Appeared: Computer & Graphics, Elsevier Science, Vol 22:1, pp. 71-81, 1998

Smooth Transitions in Texture-based Simplification

Daniel G. Aliaga, Anselmo A. Lastra

Department of Computer Science

University of North Carolina

Chapel Hill, NC 27599-3175

{aliaga | lastra}@cs.unc.edu

Abstract

We are investigating techniques for providing smooth transitions when simplifying large, static geometric models with texture-based representations (or impostors). Traditionally, textures have been used to replace complex geometry, for example the books on a shelf or the complex foliage on a tree. Using textures in a more general manner is a relatively new area of research. The key idea is that 2D image textures can be used to temporarily represent 3D geometry. Rendering speed is increased if a replacement texture can be used for several frames, since textures can be rendered quickly and independently of model complexity. Because a texture is only correct from a single viewpoint, visual discontinuities, such as misalignment with adjacent geometry, begin to appear as the texture is used from other viewpoints. Previous approaches have controlled these errors by re-rendering the textures often or providing a large set of precomputed textures. We have improved upon these methods by developing algorithms for: (a) providing continuous imagery across borders between geometry and sampled textures at all times; (b) providing smooth dynamic transitions between geometry and texture.

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

1. Introduction

Large and complex 3D models are required for applications such as virtual environments, architectural walkthroughs and flight simulators. Even with high-end computer graphics systems, it is not possible to render all the geometry of these arbitrarily complex scenes at highly interactive rates. This has led to extensive work in 3D model simplification methods, such as:

- Visibility and occlusion culling algorithms: these techniques conservatively determine the visible subset of a model [1][2][3][14] from any viewpoint (or the non-visible subset as is the case for occlusion culling [8][23]). These subsets of the model can be rendered faster than the full data set. Unfortunately, these algorithms alone cannot simplify enough 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: these methods generate multiple instances of the objects in the model, each of varying complexity. The run-time system then chooses [10] the best

objects based on distance to the object, desired image quality and performance. A large body of work has been published in this area ([7][9][19] to cite a few) and also in dynamic LOD generation [11][15] and dynamic tessellation of curved surfaces [12]. Many of the above algorithms require some manual intervention for generating the multiple LODs and cannot easily simplify scenes with a large number of visible objects. Furthermore, level-of-detail algorithms reduce the geometric complexity of objects, at the cost of losing shading and color details as well as geometric detail. For example, the facade of a large building contains windows, ornaments, bricks, portals, etc. Reducing the geometric complexity of the facades in order to reach a desired rendering speed, will eventually remove these visible details causing a noticeable loss of quality in the scene.

- A relatively new simplification approach is to dynamically represent static 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. [16] developed a system that employs texture clusters and other *impostors* to render regions of the model more efficiently. The system precomputes a hierarchy of representations (multiple geometrical LODs and textures sampled from different viewpoints) and chooses which representation to use at run-time. [20] and [21] proposed a hierarchical *image caching* system. It dynamically computes a hierarchy of images representing nodes in a spatial partitioning tree. Both systems work for outdoor environments or where there are distinct ìclustersî of detail.

There are a few problems present in systems that replace geometry with textures. The first one occurs because a texture represents an arbitrary subset of the model from a single viewpoint and subsequently changing the viewpoint causes the image displayed by a texture to be incorrect (unless image warping is used [17]). Consequently, the geometry surrounding a texture does not match the geometry sampled in the texture causing a discontinuity or ìcrackî to appear. Previous methods have either ignored the discontinuities or used an error metric to decide whether to resample the texture or display another texture from a set of precomputed textures.

The second problem occurs when switching between geometry and textures. Once the viewpoint has moved from the point where the texture was sampled, a transition from geometry to texture (or vice versa) will cause a sudden jump in the image. Therefore, we need to define transitions to smoothly change the geometry into textures and textures back into geometry.

The third problem occurs as we switch between texture samples for the same region of the model. 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. In this paper, we have not addressed this problem but look to image warping for possible solutions [18].

We present solutions to the first two problems with methods that maintain continuous borders between static geometry and opaque textures and remove the sudden jump when switching between geometry and textures. This gives us more freedom to place textures in a model although the resulting imagery will be (slightly) inaccurate. We have implemented two testbed systems which use these ideas for indoor architectural walkthroughs [2][3]. A considerable performance benefit is that our approach takes advantage of the texturing hardware that has become standard on many computer graphics architectures, even on low-end systems.

