Dynamic Algorithm Binding for Interactive Walkthroughs

PEDRO PIRES[1,2]
JOÃO PEREIRA[1,2]
[1]INESC-ID – Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9, 1000-029 Lisboa, Portugal
[2]IST/UTL – Instituto Superior Técnico, Universidade Técnica de Lisboa
{pedro.pires,joao.pereira}@gaiva.inesc.pt

Abstract. This paper presents a novel approach to the real-time rendering of complex scenes problem. Up to the present date, a huge number of acceleration techniques have been proposed, although most are geared towards a specific kind of scene. Instead of using a single, or a fixed set of rendering acceleration techniques, we propose the use of several, and to select the best one based on the current viewpoint. Thus, dynamically adapting the rendering process to the contents of the scene, it is possible to take advantage of all these techniques when they are better suited, rendering scenes that would otherwise be too complex to display at interactive frame rates.

We describe a framework capable of achieving this purpose, consisting on a pre-processor and an interactive rendering engine. The framework is geared towards interactive applications were a complex and large scene has to be rendered at interactive frame rates. Finally, results taken from our test implementation are given.

1 Introduction

Real-time rendering of complex scenes is an old problem in computer graphics. Applications such as computer aided design (CAD), architectural visualization, flight simulators, virtual environments and computer games need to display scenes comprising many thousands, if not millions of polygons. Even today's hardware accelerated rendering pipelines cannot handle all the polygons that compose such scenes. Due to our everlasting quest for visual realism, the gap between processor power and scene complexity, is expected to maintain itself in the future. Thus, it is necessary to employ rendering acceleration techniques, i.e., techniques that reduce the number of polygons sent to the graphics pipeline, at each frame.

Over the years, several techniques that address this problem where proposed. They can be categorized into the following areas: (1) visibility determination techniques, and (2) geometric complexity reduction techniques. Area (1) encompasses topics such as view-frustum culling techniques and occlusion culling techniques, where as area (2) includes geometric detail reduction techniques and image-based acceleration techniques.

Most of these techniques have one thing in common: they are special purpose, i.e., they are built to explore characteristics of certain types of scenes. As an example, consider a city model: if the camera were located inside a building, in a room, it would be wise to use techniques that take advantage of the room's specific geometric arrangement – walls, which occlude other areas, and doors through which one can see – namely: portal based techniques [Airey90,Teller91,Leubke95]. If the user were walking along a street, it would no longer be possible to use portal techniques, instead, more generic occlusion culling techniques should be used, such as shadow frusta [Coorg97,Hudson97] or hierarchical occlusion maps [Zhang97]. If the user where flying high above the city, it would be useless to use occlusion culling techniques, instead detail reduction techniques should be used, for most of the city would be visible.

As shown by this example, the best techniques to use depend on where the user is, and where he is looking at. In this paper we propose a new approach to the real-time rendering of complex scenes problem: we propose to select the set of rendering acceleration techniques based on the current viewpoint and view-direction and we present a framework capable of achieving this purpose.

This paper is organized as follows: in section 2, we review the state of the art in rendering acceleration techniques: specifically: on visibility culling, scene simplification and image-based acceleration techniques; in section 3 we present our general framework; on section 4, we provide an overview of our test implementation; on section 5 we show our results; and on section 6 we present our conclusions.

1530-1834/01 $10.00 © 2001 IEEE

2 Previous work

2.1 Visibility Determination

View-Frustum Culling

View-frustum culling techniques try to quickly determine which objects are inside the current view-frustum and pass only those through to the graphics pipeline. Fist, the scene is augmented with either bounding volume hierarchy or a spatial partitioning hierarchy; then each volume is hierarchically tested for containment within the frustum planes [Moller96].

Occlusion Culling

Occlusion culling techniques try to eliminate objects (occludees) that are hidden behind other objects (occluders).

Greene et. al. proposed the use of a hierarchical z-buffer [Greene93]. A scene in augmented with a hierarchical spatial partitioning tree and it is rendered in front-to-back order. Polygons will be tested against pyramid of z-buffers, and for densely occluded scenes. The last, and thus the furthest, will be quickly rejected. Until recently this technique was limited to software rendering, but the latest mass consumption graphic accelerators already implement hierarchical z-buffers in hardware.

