. Customers may want to view the building on computer. Just showing rendered pictures or a film prepared by moving a virtual camera inside the building model on a predetermined path sometimes may not satisfy customer. Customer may want to walk inside the model interactively. For architectural walkthroughs, image quality is the most important property. Image quality depends on texture quality, positioning of light sources and rendering quality. Real-time restrictions can be slightly loosened for the sake of quality. Considering the size of architectural model, the textures used for walls, floor and furniture and the number of light sources in a building, the power of today's computers are again exceeded.

The problem for a walkthrough in a complex environment is due to the number of polygons to be displayed exceeding the number that the computer can render for

each frame. It is trivial that current computer graphics systems cannot meet required graphics throughput for the above examples. The situation will probably not change in the future because as the graphics systems evolve and get more powerful, the size of models also grow larger and larger.

This paper presents a survey of methods that enable walkthrough of complex environments. In Section 2 capabilities of human visual system is explained together with how these restrictions can be utilized in walkthrough applications. In Section 3 different approaches for walkthrough in complex environments is discussed. The last section gives conclusions.

2 Motivation

For decreasing the load of graphics systems in complex tasks, we must consider the capabilities of human visual systems. Without any knowledge about this, graphics output requirements will certainly exceed the limits of current graphics systems. Since humans will view generated images, satisfying only the human visual system is enough for simulating reality in a virtual environment. This can reduce the workload of graphics systems considerably and enable real-time walkthrough of million-polygon models.

Different applications require different kinds of realism, however they can all be categorized into two classes [6].

i) Perceptual Realism: An image is *perceptually realistic*, if the image is the same as its virtual form. That means a viewer synthesizes a mental image similar to that is synthesized by the virtual camera. The degree of perceptual realism depends on the kind of rendering technique used for generating the image. Perceptual realism is especially important for architectural and art applications.

ii) Visceral Realism: An image is *viscerally realistic* if the viewer believes that everything in the image is real. The most important way of increasing visceral realism is complexity. Every detail must be considered and made carefully. For entertainment applications and flight simulators visceral realism is vital.

Although human visual system is complex, it is not perfect. We must consider the thresholds of human visual system to achieve both kinds of realism with low cost. Perceptual limits of the human visual system depends on the parameters such as *visual acuity* [15] and *field of view* [2]. There is also close connection of the human visual system with the parameters of the display system, such as *latency* [4], *frame rates*, *refresh rates* and *temporal resolution* [25], which effect the perceptual limits of the human visual system.

3 Methods for Walkthrough in Complex Environments

Methods that enable walk-through in complex environments vary from very simple ideas to very complex algorithms. Figure 1 gives a taxonomy of these methods. Algorithms of varying complexity can be used together to get combined advantage of each other.

3.1 Reducing Geometric Complexity

The main idea of geometric complexity reduction is to get a realistic image without modeling and rendering all the scene.

