Multiresolution Modeling for Fast Rendering

Paul S. Heckbert and Michael Garland
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213-3891, USA
ph@cs.cmu.edu, garland@cs.cmu.edu

Abstract

Three dimensional scenes are typically modeled using a single, fixed resolution model of each geometric object. Renderings of such a model are often either slow or crude, however: slow for distant objects, where the chosen detail level is excessive, and crude for nearby objects, where the detail level is insufficient. What is needed is a *multiresolution model* that represents objects at multiple levels of detail. With a multiresolution model, a rendering program can choose the level of detail appropriate for the object's screen size so that less time is wasted drawing insignificant detail. The principal challenge is the development of algorithms that take a detailed model as input and automatically simplify it, while preserving appearance. Multiresolution techniques can be used to speed many applications, including real time rendering for architectural and terrain simulators, and slower, higher quality rendering for entertainment and radiosity. This paper surveys existing multiresolution modeling techniques and speculates about what might be possible in the future.

Keywords: multiresolution model, level of detail, rendering, simplification, approximation.

The first section of this paper discusses the goals and computational cost of complex scene rendering and explains why the use of a model with the appropriate level of detail is important. Section 2 summarizes the goals of multiresolution modeling and section 3 summarizes several data structures for achieving these goals. The paper concludes with a comparison of these data structures.

1 Rendering Complex Scenes

In computer graphics, we would like to render complex scenes such as terrains, cities, building interiors, molecules, and biological structures as quickly as possible. These might contain millions of geometric primitives (polygons, spheres, etc.).

1.1 Optimized Rendering

Early rendering algorithms performed transformation, clipping, sorting, and hidden line or hidden surface removal. For a scene of n primitives, such algorithms had costs of $O(n^2)$ or $O(n \log n)$. In the late 70's and early 80's, flight simulators capable of displaying several thousand polygons in real time became available, but at a price of several million dollars. Advances in graphics hardware and memory technology have allowed brute force algorithms such as the z-buffer to supplant the earlier, sorting-based algorithms. These advances have sped rendering dramatically, so that it is now possible to draw ten thousand shaded polygons in real time on a \$40,000 workstation.

The time cost of rendering a scene of n primitives with a total screen area of a_c [1] with a simple z-buffer algorithm is $\Theta(n+a_c)$ [2]. The cost is linear in the number of primitives because of transforming and clipping, and linear in the screen area due to scan conversion and shading. When the scene is very detailed and most surfaces project to an area smaller than a pixel, the transformation and clipping cost (n) dominates. Algorithms with costs that are linear in n are an improvement over earlier algorithms, but they are still slower than necessary.

In highly complex scenes, the user cannot see all of the primitives at one time because many of them are either off-screen, occluded, or too small to be seen. If a simple z-buffer algorithm is used, as described above, then each of the primitives in the scene must be transformed and clipped, even if it is way off screen, and each on-screen primitive must be scan converted and (potentially) shaded, even if it is occluded. Off-screen primitives can be culled quickly using hierarchical bounding volumes [4], octrees,

[1] a_c is the sum of screen areas of clipped primitives, ignoring occlusion. It could be much greater than the number of pixels in the image if the depth complexity is high.

[2] Asymptotic complexity notation: $O(f(n))$ is an upper bound, $\Theta(f(n))$ is both an upper bound and lower bound. We discuss only worst case cost in this paper.

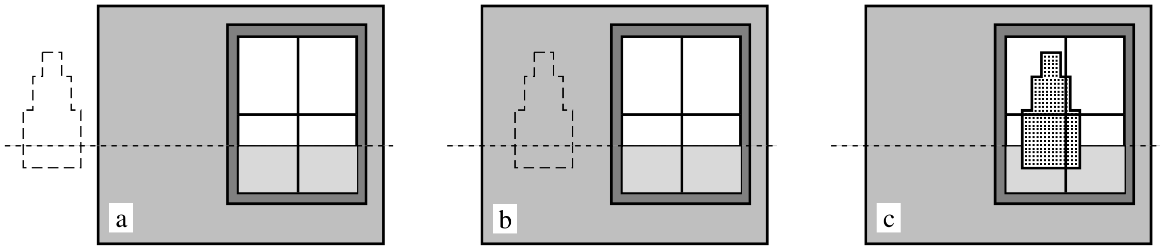

Figure 1: Three views out a window. a) Building is off-screen. b) Building is on-screen, but occluded by wall. c) Building is on-screen, and visible through window.

or other spatial data structures.