Zhang et. al. proposed another image space occlusion culling technique – Hierarchical Occlusion Maps (HOMs) [Zhang97]. For a given viewpoint, several occluders are rendered into an image, using hardware acceleration. Several lower resolutions of this image are created, also using the standard hardware, in order to form an image pyramid. The bounding box of the remaining objects are projected onto the screen and checked against the image pyramid. Any object whose bounding box is covered by the projection of the occluders is hidden behind them.

Object space occlusion techniques were also proposed [Coorg97,Hudson97]. The basic idea is that if the viewpoint is considered to be a light source, any object in the shadow of an occluder is invisible from the same viewpoint. A shadow frusta, i.e., a set of planes bounding this area can be built and each object's bounding box is tested against these planes.

Another set of techniques, instead of trying to determine what is invisible from a point in space, try to determine what is visible from a given volume in space – its Potentially Visible Set (PVS).

For static interior scenes, portals and cells are the preferred solution. The terms portals and cells where first coined by Jones [Jones71] in the context of hidden line removal, back in 1971. In his algorithm, models were manually subdivided into convex cells and convex portals. Rendering begins with the walls and portals of the cell containing the user. As each portal is drawn, the cell on the opposite side is recursively rendered and its contents clipped against its portal. Later, Airey [Airey90] focused on the problem of determining cell-to-cell visibility, proposing several solutions, including point sampling and shadow volumes. Teller [Teller91] took the concept further and found an analytic solution to the portal-to-portal visibility problem based on linear programming techniques. Luebke and Georges [Leubke95] suggested dynamically calculating the PVSs at run-time.

For static exterior scenes techniques were proposed that calculate the PVS for a volume in space. They are conservative volume techniques since they calculate a as tight as possible superset of the objects which can be seen from any point inside a given volume in space [Cohen-Or98,Durand00,Schaufler00].

2.2 Geometric Complexity Reduction

Geometry-Based Techniques

Geometry based techniques aim to reduce the number of polygons used to represent an object, based on certain criteria: typically, its distance to the viewpoint. The reason behind this is that objects further away from the viewpoint occupy a smaller screen space area, and thus can be represented with less detail.

Static levels of detail, first introduced by Clark [Clark74], use several representations of the same object at different resolutions and switch between them based on the distance to the user. Funkhouser and Séquin studied the problem of selecting the set of object representations that maximizes the visual benefit while maintaining a target frame rate [Funkhouser93].

Progressive meshes, introduced by Hoppe [Hoppe96] represent an object as a coarse base mesh, plus a record of each operations required to transform it into the full resolution mesh. These operations are invertible, and each adds a new vertex. Thus, it is possible to progressively increase or decrease detail, depending on the distance to the viewpoint. The addition and removal of vertices can be smoothly animated.

Techniques capable of selectively refining the models, where also proposed [Xia96 ,Hoppe97,Luebke97]. For large models, it is desirable to increase details in the parts that are closer to the viewpoint, and not the object as a whole.

Special purpose algorithms, capable of reducing the detail in a view dependent way were also proposed for height field (terrain) data [Lindstrom96,Duchaineau97,Hoppe98].

Another possibility is the use of parametric surfaces to represent scene content [Kumar97]. They can be tessellated on the fly, with more or less detail, depending on their distance to the user.

Image-Based Acceleration Techniques

Image-based acceleration techniques, try to replace portions of a scene with images. An image can represent as many objects as necessary, and its rendering time depends only on its resolution. On the other hand, it can only represent static content and it is only valid when seen from the point where it was created.

Marciel and Shirley were the first to use images in this context. They replaced objects with pre-rendered images (a single polygon texture mapped with an image), and coined the name impostors for this new primitive [Maciel95].

Shade et.al. [Shade96] and Schaufler and Stürzlinger [Schaufler96] introduced the idea of image caches. Impostors are generated on the fly to replace parts of the scene, and remain valid for a couple of frames. As the viewpoint moves, and the error exceeds a user specified number of pixels, the impostor(s) will be invalidated and regenerated.

Techniques to extend the life of impostors were proposed by Scilion et. al.[Sillion97] and Decoret et. al. [Decoret99].

Images have also been used in portal environments. Aliaga and Lastra introduced the concept of portal textures [Aliaga97]. They replaced portals with pre-rendered images of the cells behind it and showed how to smoothly transition from images to geometric based representations of a portal, when the user approaches it. They also introduced a method to automatically place images in order to guarantee a minimum frame rate [Aliaga99].