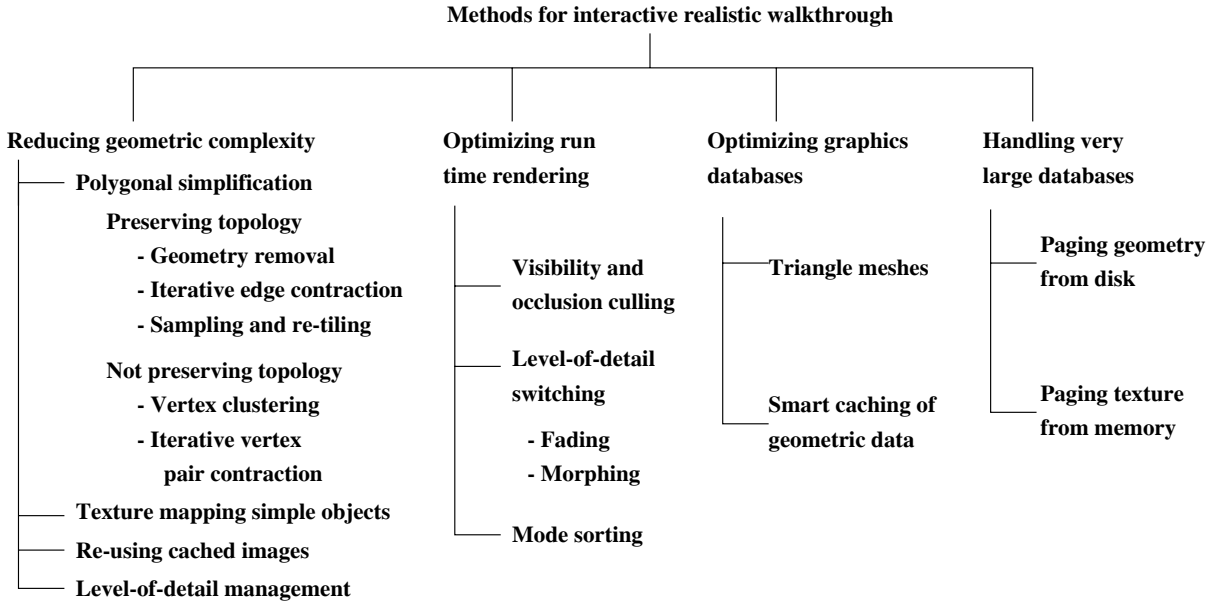

Figure 1. A taxonomy of methods for walkthrough in complex environments.

3.1.1 Polygonal Simplification

Polygonal simplification is the process of transforming a three-dimensional (3D) poly gonal model into a simpler version containing less polygons. The transformation tries not to change the original shape and appearance of the model. In this way, rendering load will be reduced considerably. Storage required for storing the model will also be reduced, which simplifies the management of data between disk and memory. Furthermore, transmission of simplified large models over networks will be faster than original models. If the simplification is good enough, then each stage of the graphics pipeline will have a workload that can be handled in real-time.

Polygonal simplification algorithms are categorized into two groups: algorithms that preserve the topology of the original model and algorithms that do not preserve the topology of the original model. There are four basic methods for simplification.

i) Geometry Removal: These algorithms removes vertices or faces from the original model order to obtain a simplified version of it. Vertex decimation algorithms iteratively select vertices for removal, remove all the adjacent faces of these vertices, and retriangulate the resulting holes. The removal process continues until a user-specified error range is reached [20]. Geometric optimization [11] is another algorithm in this category which obtains an optimized geometry by combining coplanar polygons. These algorithms do not change the topology of the model.

ii) Vertex Clustering: This algorithm places a bounding box around the original model and divides it into a grid. With each grid cell, the cell's vertices are clustered together into a single vertex, and the model faces are updated accordingly. The quality of approximations is often very low since the user has no control over the simplification process. This is not a topology-preserving algorithm [19].

iii) Iterative Edge (or Vertex-Pair) Contraction: These algorithms iteratively select edges and contract them into a vertex. They differ in the selection of the edge to contract [12, 18]. These algorithms also preserve the topology. There is another algorithm proposed in [9] which uses a generalization of the edge contraction, called *pair contraction*, which also may contract nearby vertices not connected by edges. In this way, topology may not be preserved, which is an advantage for rendering systems.

iv) Sampling and Re-tiling: This algorithm varies from the other three in that it tries to generate a model from scratch. First a sampling algorithm samples the geometry of the original model by either taking a number of random points from its surface or by overlaying the model with a 3D grid and sampling each box of the grid. The algorithm then tries to create a simple model that fits to the sampled data. This algorithm does not provide a fine-grained control on the simplified model; so it performs well on smooth curved surfaces rather than models containing sharp discontinuities. This algorithm also preserves the topology of the original model. To achieve this, rarely the vertices of the original model are also included in the simplified model [23].

3.1.2 Geometric Level-of-Detail (LOD) Management

The main idea of geometric LOD management technique is representing objects that do not contribute much to the scene with less primitives. For example, an object with many polygons that is far away from the viewer requires roughly same amount of time as an object with the same number of polygons that is nearer. The far object will cover a smaller portion of the scene than the near object. Creation of multiresolution models requires that each object must be modeled with different geometric representations and surface properties. They should be organized in the database so that accessing different representations of the object is simple and not time-consuming. Polygonal simplification algorithms can be used to generate multiresolution models automatically.

One important requirement for multiresolution models is the preservation of appearance of objects. There are measures for determining the success of an LOD management algorithm. Measures for image output are more important than measures for topology or geometry. Therefore, an image based error metric is generally used. Once the multiresolution model is prepared, the problem reduces to managing LOD of objects. The management algorithm should consider the time and image quality restrictions. If the rendering time is exceeded, low-resolution models for far objects should be used.

An adaptive display algorithm that can manage LOD according to quality and time constraints is proposed in [8]. The algorithm also manages the complexity of the rendering techniques used. The aim is to find the combination of levels of detail and rendering algorithms for all potentially visible objects that produce the best image possible within the target frame rate.

Luebke and Erikson [16] developed a method, called *Hierarchical Dynamic Simplification*, which models the whole scene as a single polygonal model and constructs a hierarchy based on the clustering of the vertices and then queries this hierarchy dynamically for simplification of different parts of the model as the viewing position shifts. They use the screen area measure for simplifying the nodes of the hierarchy.