Occluded objects can be weeded out either with sophisticated visibility analysis techniques [24] or more brute force z-buffer pyramids [10]. Together, these optimizations would reduce the cost of rendering n primitives with clipped screen area of a_c from $\Theta(n+a_c)$ to $\Theta(n_v+a_v)$ not counting preprocessing, where n_v is the number of visible primitives, and a_v is their total screen area. Note that $0 \leq n_v \leq n$ and $0 \leq a_v \leq a_c$, but the relative sizes of these variables are very scene-dependent.

The third optimization for complex scenes, speedier handling of small objects, is employed by few existing rendering systems. It is the focus of this paper.

Figure 1 shows three views that illustrate the differences between these optimizations of the z-buffer algorithm. The scene is a building that is (sometimes) visible through a window in a wall. Suppose the visible facade of the building is modeled using n polygons, the wall and ground plane are modeled using a small, bounded number of polygons, and the number of pixels in the image is a. The costs of various z-buffer-based algorithms on these three views are:

ALGORITHM	Fig. 1a	Fig. 1b	Fig. 1c
unoptimized	$\Theta(n+a)$	$\Theta(n+a)$	$\Theta(n+a)$
with off-screen culling	$\Theta(a)$	$\Theta(n+a)$	$\Theta(n+a)$
with off-screen & visibility culling	$\Theta(a)$	$\Theta(a)$	$\Theta(n+a)$
with off-screen & visibility culling, & multiresolution modeling (projection)	$\Theta(a)$	$\Theta(a)$	$\Theta(a)$

Figure 1a, where the building is off-screen, is optimized by simple off-screen culling. Figure 1b is more difficult to optimize; visibility culling is required to generate this picture quickly. Figure 1c is the most difficult of all. In the case where $n \gg a$, there are many more polygons than necessary to model the building, and they're almost all visible, so neither off-screen nor visibility culling will render this view efficiently.

What is needed are methods for simplifying an object that has been modeled with excessive detail so that arbitrary views can be rendered quickly, ideally with a cost (not counting preprocessing) proportional to the number of pixels, but independent of scene complexity. Such an algorithm would be optimal on a computer where writing a pixels takes $\Theta(a)$ time[3]. For any particular view, this goal is trivially achievable, since the scene could be modeled as a plane tiled with one rectangular polygon per screen pixel. The challenge is to find a model that works for any viewpoint, while remaining fast and compact.

2 Multiresolution Modeling

Geometric models typically describe each shape in a scene with a single representation. The scale or *level of detail* at which each object is modeled is fixed. Such a model is called a *fixed resolution model*. When rendering with a perspective projection, distant objects project to a small screen area and nearby objects project to a large area. In a complex scene, the dynamic range of these screen areas can be very great. Rendering such a scene using a fixed resolution model can be very inefficient, as seen with figure 1c.

The best method for optimizing the rendering of small and distant objects is *multiresolution modeling*: the description of geometry and surface attributes such as color and texture at a variety of scales. Depending on the screen size of a given object or cluster of objects, the appropriate level of detail within the model would be chosen [4]. The appropriate level for a given view is the coarsest level that looks the same as the finest level. Thus, nearby objects would be rendered using a detailed model, while distant objects would be rendered using a coarse model.

2.1 Applications

Multiresolution modeling has many applications. The primary one is fast display, both for real-time rendering and for high quality images that might take minutes or hours

[3]On a parallel machine with $\Theta(a)$ processors, we might be able to render in constant time.

of compute time. Architectural walkthroughs, flight simulators, scientific visualization, computer-aided design, movie special effects, and virtual reality are natural applications.

In the past, the person who models a 3-D scene has often been the one who runs the renderer. Since this person knows what the camera will see, he or she can model the scene appropriately, including only those details that will be seen.

