Appeared: IEEE Visualization '96, pp. 101-106, Oct 27-Nov 1, 1996.

Visualization of Complex Models Using Dynamic Texture-based Simplification

Daniel G. Aliaga*
Computer Science Department
University of North Carolina at Chapel Hill

ABSTRACT

We are investigating methods for simplifying complex models for interactive visualizations using texture-based representations. This paper presents a simplification method which dynamically "caches" distant geometry into textures and trades off accurate rendering of the distant geometry for performance. Smooth transitions and continuous borders are defined between the geometry and textures thus the representations can be switched without sudden jumps (as is the case with many current texturing techniques). All the computations for the transitions can be done a priori without the need to change the textures each frame thereafter.

Keywords: geometry, textures, morphing, visual complexity, space partitioning, simplification, visibility culling, interactive.

1. INTRODUCTION

Geometric models have become very large and difficult to render at interactive rates. As a result, many algorithms have been developed to simplify models until they are (hopefully) small enough to be rendered at interactive rates. Two popular approaches are visibility culling and level-of-detail management. Visibility culling algorithms determine which subset of the model is visible and only render that portion of the geometry [1][7][12]. Unfortunately, these algorithms do not work well when many primitives are still visible. For example, complex rooms, used in architectural walkthroughs, have a large amount of visual complexity that must be rendered. Level-of-detail (LOD) algorithms [5][6][10], which typically require some manual interaction for generating the multiple LODs, cannot easily simplify scenes with a large number of visible objects.

A relatively new simplification approach is to dynamically represent geometric complexity using textures. Textures have the advantage of taking constant time to render regardless of the complexity of the portion of the model they represent. We are investigating how to create a system that renders the nearby subset of a model as geometry and the distant, but visible, subset of a model with a texture-based representation. As the viewpoint changes, the system dynamically changes the geometry into textures or the textures
back into geometry. Preliminary results indicate that representing the distant geometry with textures produces an adequate image quality for architectural walkthrough visualizations. The error introduced is proportional to the viewpoint's distance from the original texture sample point. The system can bound the error by resampling the texture as needed.

There are two major problems in developing this rendering system. First, since a texture represents an arbitrary subset of the model from a single viewpoint, changing the viewpoint causes the image displayed by the texture to be incorrect (unless image warping is used [9]). Consequently, the geometry surrounding the texture does not match the geometry sampled in the texture causing a discontinuity or "crack" to appear. Previous methods [8][11] have either ignored this or used an error metric to decide whether to resample the texture or display another texture from a set of precomputed textures. Unfortunately, storing many texture samples requires a vast amount of texture memory or fast texture paging while frequently resampling the texture can significantly reduce the performance gain of using textures. Furthermore, the algorithms in [8][11] have only been applied to outdoor scenes and are not well suited for indoor scenes (architectural walkthroughs, radiosity-illuminated rooms, etc.). We present a solution to the discontinuity problem (without warping the texture), thus providing a continuous border between geometry and texture. This gives us the freedom to unnoticeably place textures anywhere in a model.

The second problem occurs when switching between geometry and texture. Once the viewpoint has changed from the texture sample point, a transition from geometry to texture (or vice versa) will cause a sudden jump in the image (as in previous texture-based simplification methods). Therefore, we need to define transitions to smoothly change the geometry into texture and texture back into geometry. We present smooth transition operations to convert geometry into textures (and vice versa).

The following section presents an overview of the visualization algorithm (preprocessing and run-time phases). Section 3 describes, how the transitions from geometry to texture (and vice versa) are performed while maintaining a continuous border. Section 4 presents how multiple textures can be created. Section 5 shows some results we have obtained by using our algorithm with three complex models. Finally, Section 6 will end with some conclusions and future work.

2. TEXTURE-BASED SIMPLIFICATION

We have devised a simplification method which dynamically replaces arbitrary portions of a model with textures and morphs the nearby geometry to match the texture. This method is capable of simplifying models of high visual complexity in cases where visibility culling and object-based LOD are insufficient. The algorithm can be combined with an automatic scheme to decide what portion of the model to simplify to texture.