The following section presents our geometry-warping solution to the above problems. Section 3 describes the geometry-warping operation in the context of a system where the user selects regions to be replaced with textures. Section 4 describes geometry warping in the context of cells and portals. Section 5 summarizes performance results of the two systems. Finally, Section 6 ends with some conclusions and future work.

2. Geometry Warping

A texture (or impostor) is a snapshot of the model from a single viewpoint (*texture-viewpoint*). In a typical texture-based simplification system, we temporarily insert a quadrilateral into the model onto which we texture-map a snapshot of the model (Color Figure 1). The image of the texture is only perspectively correct when viewed from the texture-viewpoint. Thus, when the eye moves from the texture-viewpoint, the geometry adjacent to and surrounding the texture appears discontinuous with the matching image in the texture. Furthermore, if we were to return the texture to geometry (or vice versa), we would see a sudden jump in the image. Color Figures 3(a-c) illustrates how the discontinuity increases as we move away from the texture-viewpoint. Color Figures 5(a,c) shows the difference between a sampled texture and the actual geometry from the same viewpoint.

To compensate for this discontinuity, two general approaches exist:

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

The former case corresponds to image warping [4][5][17] 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.

The second 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) the geometry warp operation can be efficiently performed using the graphics transformation stack, (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 reconstruction, and ìempty areasî produced when previously occluded regions becoming visible with no rendering information available for the newly visible pixels. Although our method could be considered less ìrealisticî than warping the texture, it takes advantage of the fact that geometry is re-rendered every frame anyway, so by slightly modifying the geometry you are able to use static textures and achieve higher frame rates. Color Figures 4(a-c) show the same sequence of frames as with Color Figures 3(a-c) but now we have warped the geometry adjacent to the texture to match the texture. Furthermore, by interpolating the geometry visible within the texture from its projected position on the texture to its correct position, we can produce a smooth transition between the images in Color Figures 5(a,c).

We obtain an overall pleasing solution. We have traded-off the sudden discontinuities by (slightly) distorting the overall image, although we can guarantee that the geometry near the viewpoint is unaffected. The error in the image is proportional to the distance between the current viewpoint and the original texture viewpoint.

2.1 Partitioning the Geometry

The textured quadrilateral partitions the model into 3 subsets of geometry*: near geometry* (unaffected by the warping operation), *texture geometry* (geometry behind the texture that will typically be culled) and *surrounding geometry* (geometry surrounding all four edges of the texture). The warping operation allows us to change the surrounding geometry to maintain C0 continuity (i.e. positional continuity) with the image of the texture. This implicitly achieves C0 continuity of the texture and surrounding geometryís border with the near geometry. It does not require warping the near geometry (near geometry piercing the texture will also appear C0 continuous). We can also render the texture geometry just as it appears on the texture even though we are not at the texture-viewpoint, and smoothly interpolate the texture geometry and surrounding geometry back to their correct position.

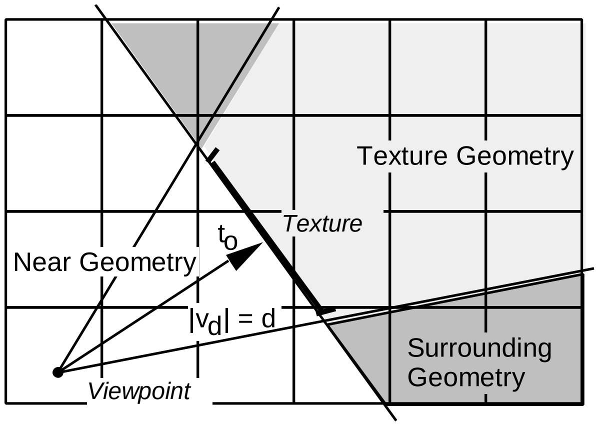


Figure 1: Model Partitioning for Geometry Warping. Each box corresponds to a space-partitioning box. The boxes are classified: near, texture and surrounding. Intersected boxes can be optionally partitioned.