For 3-D animation, objects that are seen at widely varying scales are often modeled at two or more levels of detail: a "fine" model for closeups and a "coarse" model for distant shots. As the object recedes into the distance during animation, the scene description will often be manually altered to switch from the fine model to the coarse model. This procedure can be automated by including both the coarse and fine models in the scene description, and using a measure of screen size, such as the area of the projected bounding volume of the object, to choose between the levels of detail.

In flight simulators and virtual reality, the person preparing the model is not the person choosing the viewpoints (i.e. the pilot). For these applications, the modeler must include enough detail that the user can move through the whole scene without losing the illusion of reality.

Multiresolution modeling is also useful in radiosity and other global illumination algorithms. In rendering, we determine the visible surfaces within a viewing pyramid and create a picture of them, while in radiosity, we determine the visible surfaces within a hemisphere and integrate them. These tasks are very similar.

Radiosity algorithms subdivide each input polygon into many *elements*. Early radiosity algorithms had a cost that was quadratic in the number of elements because they used a fixed subdivision and they calculated the amount of light reflected between each pair of elements. These algorithms wasted most of their time computing insignificant light transfers between distant objects. The hierarchical radiosity algorithm uses adaptive subdivision instead: when gathering light into each element, it subdivides distant polygons coarsely and nearby polygons finely [11]. With this improvement, the algorithm's cost is linear in the number of elements, but it is still quadratic in the number of polygons, since pairwise subdivision starts with the given polygons. Thus, the algorithm is fast only for simple scenes consisting of a few large polygons. This is unacceptable for complex scenes.

The quadratic cost term of hierarchical radiosity could be eliminated, yielding an algorithm whose complexity would be linear in the number of polygons or better, if multiresolution modeling were used, clustering distant objects and treating them as a single unit. Rushmeier et al. have recently employed multiresolution models for radiosity, but their model creation system was not automated [20].

2.2 Model Use

Selection of the appropriate level of detail during rendering is easy, requiring only hierarchical bounding volumes and fast estimates of screen area. If levels of detail are selected discretely, however, this will cause visible artifacts in the spatial or temporal continuity of images. Experience has shown that consistency is often more important than correctness in computer graphics. Level-switching artifacts can be eliminated by smoothing the transitions using linear interpolation of geometry and color.

2.3 Model Creation

Creation of a multiresolution model is quite difficult. Although multiresolution modeling is an old idea, most existing such databases have been created by hand. In flight simulator and architectural walkthrough systems that employ multiresolution modeling, laborious manual database preparation is still required, to the best of our knowledge [29, 9, 8]. Renderman can render multiresolution models but it supplies no automatic tools for generating them [26].

The principal challenge of multiresolution modeling is to find a set of algorithms that can take a complex scene description as input, including both geometry and surface attributes such as color and texture, and automatically generate data structures that allow rapid rendering of the scene from any viewpoint. For greatest flexibility, the system should allow arbitrary input (e.g. a set of polygons with no topological information) and not assume that the input comes with a hierarchy. It is most important that the rendering be fast and the appearance of the scene be preserved, but it is also desirable that the preprocessing time and memory requirements be low.

2.4 Preservation of Appearance

Quantifying the "preservation of appearance" objective can aid in the development of algorithms. The real measure of appearance is the raster image output by the renderer. This is more important than precise preservation of topology or geometry. We would therefore like an *image error metric* that measures the overall difference between two images. This metric should measure the difference between an image $f(x, y)$ rendered using the fully detailed input model and an image $\hat{f}(x, y)$ rendered using the multiresolution model (the approximation). We'd like the two images to be indistinguishable. Ultimately, human viewers are the judges, so the best error metric would entail a model of the human visual system, a very complex topic. Useful results can be obtained with much simpler error metrics, however. These can be viewed as crude approximations to human perception.

A simple starting point is the sum of squared distances in RGB color space between corresponding pixels:

$$E(f, \hat{f}) = \sum_{x,y} \left\| f(x,y) - \hat{f}(x,y) \right\|^2$$

This error metric can be improved by adding differential weighting for the color channels, nonlinear sensitivity to radiance, and spatial filtering. Any multiresolution modeling data structure that is developed should be validated either with perceptual tests using human viewers, or with a good image error metric.