Our simplification method has a preprocessing phase and a run-time phase. The preprocessing phase statically partitions the

* aliaga@cs.unc.edu, (919) 962-1722
CS Dept., CB #3175, UNC-CH, Chapel Hill, NC 27599-3175

model into a 3D grid of boxes. Space partitioning and view frustum culling are used so that the amount of work to be done for rendering and for transitions to texture or back to geometry is proportional to the number of primitives in the view frustum. The run-time phase performs smooth transitions to texture and back to geometry while maintaining a continuous border between the geometry and texture. The following two subsections describe the major elements of the preprocessing and run-time phase.

Preprocessing Phase

The input to our visualization algorithm is a 3D model. For interactive visualizations (or walkthroughs), typically the model does not fit completely within the field of view. Thus during rendering, only the subset of the model inside the view frustum needs to be sent down the graphics pipeline [4].

In order to efficiently perform view frustum culling, the model needs to be space partitioned. Various space partitioning algorithms have been proposed (BSP trees, octtree subdivision, etc.). In our current implementation, we found it sufficient to use a uniform grid space partitioning algorithm. The bounding box of the entire model is subdivided into a 3D grid of boxes. Each box contains a list of all the primitives that lie within it. Primitives that intersect the boundary between two or more boxes are split (this slightly increases the number of primitives but makes the implementation simpler).

Run-time Phase

The user is allowed to move through the model. In order to increase interactive performance, the user can replace a distant (and visible) subset of the model with a texture. The texture is used to represent the subset of the model from the local view area. Unfortunately, once the viewpoint moves again, geometry surrounding the texture will appear discontinuous with the texture (Color Figure 1). In order to maintain a continuous border between the texture and the geometry around it, either:

- The texture must be warped to match the geometry.
- The geometry must be warped to match the texture.

The former case corresponds to image warping [2][3][9] in which the sampled texture has depth information and is reprojected every frame by warping the texture to the viewpoint of the current frame. The adjacent geometry is rendered normally.

We use the second approach, namely morphing the vertices of the geometry to match the texture and maintain C0 continuity (higher orders of continuity are also possible). This approach is more attractive because: (a) it allows texturing hardware to be efficiently used, (b) the texture does not need to be warped every frame, (c) all of the work is performed at one time (at geometry-to-texture transition time or as a precomputation), (d) it does not introduce visible artifacts as the viewpoint changes as may be the case with image warping. The visible artifacts introduced by image warping include cracks in the image due to incorrect splatting of the pixels and "empty areas" produced by previously occluded regions becoming visible with no rendering information available for the newly visible pixels. Although this method could be considered less "realistic" than warping the texture, it takes advantage of the fact that geometry is re-rendered every frame, so by slightly modifying the geometry you are able to use static textures and achieve higher frame rates.

After replacing a subset of the model with a texture (Color Figure 2), the user cannot walk forward beyond the texture plane (without returning the subset to geometry). Furthermore, geometry near the viewpoint is rendered normally while the geometry surrounding the texture maintains a continuous border with the texture but is not rendered completely accurately. This is not a bad tradeoff for the improved performance. In order to return the texture to geometry (for example, if the viewpoint gets too close to the texture), a smooth transition operation from texture back to geometry is performed over the next few frames.

3. SMOOTH TRANSITIONS

The algorithm presented in this paper performs smooth transitions between geometry and texture by morphing the nearby geometry. Initially, the entire model is represented as geometry. Then, an arbitrary subset of the model is simplified to a texture by a *geometry-to-texture* transition. Further rendering of the subset of the model requires only displaying the texture and not the geometry represented by the image of the texture. In order to return the texture to geometry, a *texture-to-geometry* transition must occur. Any number of subsets of the model can be replaced by textures (the textures can be created from a common viewpoint or from different viewpoints; details are in Section 4). The basic steps for these two transitions are given below.

The computations involved in the transition operations are relatively simple. In fact, they are proportional to the number of primitives rendered (more precisely, to the number of primitives in the view frustum prior to replacement of geometry with texture). Displaying the textures themselves is proportional to the number of pixels in the textures and independent of scene complexity. If the location of the textures and their sampling viewpoints are determined beforehand, all the computations for the projection and morphing operations can be done a priori at the expense of additional storage.