2.3 Discussion

Most of the above-mentioned techniques are special purpose ones, i.e., they are best suited for specific kinds of scenes. Some can only handle static scenes; others are suited for both static and dynamic ones. Some require that the scene be represented as a set of disjoint objects; others do not impose any particular structure. Some are only usable for interior scenes; others are geared towards exterior ones. Some offer advantages when used with geometry located close to the viewpoint, others are preferred for objects located far away. Given these criteria, Table 1 summarizes the requirements/ appropriateness of each above-mentioned technique, and groups them into categories.

TECHNIQUES				Dynamics	Structure	Organization	Distance
Visibility	Point Visibility	Image Space Techniques	Hierarchical z-buffer	DYN/STA	IN/OUT	UNSTR	
			HOMs	DYN/STA	IN/OUT	STR	
		Object Space Techniques	Shadow Frusta	DYN/STA	IN/OUT	STR	
	Conservative Volume Visibility	Interiors	BSP+PVS	STA	IN	UNSTR	
			Portals	STA	IN	STR	
		Exteriors	Octrees+PVS	STA	OUT	STR	
Detail	Geometry Based Techniques	Static LODs		DYN/STA	IN/OUT	STR	NEAR
		View-Dependent Progressive Meshes		DYN/STA	IN/OUT	STR	NEAR
		View-Independent Progressive Meshes		DYN/STA	IN/OUT	STR	NEAR
		Parametric Surfaces		DYN/STA	IN/OUT	STR	NEAR
		Terrain		DYN/STA	OUT	STR	
	Image Based Techniques	Impostors		STA	IN/OUT	UNSTR	FAR
		Image Caches		STA	IN/OUT	UNSTR	FAR
		Portal Textures		STA	IN	UNSTR	FAR
		Pure Image Based		STA	IN/OUT	UNSTR	FAR

Table 1 - Categorization of existing Rendering Acceleration Techniques and situations where they should be used. DYN stands for dynamic scenes, STA for static scenes, IN for interior scenes, OUT for exterior scenes, UNSTR for unstructured scenes (poly soup), STR for structured scenes (objects/meshes), NEAR for scene content close to the viewpoint and FAR for scene content far from the viewpoint.

Point visibility occlusion techniques are suitable for both interior/exterior and static/dynamic scene content. Among these, Shadow Frusta and Hierarchical Occlusion Maps (HOMs) handle static occluders better, but occludees can still be dynamic. They also require that the scene be organized as a set of objects that have bounding volumes assigned. Hierarchical Z-buffer imposes no organization on the scene.

Conservative volume visibility evaluation techniques are done off-line, thus are only applicable to static scene content. Portals and BSP trees are special purpose and suitable only to interior scenes. While Portals require the scene to be subdivided into cells and portals, BSP trees impose no such structure, since they are able to create it from polygon soup models. Octree-based conservative visibility evaluation techniques are suited for exterior scene content.

The distinction between complexity reduction techniques is cleaner. With the exception of portal textures, which are more special purpose, they are all suited to both interior and exterior scenes. Geometric detail techniques require the scene to be organized as a set of objects some with special purpose representations, while image-based techniques do not impose any special structure on object representation. Geometric detail techniques are suited for both static and dynamic objects,

while image-based techniques still have difficulty to cope with dynamic scenes, be it moving objects or changing illumination conditions. Images are also better suited to represent scene content that is far from the viewer, given that the error induced as the viewpoint moves away from the viewpoint from where it was originally created is smaller, if the image is placed far away from the user. Geometric detail techniques are usually better to represent scene content closer to the user, since that after some point in the distance an object will contribute very little to the user perception of the scene and its lowest detail can be used, without having to change it further.

Moreover, it is useless to apply occlusion-culling techniques if a substantial part of the scene is not occluded from a given viewpoint. Similarly, it is futile to employ complexity reduction techniques, if there are not enough polygons visible.

As a conclusion, it is safe to say that: *the best acceleration techniques are dependent on the chosen viewpoint and view-direction.*

3 Selecting the best rendering acceleration algorithm

In order to select the best rendering acceleration techniques, it is necessary to analyze the scene and determine which of the available techniques are best suited for the current viewpoint. This process takes its time, which should be minimized. Specifically, the time to select an algorithm α_i, amortized by the number of frames it remains valid, plus the time to render the scene using it, should be inferior to the time to render the scene with a single algorithm β.

$$\frac{t_{select}(\alpha_i)}{valid\_frames} + t_{render}(\alpha_i) < t_{render}(\beta)$$

Another important criterion is that it is undesirable to switch between algorithms every frame. In other words, spatial coherency hints that the best techniques for a given viewpoint remain the same over a given neighborhood thereof.