3 Multiresolution Data Structures

For rendering, a model can be regarded as an abstract data type that supports queries of the form:

> **what does this object look like when viewed from a given viewpoint, with a given resolution?**

Any fast, compact data structure for such queries would suffice as a multiresolution representation. We discuss the following six data structures as possible candidates:

1. image pyramids,
2. volume pyramids,
3. texture and reflectance,
4. pictures from multiple angles,
5. ray space, and
6. polygonal models.

Several of these are rather speculative.

3.1 Image Pyramids

In two dimensions, the most natural multiresolution model is the image pyramid. Image pyramids are ubiquitous in image processing and computer vision [18], and are also widely used to optimize texture mapping [28, 12]. Image pyramids are an attractive multiresolution model because they are so easily resampled. Unfortunately, they are limited to 2-D.

3.2 Volume Pyramids

More natural as a 3-D multiresolution modeling data structure is the 3-D volume pyramid. Volume pyramids are very helpful for fast volume rendering [21], but as a surface representation, they are bulky and crude.

3.3 Texture and Reflectance

Texture and reflectance models are a form of multiresolution modeling. They model the visible effects of fine-scale variation in geometry and surface attributes that are too small to be modeled using geometry. Texture mapping is commonly used to model features whose geometry is smaller than a pixel but whose visible patterns are bigger than a pixel, and reflectance models describe features whose patterns are much smaller than a pixel.

In pictures or animation encompassing a wide range of scales, the choice of representation should be allowed to vary from frame to frame and from pixel to pixel. When flying over a terrain, for example, mountains in the far distance are best modeled as a textured plane, and the appearance of the trees on the mountain are best modeled statistically, in the reflectance model. In the near distance, what was texture (the mountain) should become geometry, and some of the larger features influencing reflectance (the trees) should become texture. Finally, in a closeup, the trees become geometry.

This idea has been proposed by Perlin [16], Kajiya [15], and others, but has never been implemented in a general way. The best progress along these lines has been made in generating bidirectional reflectance distribution functions (BRDF's) from geometry [2, 7, 27] and in smoothing the transitions between BRDF's, bump mapping, and displacement mapping [1].

3.4 Pictures from Multiple Angles

In architecture, initial design is typically done by sketching a building from multiple viewpoints. When we watch film or video, we are seeing a sequence of still images of objects from different viewpoints. Such representations suffice, in a practical sense, to define a 3-D shape. Hence the idea to represent an object not with a set of surface primitives, but with a set of pictures.

This approach has the obvious advantages that the representation is of the same form as the output of a renderer (a picture) and that image pyramids could be used, allowing quick extraction of an image of the desired resolution. The major disadvantage is that the appearance of the object from arbitrary viewpoints is not directly available; in the process of generating the pictures, information is lost.

Intermediate views can differ from the chosen views because of either occlusion or specular reflection. In a scene where sunlight shines directly through a tunnel, for example, only certain views see the sun, so if those views were not chosen, the system would have difficulty generating accurate intermediate views. And in a scene containing mirrors, only certain views see a reflection of the light sources, so again, intermediate views would be difficult to interpolate correctly.

Approximate intermediate views can be generated automatically if the correspondence between pixels of the chosen views is known, and the correspondences can be derived if the z-buffers of the chosen views are available [3]. This technique does not solve the complications caused by occlusion and specular reflection, however, so there are large unsolved problems to make this approach viable as a general multiresolution modeling data structure.

3.5 Ray Space

Another approach to multiresolution modeling is to treat an object's appearance in terms of ray queries, the type of queries performed in a ray tracing algorithm. A ray query takes a ray and returns the color traveling backward along that ray. Existing data structures for fast ray queries require huge memories, so this approach does not seem as promising as the use of textures and polygons. This approach is attractive, however, because it provides a unified, high level abstraction that allows us to blur the distinction between geometry and surface attributes such as BRDF's.

3.6 Polygonal Models

The final approach to multiresolution modeling that we consider, polygonal models, has received the most work, so we discuss it in the greatest detail.

The principal challenge when using a polygonal model for multiresolution modeling is *simplification*: automatically converting a detailed model into a simpler one that faithfully represents the underlying object. We seek algorithms that will minimize both the number of polygons in the simplified model and the error of the approximation.