Geometry-To-Texture Transition

1. The user selects a subset of the model to replace with a texture. This can be done in various ways. We adopted the following strategy: select all geometry inside the view frustum and beyond a distance **d** from the viewpoint. Since we are using view frustum culling for rendering, determining what geometry is in the view frustum is trivial. The space partitioning allows us to easily determine which boxes (and thus what geometry) are behind the texture plane. The texture plane is defined as the plane whose normal is the current view direction $\mathbf{v_d}$ and contains the point $\mathbf{t_o}$ which is at a distance **d** from the eyepoint along the view direction (Figure 1). The subset of the model to be replaced with a texture is called the *culled geometry*.

2. The current rendered image of the culled geometry is copied from the framebuffer to texture memory. A texture primitive (a texture-mapped quadrilateral covering the subset of the model being replaced) is added to the model. Since the texture contains an image of shaded geometry, lighting is temporarily turned off when rendering the texture primitive (in our test cases, we used static lighting: directional lights or precomputed radiosity-illuminated scenes).

To reduce texture memory requirements, the texture can be sampled at a resolution lower than the framebuffer's resolution. The texturing hardware is used to magnify the texture using bilinear interpolation between the texels. On the other hand, the texture can be sampled at a higher resolution than it will be

displayed and prefiltered to achieve apparent high-quality antialiased imagery (in addition to MIP mapping the texture).

3. The culled geometry is removed from the set of rendered geometry. The space partitioning boxes that intersect the view frustum can be further partitioned or not culled at all. In our implementation, we choose not to cull these boxes. Thus, some geometry is rendered "behind" the edges of the texture and is never actually visible; in practice this amounts to only a small amount of geometry.

4. The geometry in front of the texture plane is rendered normally (*near geometry*). The geometry behind the texture plane (that has not been culled) and the geometry surrounding the texture are morphed to match the geometry represented by the texture (this is the continuous border problem). Both these sets become the *morphed geometry* (Figure 1).

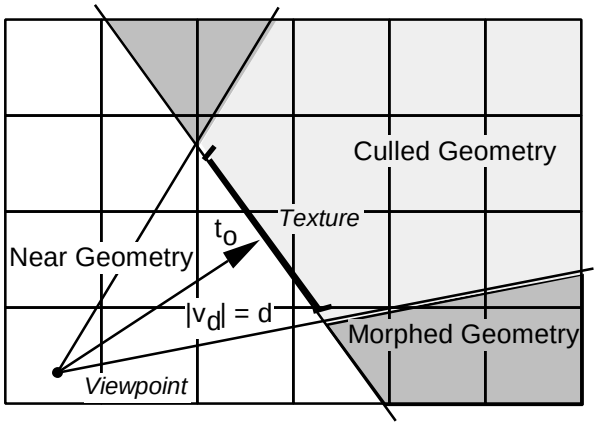


Figure 1: Model Partitioning Strategy. Each box corresponds to a space-partitioning box. The boxes are classified: near, culled and morphed. Intersected boxes can be optionally partitioned.

We defined the texture plane to always be parallel to the screen plane at sampling time (though it does not necessarily cover the entire view frustum). Therefore, during the first geometry-to-texture transition of a texture there is no need to gradually morph the geometry surrounding a texture from its original position to its projected position on the texture plane. Instead the surrounding geometry is immediately projected onto the texture plane.

Once a texture has been computed it might undergo various geometry-to-texture and texture-to-geometry transitions. All subsequent transitions (after the first geometry-to-texture transition) will generally be from viewpoints other than the texture sample point. Thus the surrounding geometry is gradually morphed from its original position to its projected position on the texture plane. In any case, since space partitioning and view frustum culling are used, only the visible geometry is actually morphed or projected. The morphing operation can be described by the following equation:

$$\mathbf{v_m(s) = sv_p + (1\text{-}s)v_o}$$

This describes a linear interpolation between the model-space vertex positions ($\mathbf{v_o}$) and the vertex positions projected onto the texture plane ($\mathbf{v_p}$), as **s** varies from 0 to 1. The latter set of vertices are almost the same as the screen-space projection of the vertices at the time of the first geometry-to-texture transition of a given texture. In fact, they are obtained by transforming the model-space vertices to screen-space ($\mathbf{v_s}$), then setting their z-value to be the screen-space projected z-value of the texture plane ($\mathbf{t_{sz}}$) and transforming them back to model-space:

$$\mathbf{v_s = Tv_o \;\; t_s = Tt_o \;\; \textregistered \;\; v_{sz} = t_{sz} \;\; \textregistered \;\; v_p = T^{-1}v_s}$$

In order to morph the vertices from their original position to their projected position, the value of **s** is gradually incremented from 0 to 1 over the next few frames after the start of the transition (Figure 2 and Color Figure 3).