To keep the selection time low, we propose to analyze the scene off-line, and store information that can be used at runtime to select the appropriate algorithm, once the viewpoint is known.

To explore spatial coherency, we suggest augmenting the scene with a spatial subdivision hierarchy, and marking each node with the techniques to use, when the viewpoint is located within the volume it represents. This tree should be kept separate from the usual view-frustum culling hierarchy, since it fits a different purpose: the former groups objects based on their spatial location, and the later categorizes zones of space. Even if the same type of data structure is used, each will benefit from different subdivision criteria. Possible choices for such a tree are an octree, a k-dtree, a bsp-tree or a similar data structure.

Since the best techniques are liable to change with the view-direction, it is necessary to consider several view-directions per node. Ignoring the camera tilt, its orientation can be expressed by two angles: θ (azimuth) and φ (elevation). Dividing the space of all possible orientations into n discrete intervals of the form $[\theta_i, \theta_{i+1}[\times [\varphi_i, \varphi_{i+1}[$, it is possible to determine the best techniques when the view-direction is inside each interval. This information can be stored at each node using a bi-dimensional table, indexed using the current view direction (θ, φ) (see Figure 1)

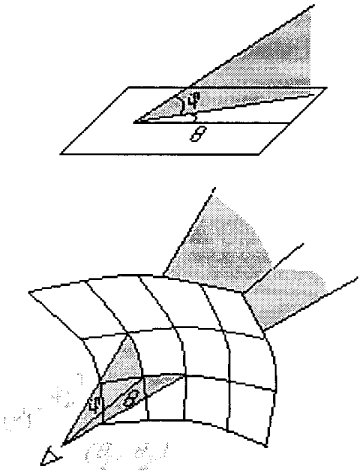

Figure 1 - Dividing the space of all possible view-directions into discrete intervals

Such representation takes advantage of space coherency since that the contents of a scene, as seen from two points close by in space, using more or less the same view-direction, are likely to be very similar. Thus the best techniques are also likely to be the same in both situations.

Summarizing: during the preprocessing step the scene is divided into a set of volumes, using a spatial subdivision hierarchy. Each volume is analyzed in order to determine the best techniques to use when the viewpoint is therein, and looking in a given direction. This information is written in the corresponding tree node. Additional preprocessing required for each particular algorithm is done and stored at this time. At run time, once the viewpoint is known, the spatial subdivision hierarchy is used to rapidly access the pre-computed information and to determine the techniques to use (refer to Figure 2).

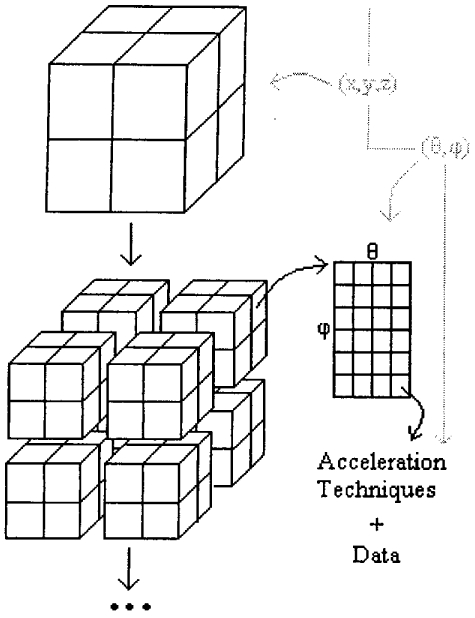

Figure 2 – The scene is augmented with a Hierarchical Space Partition Tree (an octree in this example) and each node stores a table with the techniques to use for each view-direction. At runtime the viewpoint (x,y,z) is used to determine the current tree node and the view-direction (θ,φ) is used to index the corresponding table, obtaining the acceleration techniques and associated data.

3.1 Scene Preprocessing

The preprocessor's goal is to build the hierarchical spatial partition tree, to analyze the scene as seen from each node and view-direction, to determine the techniques that are better suited in each case, to store this information at each node and to write the hole data structure to disk.