Simplification algorithms differ greatly depending on the topology of the polygonal model. The simplest are curve models, consisting of a sequence of points or line segments (not really polygons at all). Next in complexity are mesh models, which consist of a network of polygons forming a single, continuous surface. The most general class of polygonal models are *polyhedral models*, where arbitrary topology is allowed. The latter class is the most relevant to multiresolution modeling.

3.6.1 Curve Simplification

Numerous algorithms for approximating one piecewise linear curve with another have been developed [6]. It is possible to find the least squares optimal m-segment approximation to an n-segment curve in time $O(n^3 logm)$ using dynamic programming [14]. Unfortunately, this is too slow for use on complex curves, and it does not appear to generalize to surfaces. Curve simplification algorithms may be of some guidance in our search for surface simplification methods, however.

3.6.2 Mesh Simplification

The aspect of polygonal simplification that has received the most attention is the simplification of surface meshes. Such models are commonly generated from digital sampling of real world objects. The data tend to be dense and redundant, so they can typically be drastically simplified without significant loss of fidelity. We consider grids with rectangular topology first, then height fields, and finally general meshes.

If the mesh is a grid with rectangular topology then a natural simplification technique is to low pass filter the data and then discard every other row and every other column from the grid, performing what is called "decimation by 2" in signal processing. Williams proposed this as a multiresolution modeling technique both to reduce the time needed to transform polygons and to reduce the need for antialiasing [28].

Another area of research is the generation of compact triangulations from digital terrain data and other height fields [17, 22]. Given a regular grid of height samples, the task is to construct a triangle mesh that closely approximates the actual surface with a small number of vertices. Typically, these algorithms are constructive; they begin with a minimal set of points and then add new points until the error of the approximation is below some threshold. Various criteria are used to select which points to add. Some of these algorithms are quite slow. For example, Polis and McKeown's algorithm required 18 hours to achieve a 40-to-1 simplification of a 4,000,000 point terrain. These applications compute the simplified model off-line, however, so for them, preprocessing speed was much less important than accuracy and simplification.

A broader problem is the simplification of general meshes. The typical goal here is to digitize a real world object and construct a compact surface description of it. In [5], DeHaemer and Zyda present an adaptive subdivision algorithm that fits polygons to a set of samples. This algorithm combines surface reconstruction and simplification; it constructs a simple surface directly from the data.

Other algorithms require a mesh as input. Schroeder, Zarge, and Lorensen [23] propose a decimation algorithm. They iteratively remove unimportant points from the mesh, performing local retriangulations to preserve the surface. Turk [25] describes a related approach. He selects a set of points on the surface that will become the vertices of a new mesh and uses point repulsion to achieve good coverage of the surface. A new triangulated mesh is generated by combining the old and new vertices. The

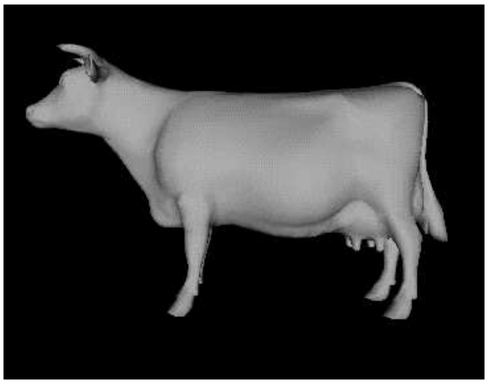

Figure 2: Original cow (5804 triangles)

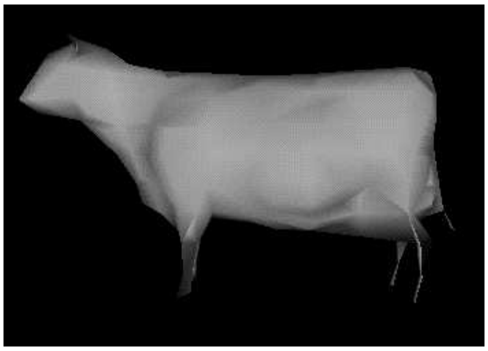

Figure 3: Simplified cow (658 triangles)

Figure 4: Original Beethoven (4998 triangles)