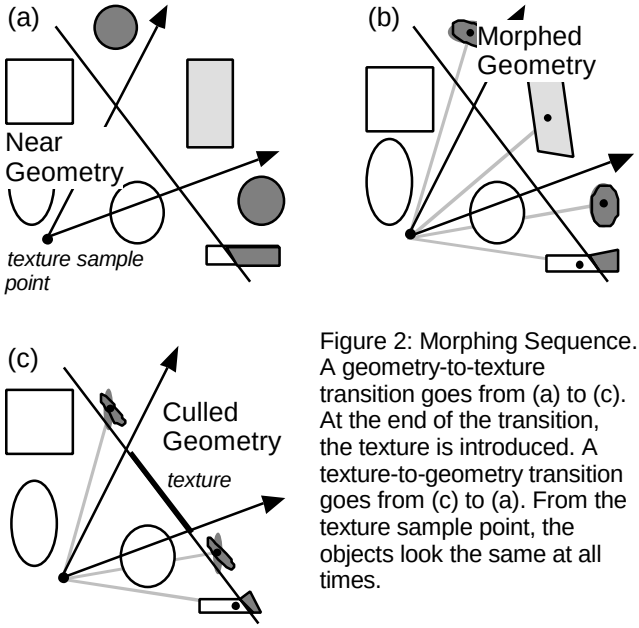


Figure 2: Morphing Sequence. A geometry-to-texture transition goes from (a) to (c). At the end of the transition, the texture is introduced. A texture-to-geometry transition goes from (c) to (a). From the texture sample point, the objects look the same at all times.

The above morphing operation maintains C0 continuity (i.e. positional continuity) between the texture and the geometry surrounding the texture (morphed geometry). This implicitly achieves C0 continuity of the texture and morphed geometry's border with the near geometry. It does not require morphing the near geometry. Higher order interpolation could be used to achieve smoother continuity between the texture, near geometry and morphed geometry. In this case, the near geometry, close to the texture, would be modified to obtain smoother continuity with the texture and morphed geometry. This would improve the texture and morphed geometry's border with the near geometry when viewed far from the original texture sample point. Unfortunately, this would violate the desire to keep near geometry undistorted.

Texture-to-Geometry Transition

1. The culled geometry is reintroduced into the model. The vertices are set to their projected position on the texture plane. This is done by computing the vertex positions with **s** = 1.

2. The vertices of the geometry reintroduced into the model are morphed from their projected position on the texture plane to their original position (note that if the texture plane is currently not in the view frustum, an instantaneous transition can be performed). The value of **s** is gradually reduced from 1 to 0 over the next few frames (Color Figure 3).

3. The vertices of the morphed geometry are similarly returned to their original position. Again, since space partitioning and view frustum culling are used, only the visible geometry is actually morphed.

4. MULTIPLE TEXTURES

So far we have described how to replace a single subset of a model with a texture. Multiple subsets of a model can also be simultaneously represented by using multiple textures.

Additional textures can be created either from a common viewpoint and constant distance or from different viewpoints and distances. The main difference between the two strategies is the effect they have on how geometry is morphed. In our system, we implemented both strategies except for the cases that require morphing of geometry to match two textures simultaneously.

• Common Viewpoint and Constant Distance: The multiple textures are created by changing only the view direction and selecting non-overlapping subsets of the model. The same distance from viewpoint to texture (**d**) is used. There is no need to morph the geometry surrounding each individual texture until all the texture samples have been obtained. Therefore, each texture will have an image of unmorphed geometry. If adjacent textures share a common edge (i.e. no geometry is rendered between the textures), the textures combine to form a single large texture with piecewise planar components. This last variation can be used to create textures that cover a complex region of the model or even completely surround the viewpoint. The geometry surrounding each texture is morphed in the same way as described in the previous subsections (Figure 3a).