2.2 Geometric Continuity

In order to maintain geometric continuity, we need to warp the surrounding geometry to match the texture. Let the four vertices of the textured quadrilateral and the texture-viewpoint define the view frustum used to create the texture. We denote this view frustum by [$\mathbf{v_0}$-$\mathbf{v_3}$, $\mathbf{p_a}$]. The current view frustum can be similarly defined by [$\mathbf{v_0}$-$\mathbf{v_3}$, $\mathbf{p_b}$], where $\mathbf{p_b}$ is the current viewpoint. Despite having our eye at point $\mathbf{p_b}$, we need to project the surrounding geometry onto the texture plane as if we were at $\mathbf{p_a}$. We use an inferred perspective transformation to warp the projection plane, defined by frustum [$\mathbf{v_0}$-$\mathbf{v_3}$, $\mathbf{p_a}$], to appear as if it were seen from the current viewpoint $\mathbf{p_b}$.

The current view frustum, [$\mathbf{v_0}$-$\mathbf{v_3}$, $\mathbf{p_b}$], can be expressed using a model-space transformation $\mathbf{M_b}$ and a projection $\mathbf{P_b}$. Similarly, the textureís view frustum can be defined by a model-space transformation $\mathbf{M_a}$ and a projection $\mathbf{P_a}$. The final (warped) frustum is defined by $\mathbf{M_w} = \mathbf{P_a M_a}$ and $\mathbf{P_w} = \mathbf{W_{ab}}$, where $\mathbf{W_{ab}}$ is the perspective warp from $\mathbf{p_a}$ to $\mathbf{p_b}$. This sequence of transformations is illustrated in Figure 2.

To construct the warp matrix $\mathbf{W_{ab}}$, we employ a four-corner mapping (assuming planar quadrilaterals). We project the vertices $\mathbf{v_0}$-$\mathbf{v_3}$ using $\mathbf{P_a M_a}$ and $\mathbf{P_b M_b}$ and use their projected positions to construct the four corner mapping. In order to resolve occlusion properly, we must set up the matrix $\mathbf{W_{ab}}$ so that the final transformation matrix will produce z-values that correspond to the projection onto [$\mathbf{v_0}$-$\mathbf{v_3}$, $\mathbf{p_a}$]. In essence, we let the projected z-value pass through the warp unaffected. We can accomplish this by placing the nine coefficients of the warp matrix as follows:

$$\begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & i \end{bmatrix}$$

2.3 Smooth Transitions

If we wish to return a texture to geometry (or vice versa), we can smoothly interpolate over time the texture geometry from its projected position on the texture plane to its correct position. We accomplish this by augmenting the above warp operation to use intermediate view frustums. Namely, we re-project the texture geometry using the interpolated view frustum [$\mathbf{v_0}$-$\mathbf{v_3}$, $\mathbf{p_i}$] where $\mathbf{p_i}$ is a point along the line segment $\mathbf{p_a}$-$\mathbf{p_b}$. Then, we apply the inferred perspective transformation to warp the projection plane, defined by frustum [$\mathbf{v_0}$-$\mathbf{v_3}$, $\mathbf{p_i}$], to appear as if it were seen from the current viewpoint $\mathbf{p_b}$.

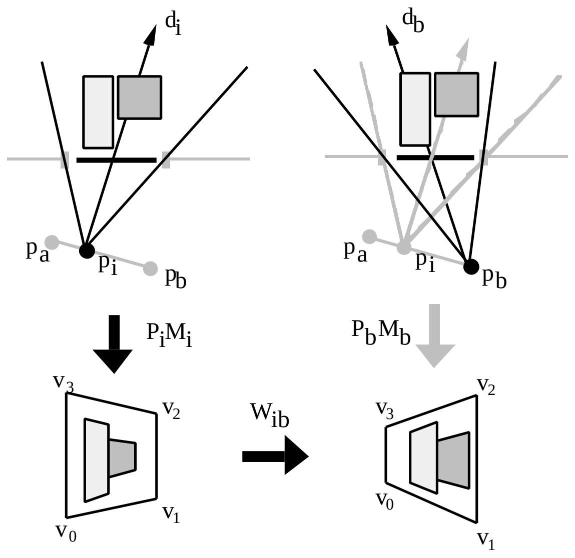


Figure 2: Sequence of Transformations for Geometry Warping. We first project along d_i onto the plane of the quadrilateral using the interpolated view frustum or the texture view frustum (in the latter case, $p_i=p_a$, $P_i=P_a$ and $M_i=M_a$).