Xia et al. [24] developed a similar method which involves statically generating a continuous LOD representation for the object which is then used at run-time to guide the selection of appropriate triangles for display. They utilize image-space, object-space and frame-to-frame coherences for simplifications. This method gives good results when there are a limited number of highly complex objects relatively close to the viewer.

Hoppe [14] describes a framework for selectively refining a triangle mesh according to viewing parameters. He constructs *a progressive mesh* [13] representation of the model, which contains the simplest version of the model together with the edge collapse transformations to obtain the most detailed version. Frames are rendered depending on the viewing position by applying selective refinement to the progressive mesh.

Hierarchical data structures for storing LOD are proposed in [1, 17]. In these tech-

niques, the intermediate nodes of the hierarchy contain simplified data of its children. Leaf nodes store original data. A sample hierarchy is given in Figure 2. In [1], the representation of a scene is selected according to the distance of the scene to the viewpoint and the area that the scene covers on the screen. If the scene is far away from the viewpoint, the root of the hierarchical organization is rendered. As the scene gets closer, more detailed representations are selected. In [17], different search strategies such as depth-first search and best-first search are proposed for selecting the representation.

3.1.3 Texture Mapping

Algorithms explained so far use geometric simplifications of the original model. Replacing the original model with a texture or a colored cube is another method. The contribution of texture mapping to walkthrough is the increase of realism without additional polygonal complexity. A simple polygonal object with a texture can be used instead of a complex polygonal model if it is in a far away position in the screen. In such methods geometric simplification cost is eliminated. Furthermore, the representations are much more simple to render. In [5], a method which replaces cells of the scene obtained by spatial subdivision by colored cubes is proposed. In the preprocessing step, the algorithm partitions the scene using octree subdivision. Each cell of octree is assigned a colored cube. During rendering, geometry in near cells are rendered. For far cells that cover a small number of pixels, corresponding colored cubes are rendered. Although this method causes some visual artifacts such as discontinuities in solid surfaces, it accelerates rendering considerably.

Another algorithm [21] makes use of frame-to-frame coherence by caching images of objects rendered in one frame for possible reuse. The algorithm first creates a Binary Space Partitioning (BSP) tree and partitions the environment. The nodes contain geometric primitives and cached images. Then the BSP tree is traversed for each frame.

3.2 Optimizing Run-Time Rendering

To optimize the run-time rendering in a walkthrough application, visibility and occlusion culling techniques are used to reduce the number of objects to be rendered for each frame. To eliminate visual defects caused by switching between different representations of objects, *fading* and *morphing* techniques are used. Grouping the polygons in terms of texture and material properties also increases run-time rendering performance.

i) Visibility and Occlusion Culling: Users in walkthrough applications usually are in the middle of a huge database. They can only see a very small portion of the database. Therefore, sending all the objects in the database to the graphics pipeline would be wasteful. One trivial optimization to reduce the number of objects to be rendered is culling of the objects to the viewing frustum, which is the chopped-off pyramid containing all the visible objects when the user looks through a virtual camera. Figure 3 shows the virtual camera model. After a viewing frustum is formed, objects are culled against that frustum. Frustum culling calculations can be performed easily if the data is organized as a hierarchical structure. The explicit frustum culling introduces an overhead when a large number of objects are in the frustum. However, for cases when explicit frustum culling prunes object trees, its overhead is compensated.

Clark [7] used an object hierarchy to rapidly cull surfaces that lie outside the viewing frustum. The method explained in [10] takes care of many aspects of visibility computation and accelerates rendering considerably compared to classical z-buffering. It

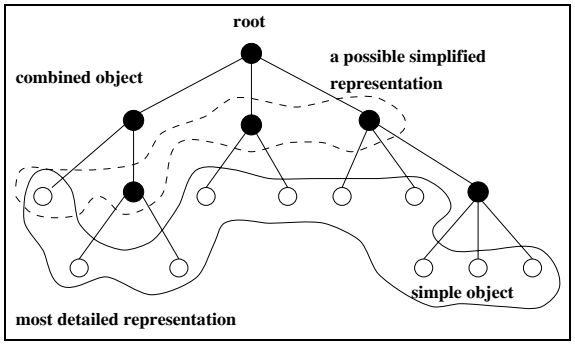

Figure 2. A sample hierarchy.