If two textures (a "left" texture and a "right" texture) are created using a common viewpoint but different **d** values and not sharing an edge, the geometry in between the textures will have to be morphed to match both textures simultaneously. While this is possible, the distortion introduced by the two textures might be very apparent from certain view directions. This is especially true if very different **d** values are used. For example, assume the left texture was sampled at a significantly closer distance than the right texture. Thus, viewing the left texture from the left side might occlude some of the right texture and all of the geometry in between both textures.

• Different Viewpoints and Distances: If multiple textures are created from different viewpoints each at a potentially different distance from its viewpoint, the geometry surrounding each texture must be morphed before continuing on to create the next texture (Figure 3b). Consequently, the first texture will have an image of unmorphed geometry. Subsequent textures that are created by using geometry surrounding the previous textures, will contain images of morphed geometry. Textures that are created from viewpoints and view directions that do not contain the geometry in the plane of the previous textures are sampled using unmorphed geometry. Thus, it might be the case that the distortion introduced by the morphing operations will be magnified after several instances of textures from different viewpoints. Typically, this will not be the case since the number of textures needed to surround the local view area is small. If the view area migrates to another portion of the model (by a series of transitions), a new set of textures is used.

The subset of the model morphed for a particular texture can intersect with another morphed geometry subset. This is similar to the case of two textures using a common viewpoint but different **d** values. Both morphed geometry subsets will have to be morphed to match the textures simultaneously.

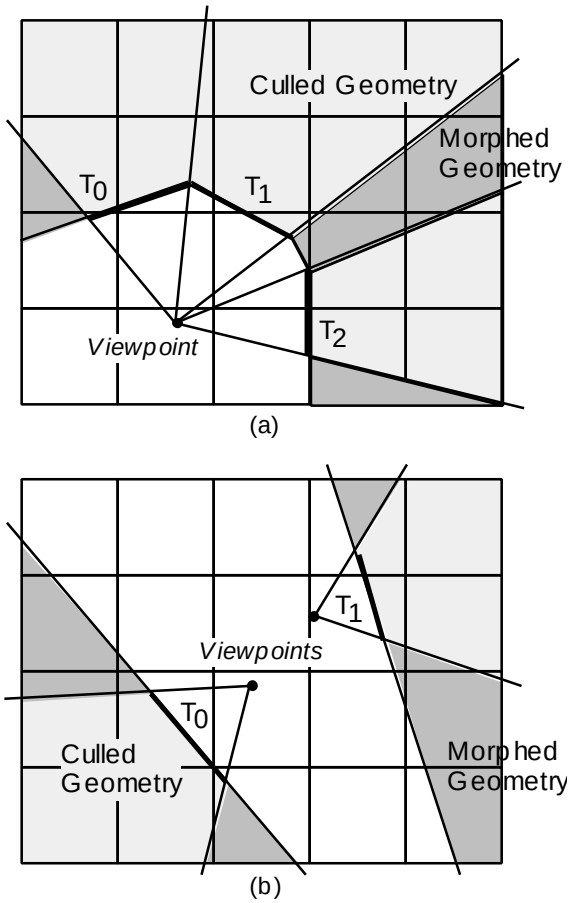


Figure 3: Multiple Textures. (a) Common viewpoint and constant distance. (b) Different viewpoints and distances.