In order to determine if acceleration techniques are required, it is necessary to measure how complex is a scene, as seen from a given viewpoint. Then there are two characteristics that have to be analyzed for each viewpoint and view-direction: first, in order to justify the use of occlusion techniques, it is necessary to determine the set of objects that can be seen, and the occlusion relationships between these objects. Second, to determine if detail reduction techniques are required, it is necessary to determine how complex is this set of objects.

Measuring Geometric Complexity

In order to measure how complex a scene is, instead of rendering it and measuring the time it took, we approximate the cost of rendering an object, measured in time, using the following model (see Funkhouser and Séquin [Funkhouser93]):

$$cost = \max\{\alpha_v \times num\_vertexes, \alpha_p \times num\_pixels\}$$

Constants α_v and α_p can be measured and established for each target machine. Using this model, results are not 100% accurate, but it is not necessary to preprocess a scene in the same machine where the visualization will occur.

Selecting Visibility Techniques

Since interior areas are usually modeled separately, we assume that they are marked as such in the scene model. Thus, it is not necessary to determine which areas are interior and which are exterior automatically. Interior areas are represented as cells with portals which link to other cells, or to the tree structure (for example, an octree) used to represent the outside areas (windows or doors can lead outside or to other rooms). Portals leading into inside areas will be represented in the nodes of this tree structure, corresponding to the volume where they are located.

Thus, if portal based techniques are used in a particular implementation, its hierarchical structure will be a combination of a cell graph, and an octree. The only requirement is that given a point in space, it must be possible to determine in which cell it is or in which octree node it is located.

A cell will assume the same role as an octree node. It will also have a table, which is indexed using the current view direction. If justifiable, a cell can be further divided (for instance, complex factory models are known to exhibit large, and highly intricate interior rooms, with pipes and machinery occluding each other). For all purposes, a portal is treated as any other object, except that when it is rendered, the cell behind it will be displayed instead.

During the preprocessing step this hierarchy is built, using the usual techniques. If memory constraints allow it, the PVS for the interior cells can be computed and stored at this time [Airey90,Teller91]. Alternatively visibility can be evaluated at run-time using Leubke and Georges' technique [Leubke95].

The first thing to do at each viewpoint is to cull all the objects outside the view-frustum, and determine if the remaining ones can be rendered under the allotted frame time. If they can, it is not necessary to use any special techniques. If they cannot, it is necessary to determine if there are objects that can be used as occluders. Object space occlusion culling techniques require a reduced number of large close by occluders. On the other hand, Zhang's hierarchical occlusion maps can handle several

smaller occluders. More specific measures on how to select ocluders for each technique are given in [Coorg97,Hudson97] and [Zhang97] respectively. Using our strategy, the acceleration techniques are chosen based on the number and type of possible occluders, and not the other way around.

Whatever technique gets chosen, the set of occluders to use for this viewpoint and direction are stored at the corresponding node.

Selecting Complexity Reduction Techniques

Given a viewpoint, a view-direction and the set of visible objects, the preprocessor determines if they can be rendered in the allotted frame time, without complexity reduction techniques. If they cannot, it tries to use different techniques for different parts of the scene.

The relevant criterion here is proximity: Image-based techniques are better suited for representing far away portions of the scene, since they can group lots of small objects into a single image, and the further the image is, the further one can move away from the point where it was created without noticing errors. On the other hand, after some point in the distance, the object will be too small and it can be rendered using the lowest detail possible. Thus geometric detail techniques are better suited for objects that are closer to the viewpoint.

Furthermore, since most image-based techniques can induce visual artifacts, we have chosen to try geometric detail techniques first and image-based ones only if necessary.

The visible objects are placed in a list sorted by distance to the viewpoint. Two passes are made through this list, starting from the further object to the closest one. In the first, each object is selected to be rendered using a geometric detail reduction approach. In the second, each object is selected to be rendered using an image-based approach. Note that in the later case, all selected objects can be represented using the same image-based primitive, since images can effectively combine several objects. After each selection, the time to render the scene is evaluated, and this process stops as soon as this value is under the required maximum frame time.

Note that a portal is treated as any other object. It can be displayed by rendering the geometry of the cell behind it, or using a portal texture [Aliaga97,Aliaga99].

The required images will be stored at the corresponding tree node table, and cannot be reused at other nodes since they are view-dependent.

Data required for the geometric detail reduction techniques, will be stored in a common pool and a pointer to the relevant information will be kept at the node, since this data can usually be shared by other nodes (for instance, if it is decided to use a progressive mesh to render an object, the same mesh will probably be used for many other positions and orientations.