2.4 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. We will discuss the issues involved with having multiple textures present and how it affects geometric continuity and smooth transitions. We assume that textures do not overlap and, without loss of generality, we can divide the textures into those with a common viewpoint and those with different viewpoints. We explore each of these categories below.

2.4.1 Common Viewpoint

Textures that share a viewpoint can each have a different view direction and view depth. Since each texture contains a complete view of the model along its view direction, we are only interested in textures whose views do not overlap. The view depth parameter will control the amount of near geometry. With these criteria is mind, we can further subdivide this category into adjacent textures and textures with gaps between them.

First, we will address adjacent textures. Adjacent textures form a single large texture with piecewise planar components. They can be used to create textures that cover a complex region of the model or even completely surround the viewpoint. Geometric continuity between adjacent textures can easily be maintained (except for the case in Figure 3c, which should be avoided).

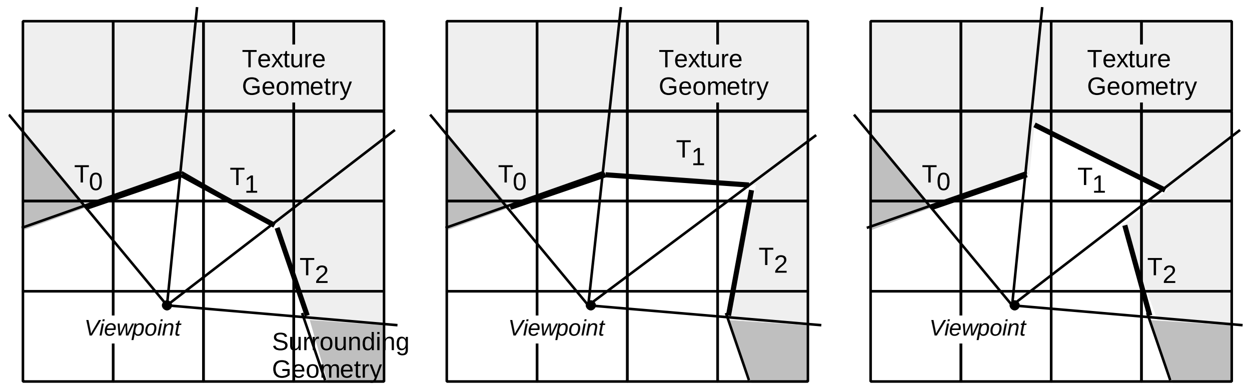


Figure 3: Common viewpoint, adjacent textures. (a) Textures at equal view depth, thus will maintain geometric continuity. (b) Varying view depth but same at the seams, geometric continuity maintained. (c) Discontinuous view depth, geometric continuity not maintained. This case can easily be avoided.

Maintaining smooth transitions, on the other hand, is slightly different than with a single texture. If a texture at the edge of a string of adjacent textures is returned to geometry, the vertices must be interpolated between the previous edge-texture plane and the new edge-texture plane (Figure 4a). If a texture in the middle of a string of adjacent textures is returned to geometry, the vertices need to be warped to match the projection of the texture being removed. No interpolation occurs (Figure 4b).