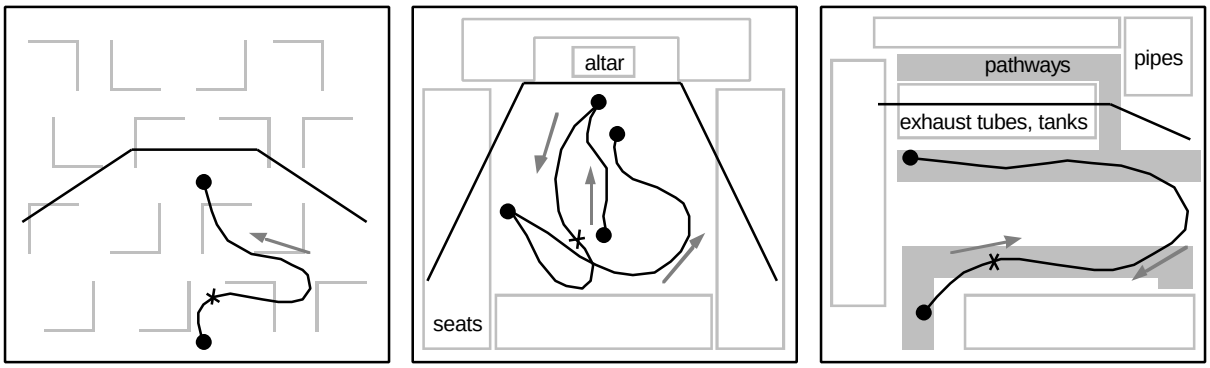
5. RESULTS

We implemented this algorithm on a SGI graphics workstation using OpenGL. The algorithm was applied to three models: a procedurally generated pipes model, a radiosity-illuminated church[1] and an auxiliary machine room of a nuclear submarine[2].

For each model, several textures were created. We recorded various paths through the models (spline-interpolated paths and some freehand manipulation of the viewpoint). Interactive performance is significantly improved when textures are introduced. The distortion caused by the morphing is not very noticeable as you can see in the color figures (and video). Furthermore, no discontinuities or "cracks" are perceivable at the border between geometry and texture, though some color disparities are present. Since we are using static lighting, the texture and geometry's colors should match at the border. The small disparity may be a consequence of the texture sampling or of exactly how the texturing hardware processes the texel colors.

The procedurally generated pipes model provides a very good test case of a high depth complexity model where visibility culling (for example, cells and portals [1][7][12]) is hard to apply. The model has 205,536 primitives (triangles). The space partitioning algorithm divided the model into a

[1] Courtesy of Lightscape Technologies Inc.
[2] Courtesy of Electric Boat Division, General Dynamics Corporation.



(a) (b) (c)

Figure 4: (a) Outline of pipes model and the path through the model (3 textures present). (b) Radiosity-illuminated church model and the recorded path (3 textures). The viewer is always looking in the general direction of the textures. (c) Auxiliary machine room model with its flythrough path (2 textures). The path is traversed in both directions but always looking towards the complex region of the model. In all figures, the 'x' marks the spot where the corresponding color figure was taken.

10x1x10 grid of boxes (most of the complexity lies in the plane of the observer, namely the XZ plane). If the viewpoint was near the edge of the model looking outward, view frustum culling alone was able to produce decent interactive performance (18.52 frames/second). But if the view direction rotated to look inside the model, or if the viewpoint was approximately in the middle of the model, performance without texture-based simplification is very poor (1.21 frames/second at the edge of the model looking inward, 1.98 frames/second in the middle of the model). Three textures were introduced covering the field of interest at a far distance. A single path was traversed from the border of the model towards the middle of the model. The average frame rate at the edge of the model looking inwards was 9.09 frames/second and in the middle of the model was 22.20 frames/second (Figure 4a and Color Figure 4a). We do not regard the "average frame rate" as the best means to measure the performance. The frame rate with textures present depends greatly on how much geometry is actually rendered. A decent view of the model can be produced with very little geometry and a few textures. We are still investigating better metrics of performance.

The radiosity-illuminated church model is an example of a single room that is visually complex (158,604 triangles, 8x3x8 grid of boxes). Geometric-based LOD algorithms which simplify enough to significantly improve rendering performance would lose much of the shape and color details of the room. A texture on the other hand reduces rendering complexity, but maintains the apparent detail. In our test case, we recorded a path rotating and translating about the middle of the room (Figure 4b and Color Figure 4b). The average frame rate without texture-based simplification was 2.19 frames/second. After we introduced three textures, the frame rate rose to an average of 7.25 frames/second. The video demonstrates the dynamic transitions of each texture with this model and the subsequent performance increases and decreases.

Finally, the auxiliary machine room is an example of a visually complex model with high depth complexity (273,531 triangles, 10x1x14 grid of boxes). The recorded path (Figure 4c and Color Figure 4c) takes approximately 113.90 seconds to render using only view frustum culling (1.17 frames/second average). We created two textures representing the geometry in the distance and it took only 17.29 seconds to traverse the same recorded path (7.69 frames/second average; we introduced a third texture and the frame rate increased to an average of 9.47 frames/second).