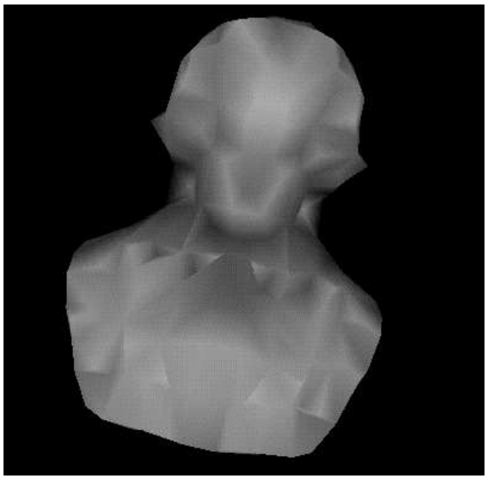

Figure 5: Simplified Beethoven (652 triangles)

old vertices are then iteratively deleted, using local retriangulations to preserve the topology of the surface. Most recently, Hoppe et al. [13] present an algorithm for optimizing fairly general surface meshes. They cast the problem in terms of minimizing an energy function that captures the conflicting goals of mesh simplification and error minimization.

3.6.3 Polyhedral Simplification

Rossignac and Borrel [19] have made one of the few efforts to address the simplification of general, polyhedral models with arbitrary topology. Their motivation is to speed interactive viewing of complex objects, so they seek a minimal set of polygons and lines that suggests a shape to the user. Given a polyhedral model that has been triangulated, they subdivide its bounding volume into a grid of boxes. All vertices within each box are merged together into a new representative vertex. A simplified model is then synthesized from these representative vertices by forming triangles according to the original topology.

This is essentially a signal processing approach: the model is filtered, resampled, and reconstructed. As with all sampling algorithms, aliasing can arise. One weakness of the algorithm is that averaging vertices removes high-frequency details that might have significant importance (features on a face, for example). Another weakness is that the results are not invariant to rigid body motion of the input model; if the model is rotated or translated, the output model ripples like a point-sampled image.

Figures 2–5 show results from an algorithm based on Rossignac and Borrel's. Figure 2 shows the original model of a cow and figure 3 is the result of simplification. With this model, the results are good; viewed from a distance the models are fairly similar. The second example is a bust of Beethoven, figure 4, whose simplified version, figure 5, illustrates the loss of important detail. A better algorithm would use more polygons in areas of high surface curvature and fewer polygons in areas of low curvature.

4 Conclusions

Most current rendering algorithms are inefficient when rendering very complex scenes. Their cost is linear in scene complexity, and this is unacceptable when the complexity is very high. When given a scene with many more surface primitives than pixels, z-buffer algorithms waste a lot of time transforming and clipping objects smaller than a pixel that have negligible impact on the final picture. Using multiresolution modeling it may be possible to render scenes in time proportional to screen area but independent of scene complexity.

The six data structures for multiresolution modeling that we have discussed are evaluated below:

Image Pyramids. Image pyramids are very good for planar and smoothly curved surfaces, but they do not represent real 3-D features well.

Volume Pyramids. Total brute force. The results will look blurry or blocky unless a huge memory is available.

Texture and Reflectance Models. Texture and reflectance don't represent geometry, but they are excellent, compact representations for fine detail, so they would be important components of any complete multiresolution modeling system. Much work remains to be done to derive textures from geometric models, however.

Pictures from Multiple Angles. This approach is intriguing, but can the problems of specular objects and occlusion be solved? Perhaps it should be used primarily for diffuse, nearly-convex objects.

Ray Space. Also intriguing, but the memory requirements may be extreme.

Polygonal Models. Polygonal models will probably form the core of a successful multiresolution modeling system, since they are the simplest, most versatile representation for geometry.

The polygonal simplification methods we discussed were developed with different goals in mind. Many of the simplification algorithms are limited to meshes, they are slow, and they consider shape only when doing their simplification, not material attributes such as color, specularity, or texture. Further work is needed to adapt them to the goals of multiresolution modeling.

Rossignac and Borrel's simplification algorithm is the most general, since it accepts polyhedral models with arbitrary topology as input. It can achieve greater simplification since it is free to change the topology of models. On the negative side, this algorithm shows artifacts of the clustering grid and it does not preserve detail as well as might be possible. Nevertheless, this algorithm is a good starting point for future research. With additional work on preservation of appearance, a simplification algorithm well suited for fast rendering could be developed.