3.2 Interactive Rendering

Once the preprocessing is done, rendering the scene is a simple matter. The user's position in space is used to index the spatial structure and the node corresponding to the volume where the user is. The view-direction is used to index the scene and determine the techniques to use.

Once again, if a portal is visible it will be treated as any other object, and will be rendered with the technique assigned during the preprocessing step.

4 Test implementation

In order to validate our ideas, we have developed a prototype preprocessor and a rendering engine. Since it would be impossible to implement all the techniques proposed in the literature in any reasonable amount of time, we decided to focus on a subset of thereof. In addition we chose the techniques that would be faster to implement.

We chose to use a quadtree to index exterior scene content and portals to represent interiors. For simplicity we do not calculate the PVS for each cell, instead we use Leubke and Georges' technique [Leubke95]. The techniques to use are kept in a separate structure, a 3D grid, since an octree would use too much memory.

We use hierarchical view-frustum culling to eliminate objects exterior to the view volume and shadow frusta to handle occlusions. As a geometric complexity reduction technique, we used a static level of detail approach.

At this time, we have not experimented with image-based acceleration techniques. We plan to use simple impostors for the further objects when necessary. As we are not considering warping techniques, [McMillan95] there will surely be artifacts when switching between impostors, but that is a characteristic of the chosen acceleration technique, and it does not invalidate the ideas presented at this paper, namely the possibility of preprocessing a scene in order to extract enough information to permit a fast real-time selection of the best algorithm for any viewpoint and view-direction.

5 Results

Our approach was tested using a model of our university's campus and surrounding area. It contains about 75k polygons spread by 3500 objects. In order to increase its

complexity, it was instanced 49 times (in a 7x7 square), adding up to 3,6M polygons.

To reduce the number of possible viewpoints, the user was only allowed to fly within the campus up to a height of 70m above the ground, at which time he can view the whole model.

The system was implemented entirely in C++ and OpenGL. The testes were run on a Pentium III 450MHz with a GeForce256 graphics accelerator and the frame rate was locked to the monitor refresh rate, 75Hz, yielding a maximum of 75fps. Without any acceleration techniques, the model could only be rendered at 0,6fps in this system. From within the campus, using only view frustum culling, the worst-case views were rendered at 2fps.

Depending on the user position and orientation, frame rate varied significantly. In interior areas 75fps where easily achieved. In exterior areas, close to ground level, the results depended on how well the preprocessor selected the occluders. In the presence of large objects close to the camera, the Shadow Frusta algorithm was activated and we achieved frame rates between 10fps and 30fps (depending on the number of occluders and how much of the scene they occluded). In open areas, with many small occluders close to the horizon and when flying above the campus, these were not the most appropriate algorithms, thus frame rate plummeted to 2fps. It is our belief that if we use Hierarchical Occlusion Maps and/or image based techniques, speedups will be possible in these areas, although we need to implement them, and perform further tests in order to make that claim.

6 Conclusions

We have presented a novel approach to the real-time rendering of complex scenes problem. We proposed to select the best rendering techniques for each viewpoint and view-direction. In order to minimize the time to select new algorithms, we analyze the scene offline and store information to permit a rapid selection at run-time.

We described a general framework capable of achieving this, composed of a preprocessor and an interactive rendering engine. We provided an overview of the state of the art in real time rendering acceleration techniques, and identified criteria that permit the selection of each given a specific type of scene. Given the encompassing nature of this work, we chose a small subset of these techniques, and implemented a prototype, which, although in its initial stages, validated our ideas: in situations for which appropriate algorithms were available, interactive frame rates where achieved. For other situations, further tests are needed.

7 Acknowledgements

Our main test model was provided by the Environment and Energy Group at Instituto Superior Técnico (SAE/IST) (http://visualis.ist.utl.pt). It is a small part of the Virtual Lisbon Project, which aims to build and display a 3D representation of the city of Lisbon. We would like to thank the team behind this project, for allowing us to use a part of their model.

References

[Airey90] John Airey. Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, TR#90-027, 1990.

[Aliaga97] Daniel G. Aliaga and Anselmo A. Lastra. Architectural Walkthroughs Using Portal Textures, IEEE Visualization '97, pp. 355-362 (November 1997). IEEE. Edited by Roni Yagel and Hans Hagen. ISBN 0-58113-011-2.