6. CONCLUSIONS & FUTURE WORK

Visibility culling algorithms alone cannot simplify models when a large subset of a model is still visible. For example, models consisting of many rooms are well suited for visibility culling algorithms, but if the geometry inside one room is still very complex there is inherently a large amount of geometry in the view frustum. Object simplification requires highly structured models in order to separate the model into objects. Furthermore, in scenes where there is high visual complexity, object simplification is not always sufficient.

The method presented here is able to simplify models and increase performance in the cases where visibility culling breaks down. Furthermore, it does not require highly structured models as with object simplification. Scenes can be rendered quickly regardless of visual complexity and we are able to achieve a rendering time proportional to the amount of nearby geometry and the number of textures used.

We are currently exploring algorithms to decide automatically when to perform the transitions. A cost-benefit style function [6] determines when and where in the scene the transitions should occur in order to maintain an interactive frame rate. This will enable us to visualize complex models while automatically "caching" distant geometry into texture-based representations.

Furthermore, we are developing methods to (quickly) measure the perceptual error introduced by the morphing operations. This will help us to decide when a new texture is needed (since the texture-caches are only valid for the local view area) and how to perform any necessary morphing operations, including higher order interpolation of geometry near and in front of the textures.

In addition, we are investigating ways to remove the restriction of using static lighting. For example, by including per-texel normals and other information it might be possible to recompute the shading for the geometry represented by the texture.

ACKNOWLEDGMENTS

This work would not have been possible without the support and advice of my advisor, Anselmo Lastra. I would also like to thank Gary Bishop, Frederick J. Brooks, Bill Mark, Michael North and Peggy Wetzel for her great job helping me with the video. This work was supported in part by NSF MIP-9306208 and ARPA Order No. A410.

REFERENCES

[1] Airey J., "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments", *Symposium on Interactive 3D Graphics*, 1990, pp. 41-50.

[2] Chen S. E., Williams L., "View Interpolation for Image Synthesis", *Computer Graphics (Proceedings of SIGGRAPH '93)*, 1993, pp. 279-288.

[3] Chen S. E, "QuickTime VR - An Image-Based Approach to Virtual Environment Navigation", *Computer Graphics (Proceedings of SIGGRAPH '95)*, 1995, pp. 29-38.

[4] Clark J., "Hierarchical Geometric Models for Visible Surface Algorithms", *CACM*, Vol. 19(10), October 1976, pp. 547-554.

[5] DeHaemer M., Zyda M., "Simplification of Objects Rendered by Polygonal Approximations", *Computer Graphics*, Vol. 15(2), 1991, pp. 175-184.

[6] Funkhouser T., Sequin C., "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments", *Computer Graphics Proceedings (Proceedings of SIGGRAPH '93)*, 1993, ACM SIGGRAPH, pp. 247-254.

[7] Luebke D., Georges C., "Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets", *Symposium on Interactive 3D Graphics*, 1995, pp. 105-106.

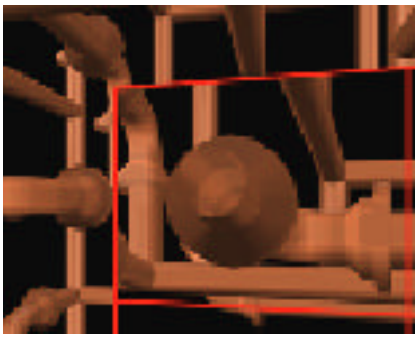
[8] Maciel P., Shirley P., "Visual Navigation of Large Environments Using Textured Clusters", *Symposium on Interactive 3D Graphics*, 1995, pp. 95-102.

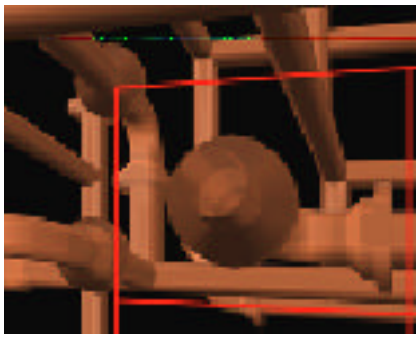
[9] McMillan L., Bishop G., "Plenoptic Modeling: An Image-Based Rendering System", *Computer Graphics (Proceedings of SIGGRAPH '95)*, 1995, pp. 39-46.