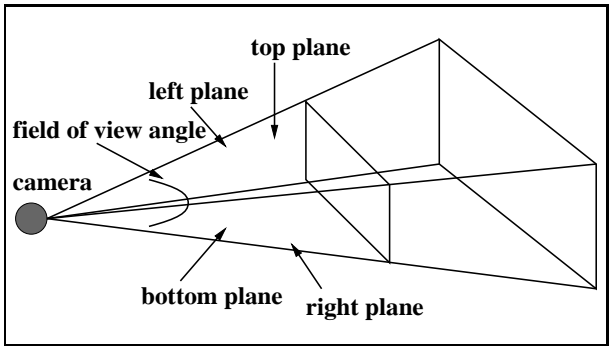

Figure 3. The virtual camera model.

processes the scene in terms of object-space coherence, image-space coherence and temporal coherence. Occlusion culling techniques proposed in [22] take into account large occluders in the database. These techniques are especially important for architectural models. Architectural models generally consists of cells (rooms) and portals (windows, doors, etc.). User in a cell can see only the objects in the cell and objects that are visible through the portals. Different visibility and occlusion culling algorithms can be used together to eliminate more objects.

A visibility and occlusion culling algorithm should not create a bottleneck in the graphics pipeline. Ironically, the better the algorithm culls the less time for culling, since the frame rate increases because of the small number of polygons. For example, if an algorithm eliminates nearly all objects in the scene, then the rendering will take a very short time, and the culling algorithm will require much less time for preparing the polygons of the next frame. Therefore, culling algorithms should be customizable according to the CPU and graphics subsystem power.

ii) Level of Detail Switching: Switching between different representations of objects causes visual defects such as popping. Therefore transitions between different representations should be optimized. There are a number of ways to reduce the popping effect. The first method is called *fading*. Instead of simply switching the models, for a period of time both models are drawn blended together. This reduces the popping, however increases the workload of the graphics system. The second method is *morphing*. One of the objects is morphed to the other object continuously until they are the same. This method has the best visual effect compared to other techniques, however it is difficult to morph between arbitrary models. For time-critical applications the popping effect can be ignored, because these methods require expensive graphics operations.

iii) Mode Sorting: For most graphics workstations changing the mode is very expensive. By mode, the texture and material properties is meant. Therefore, grouping polygons to reduce the number of mode changes is a simple and efficient optimization. It can increase performance incredibly for scenes with many small pieces of textures.

3.3 Optimizing Graphics Databases

Geometric data should be organized in such a way that graphics processors should be able to manipulate the geometric data efficiently. Besides, the arrangement of data in memory should reduce the delays during rendering.

i) Triangle Meshes: Applications for creating geometric models usually are designed to give the designer the easiest way for modeling an object. On the other hand, graphics engines are usually designed and optimized for rendering triangles. Therefore,

triangulating the model as a preprocessing step reduces the rendering time of the scene.

ii) Smart Caching: Another method for reducing the rendering cost, data can be arranged in contiguous memory locations. Also secondary processors for pre-fetching data can be used to reduce delays caused by page-faults. This method becomes especially important when the size of the model exceeds the size of available memory.

3.4 Handling Very Large Databases

Another problem with complex models is that the model can be too large to fit into the memory. This problem has two components. The first one is that the size of the model exceeds the RAM and the second one is that the size of texture does not fit into the texture memory. Both problems must be handled separately.

i) Paging Geometry from Disk: When paging geometric data from disk, full I/O bandwidth should be used. This requires structuring the data so that it can be read in large blocks; preferably being transferred using Direct Memory Access into the application's address space. Paging operation should not affect the frame rate. Therefore, it can be performed synchronously between frames. However the amount of data being transferred between frames can be too small to utilize full I/O bandwidth. To avoid this problem in multi-threaded systems, asynchronous loading can be performed by creating a thread for load operation.

In order to avoid arriving at a point in the scene and not having the correct data to render, the application must predict and adjust timing of loading data. In very large scenes with long visibilities, it is also important to have the data structured so that low-resolution LOD models and textures can be loaded in without having to read all the resolutions of the model or texture.

ii) Paging Texture from Memory: For achieving fast texture mapping, graphics subsystems have their own texture memories that are used for caching textures. Texture memories are much more expensive than the conventional RAM and maximum available texture memory size is much less than RAM. Therefore, management of texture memory is very critical.

Unlike paging data from disk, paging texture cannot be done asynchronously with rendering because on most graphics architectures texture loading share the same data paths as normal rendering. A fraction of the rendering time must be reserved for texture loading. Memory management issues such as fragmentation should also be considered. The simplest solution to this problem is to load textures with the same size.

4 Conclusions

The need for interactive frame-rates in simulation, CAD/CAM, architecture and entertainment applications led many researchers to work in this area. The models for such applications exceed the limits of current graphics workstations. Researchers make use of limitations of human visual system for reducing the number of primitives and the hardware properties of state-of-the-art graphics systems for optimizing the performance.