A full multiresolution modeling system would need to combine several of these data structures in order to represent objects using a combination of geometry, texture, and reflectance, and it would need to smooth the transitions between representations during rendering.

Acknowledgments

Thanks to Tom Funkhouser, Jon Webb, Andy Witkin, Dave McKeown, and Jarek Rossignac for valuable discussions. This work was supported by ARPA contract F19628-93-C-0171.

References

[1] Barry G. Becker and Nelson L. Max. Smooth transitions between bump rendering algorithms. In *SIGGRAPH '93 Proceedings*, pages 183–189. ACM, 1993.

[2] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):273–281, July 1987.

[3] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH '93 Proceedings*, pages 279–288. ACM, 1993.

[4] James H. Clark. Hierarchical geometric models for visible surface algorithms. *CACM*, 19(10):547–554, Oct. 1976.

[5] Michael DeHaemer, Jr. and Michael J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers and Graphics*, 15(2):175–184, 1991.

[6] James George Dunham. Optimum uniform piecewise linear approximation of planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):67–75, January 1986.

[7] Alain Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, May 1992.

[8] Thomas A. Funkhouser. *Database and Display Algorithms for Interactive Visualization of Architectural Models*. PhD thesis, CS Division, UC Berkeley, 1993.

[9] Thomas A. Funkhouser, Carlo H. Sequin, and Seth J. Teller. Management of large amounts of data in interactive building walkthroughs. In *1992 Symposium on Interactive 3D Graphics*, pages 11–20, 1992. Special issue of Computer Graphics.

[10] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical z-buffer visibility. In *SIGGRAPH '93 Proceedings*, pages 231–238. ACM, 1993.

[11] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):197–206, July 1991.

[12] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, Nov. 1986.

[13] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH '93 Proceedings*, pages 19–26, Aug. 1993.

[14] Insung Ihm and Bruce Naylor. Piecewise linear approximations of digitized space curves with applications. In N. M. Patrikalakis, editor, *Scientific Visualization of Physical Phenomena*, pages 545–569, Tokyo, 1991. Springer-Verlag.

[15] James T. Kajiya. Anisotropic reflection models. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):15–21, July 1985.

[16] Ken Perlin. A unified texture/reflectance model. In *SIGGRAPH '84 Advanced Image Synthesis seminar notes*, July 1984.

[17] Michael F. Polis and David M. McKeown, Jr. Issues in iterative TIN generation to support large scale simulations. In *Proc. of 11th Intl. Symp. on Computer Assisted Cartography (AUTOCARTO 11)*, pages 267–277, November 1993.

[18] Azriel Rosenfeld, editor. *Multiresolution Image Processing and Analysis*. Springer, Berlin, 1984. Leesberg, VA, July 1982.

[19] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. Technical report, Yorktown Heights, NY 10598, February 1992. IBM Research Report RC 17697 (#77951). Also appeared in the *IFIP TC 5.WG 5.10 II Conference on Geometric Modeling in Computer Graphics*, Genova, Italy, 1993.

[20] Holly E. Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. In *Proc. Graphics Interface '93*, pages 227–236, Toronto, Ontario, May 1993. Canadian Inf. Proc. Soc.

[21] Georgios Sakas and Matthias Gerth. Sampling and anti-aliasing of discrete 3-D volume density textures. In *Eurographics '91*, pages 87–102, 527, Amsterdam, 1991. North-Holland.

[22] Lori Scarlatos and Theo Pavlidis. Hierarchical triangulation using cartographic coherence. *CVGIP: Graphical Models and Image Processing*, 54(2):147–161, March 1992.

[23] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):65–70, July 1992.

[24] Seth J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, CS Division, UC Berkeley, October 1992. Tech. Report UCB/CSD-92-708.

[25] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):55–64, July 1992.

[26] Steve Upstill. *The Renderman Companion*. Addison Wesley, Reading, MA, 1990.

[27] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting reflectance functions from complex surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(4):255–264, July 1992.

[28] Lance Williams. Pyramidal parametrics. *Computer Graphics (SIGGRAPH '83 Proceedings)*, 17(3):1–11, July 1983.

[29] Michael J. Zyda. Course notes, book 10. Technical report, Graphics & Video Laboratory, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, November 1991.