[10] Rossignac J., Borrel P., "Multi-resolution 3D Approximations for Rendering Complex Scenes", IBM T.J. Watson Research Center, Technical Report, Yorktown Heights, NY, February 1992.

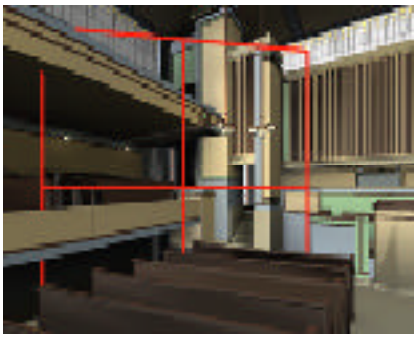
[11] Shade J., Lischinski D., Salesin D., DeRose T., Snyder J., "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments", University of Washington CSE Dept., TR#UW-CSE-96-01-96, to appear in *Proceedings of SIGGRAPH '96*, 1996.

[12] Teller S., Visibility Computation in Densely Occluded Polyhedral Environments, Ph.D. Thesis, UC Berkeley CS Dept., TR#92/708, 1992.

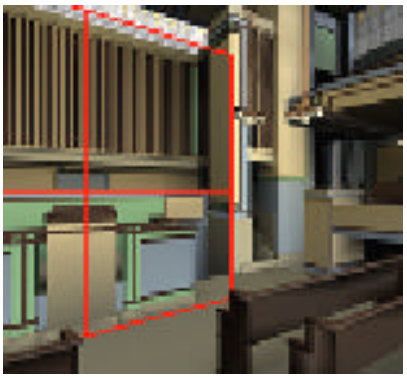


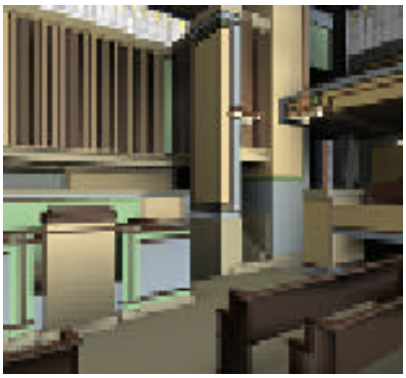


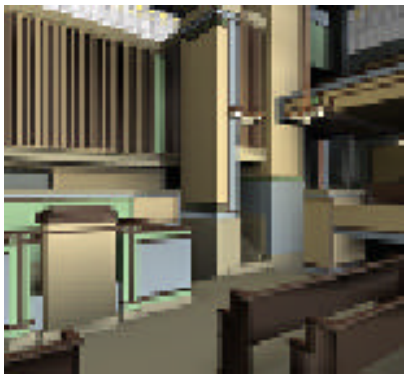
Color Figure 1: Geometry-Texture Border.
(Left) No morphing. (Right) Morphed geometry.



Color Figure 2: Texture-based Simplification. Church Model.

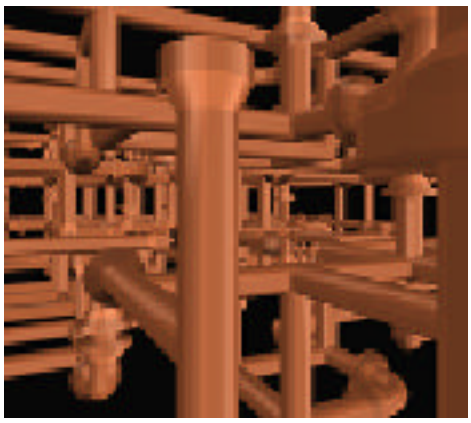






Color Figure 3: Morphing Sequence. Texture-to-Geometry Transition (left to right).

a)



b)

c)

Color Figure 4: Views from the recorded paths (pipes model, church model, AMR model). Each view contains multiple textures. See Section 5 for a more detailed description of the paths.