Methods for walkthrough range from simple ideas, such as texture mapping very simple geometric objects, to very complex level-of-detail management algorithms. Reducing the number of polygons to render without affecting the image quality is the main idea of these methods. For reducing the number of primitives either geometric simplification algorithms or visibility and occlusion culling algorithms are used. Such algorithms reduce the number of polygons per frame to manageable numbers.

References

[1] S. Belblidia, J.P. Perrin and J.C. Paul, "Multi-Resolution Rendering of Architectural Models," *Computer-Aided Architectural Design Featurres'95*, July 1995.

[2] P. Buser and M. Imbert, *Vision*, MIT Press, Cambridge, MA, 1992.

[3] B. Cabral, N. Greene, J. Rossignac and T. Funkhouser, "Interactive Walkthrough of Large Geometric Databases," *SIGGRAPH'96 Course Notes*, Course 35, August 1996.

[4] F.M. Cardullo, "Virtual Systems Lags: the problem, the cause, the cure," *Proc. of the Image V Conference*, pp. 31-42, June 1990.

[5] B. Chamberlain, T. DeRose, D. Lischinski, D. Salesin, J. Snyder, *Fast Rendering of Complex Environments Using a Spatial Hierarchy*, Technical Report UW-CSE-95-05-02, University of Washington, 1995.

[6] K. Chiu and P. Shirley, "Rendering, Complexity and Perception," *Proc. of the 5th Eurographics Rendering Workshop*, Darmstadt, 1994.

[7] J.H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, Vol. 19, No. 10, pp. 547-554, October 1976.

[8] T.A. Funkhouser and C.H. Sequin. "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments," *Proc. of SIGGRAPH'93*, pp. 247-254, August 1993.

[9] M. Garland and P.S. Heckbert, "Surface Simplification Using Quadric Error Metrics," *Proc. of SIGGRAPH'97*, pp. 209-216, August 1997.

[10] N. Greene, M. Kass and G. Miller, "Hierarchical Z-Buffer Visibility," *Proc. of SIGGRAPH'93*, pp. 231-236, August 1993.

[11] P. Hinker and C. Hansen, "Geometric Optimization," *Proc. of Visualization'93*, pp. 189-195, October 1993.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, "Mesh Optimization," *Proc. of SIGGRAPH'93*, pp. 19-26, August 1993.

[13] H. Hoppe, "Progressive Meshes," *Proc. of SIGGRAPH'96*, pp. 99-108, August 1996.

[14] H. Hoppe, "View-Dependent Refinement of Progressive Meshes," *Proc. of SIGGRAPH'97*, pp. 189-198, August 1997.

[15] M. Jones, "Designing Real-Time Graphics for Entertainment," *SIGGRAPH 94' Course Notes*, August 1994.

[16] D. Luebke and C. Erikson, "View-Dependent Simplification of Arbitrary Polygonal Environments," *Proc. of SIGGRAPH'97*, pp. 199-208, August 1997.

[17] P.W.C. Maciel and P. Shirley, "Visual Navigation of Large Environments using Textured Clusters," *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Computer Graphics*, pp. 95-102, 1995.

[18] R. Ronfard and J.R. Rossignac, "Full-Range Approximation of Triangulated Polyhedra," *Computer Graphics Forum (Proc. Eurographics'96)*, 15(3), 1996.

[19] J.R. Rossignac and P. Borrel, "Multi-resolution 3D Approximations for Rendering Complex Scenes," *Modeling in Computer Graphics: Methods and Applications*, B. Falcidieno and T. Kunii, editors, pp. 455-465, 1993.

[20] W.J. Schroeder, A.Z. Jonathan and E.L. Lorensen, "Decimation of Triangle Meshes," *Proceedings of SIGGRAPH'92*, pp. 65-70, July 1992.

[21] J. Shade, D. Lischinski, D. Salesin, T. DeRose and J. Snyder, *Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments*, Technical Report UW-CSE-96-01-06, Microsoft Research, 1996.

[22] S.J. Teller and C.H. Sequin, "Visibility Preprocessing for Interactive Walkthroughs," *Proc. of SIGGRAPH'91*, pp. 61-69, August 1991.

[23] G. Turk, "Re-Tiling Polygonal Surfaces," *Proc. of SIGGRAPH'92*, pp. 55-64, July 1992.

[24] J.C. Xia, J. El-Sana and A. Varshney. "Adaptive Real-Time Level-of-detail based Rendering for Polygonal Models," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 2, pp. 171-183, June 1997.

[25] G. Wyszecki and W.S. Stiles, *Color Science*, John Wiley and Sons, New York, 1982.