[Aliaga99] Daniel G. Aliaga and Anselmo Lastra. Automatic Image Placement to Provide a Guaranteed Frame Rate, Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference Series, pp. 307-316 (August 1999, Los Angeles, California). Addison Wesley Longman. Edited by Alyn Rockwood.

[Cohen-Or98] Daniel Cohen-Or, Gadi Fibich, and Dan Halperinand Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. Computer Graphics Forum, 17(3):243–254, 1998. ISSN 1067-7055.

[Coorg97] S. Coorg and S. Teller. Real Time Occlusion Culling for Models with Large Occluders. In Simposium on Interactive 3D Graphics, pages 83–90, April 1997.

[Decoret99] Xavier Decoret, François Sillion, Gernot Schau er, and Julie Dorsey. Multi-layered impostors for accelerated rendering. Computer Graphics Forum, 18(3):61–73, September 1999. ISSN 1067-7055.

[Durand00] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. Proceedings of SIGGRAPH 2000, pages 239– 248, July 2000.

[Duchaineau97] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. IEEE Visualization '97, pages 81–88, November 1997.

[Funkhouser93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Proceedings of SIGGRAPH 93*, pages 247–254, August 1993. ISBN 0-201-58889-7.

[Greene93] N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer Visibility. In Computer Graphics (SIGGRAPH 1993 Proceedings), pages 231–238, August 1993.

[Hoppe96] Hugues Hoppe. Progressive meshes. Proceedings of SIGGRAPH 96, pages 99–108, August 1996. ISBN 0-201-94800-1.

[Hoppe97] Hugues Hoppe. View-dependent refinement of progressive meshes. Proceedings of SIG-GRAPH 97, pages 189–198, August 1997. ISBN 0-89791-896-7.

[Hoppe98] Hugues H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. IEEE Visualization '98, pages 35–42, October 1998. ISBN 0-8186-9176-X.

[Hudson97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated Occlusion Culling Using Shadow Frusta. In Simposium on Interactive 3D Graphics, pages 1–10, June 1997.

[Jones71] C.B. Jones. A New Approach to the 'Hidden Line' problem. The Computer Journal, 14(3):232, August 1971.

[Kumar97] Subodh Kumar, Dinesh Manocha, Hansong Zhang, and III Kenneth E. Hoff. Accelerated walkthrough of large spline models. 1997 Symposium on Interactive 3D Graphics, pages 91–102, April 1997. ISBN 0-89791-884-3.

[Leubke95] D. Luebke and C. Georges. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In Symposium on Interactive 3D Graphics, pages 105–106 and 212, April 1995.

[Luebke97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. Proceedings of SIGGRAPH 97, pages 199–208, August 1997.

[Lindstrom96] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hughes, Nick Faust, and Gre-gory Turner. Real-time, continuous level of detail rendering of height fields. Proceedings of SIGGRAPH 96, pages 109–118, August 1996.

[McMillan95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. Proceedings of SIGGRAPH 95, pages 39–46, August 1995. ISBN 0-201-84776-0.

[Moller96] Tomas Moller and Eric Haines. Real Time Rendering. A K Peters, 1996.

[Maciel95] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. 1995 Symposium on Interactive 3D Graphics, pages 95–102, April 1995.

[Shade96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. Proceedings of SIGGRAPH 96, pages 75–82, August 1996. ISBN 0-201-94800-1.

[Schaufler96] Gernot Schaufler and Wolfgang Stürzlinger. A three dimensional image cache for virtual reality. Computer Graphics Forum, 15(3):227–236, August 1996. ISSN 1067-7055.

[Schaufler00] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and Fran¸cois X. Sillion. Conservative volumetric visibility with occluder fusion. Proceedings of SIGGRAPH 2000, pages 229–238, July 2000. ISBN 1-58113-208-5.

[Sillion97] François X. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. Computer Graphics Forum, 16(3):207–218, August 1997. ISSN 1067-7055.

[Teller91] S. Teller and C. Séquin. Visibility preprocessing for interactive walkthroughs. In Computer Graphics (SIGGRAPH 1991 Proceedings), pages 61–70, August 1991.

[Xia96] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. IEEE Visualization '96, pages 327–334, October 1996. ISBN 0-89791-864-9.

[Zhang97] H. Zhang, D. Manosha, T. Hudson, and K. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. In Computer Graphics (SIGGRAPH 1997 Proceedings), pages 77–88, August 1997.