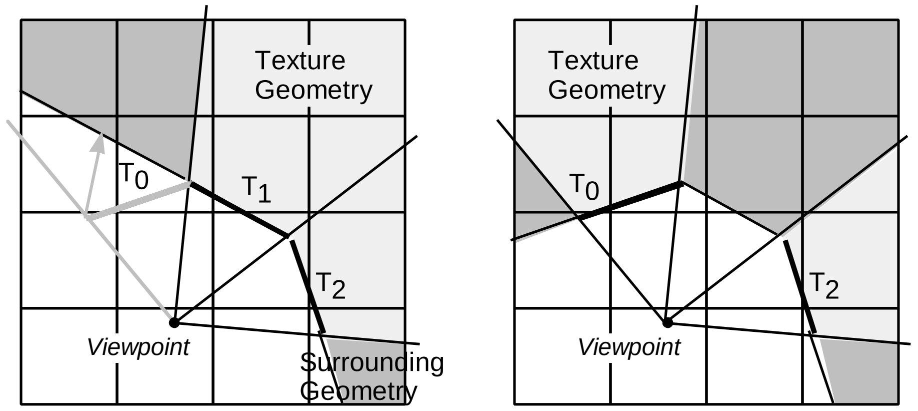


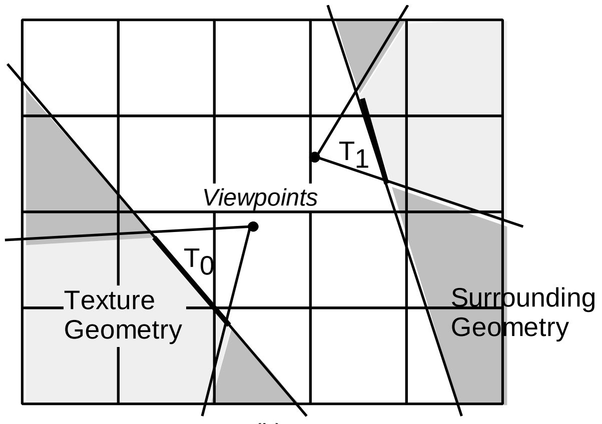
Figure 4: Smooth Transitions. (a) Removing a texture at the edge of a string of adjacent textures, interpolation required. (b) Removing a texture in the middle of a string of adjacent textures, no interpolation.

For textures with a gap between them, a (virtual) texture needs to be added that spans the gap. Now they can be treated as adjacent textures. It is worth noting that for two textures (a ìleftî texture and a ìrightî texture) with a gap between them and very different view depth values, the geometry can be warped to match both textures but the distortion introduced might be very apparent from certain view directions. For example, assume the left texture was defined at a significantly closer distance to the viewpoint 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.

2.4.2 Different Viewpoints

If multiple textures are created from different viewpoints each at a potentially different view direction and view depth, the geometry surrounding each texture must be warped before continuing on to create the next texture (Figure 5). Consequently, the first texture will have an image of unwarped geometry. Textures that are created from viewpoints and view directions that do not contain the geometry in the plane of the previous textures are using unwarped geometry. Subsequent textures that are created by using geometry surrounding the previous textures or using the textures themselves (thus increasing the texture depth complexity), will contain images of warped geometry. Thus, it might be the case that the distortion introduced by the warping operations will be magnified after several instances of textures from different viewpoints (at present, we have no metric to control this distortion). 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 case where a subset of the model geometry, warped for a particular texture, intersects with another warped geometry subset is similar to the case of two textures using a common viewpoint but different view depth values. Both warped geometry subsets will have to be warped to match the textures simultaneously.



(b)

Figure 5: Multiple Textures. Different viewpoints and distances.

3. User-Selected Textures

In this section, we describe the first of our two testbed implementations that employ geometry warping to provide geometric continuity and smooth transitions when replacing geometry with textures.

In this system, as the user moves through the model, distant (and visible) subsets of the model can be replaced with a texture [2]. The texture geometry is culled from the model and consequently significant rendering performance increases can be accomplished. The texture is used to represent the subset of the model from the current viewing area. Unfortunately, once the viewpoint moves again, geometry surrounding the texture will appear discontinuous with the texture. In order to maintain a continuous border between the texture and the geometry around it, we employ the geometry warp operation so that the surrounding geometry matches its corresponding points in the texture and maintains C0 continuity.

After replacing a subset of the model with a texture, the user cannot walk forward beyond the texture plane without returning the subset to geometry. Geometry near the viewpoint is rendered normally. The geometry surrounding the texture maintains a continuous border with the texture but is not rendered completely accurately. For many applications, 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 (e.g. 5).

The textures for the transitions can be computed on demand or they can be precomputed. If they are computed on demand, the eye and texture-viewpoint coincide and there is no need to perform the first geometry-to-texture transition. On the other hand, if the textures have been precomputed (or a cache of the most recently used textures exists), the eye will most likely not be at the texture-viewpoint, thus geometry-to-texture transitions are needed.

The following two sections will describe in more detail the geometry-to-texture and texture-to-geometry transitions.

3.1. 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. For our implementation, we adopted the following simple 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. We employ a uniform spatial partitioning of the model to easily determine which boxes (and thus what geometry) are behind the texture plane [6]. 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 viewpoint along the view direction (Figure 1). The subset of the model behind the texture will be the texture geometry.

2. We push the near clipping plane back to coincide with the texture plane. Then, we render the texture geometry and copy the rendered image from the framebuffer to texture memory. 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 precomputed lighting: static directional lights or precomputed radiosity illumination).

To reduce texture memory requirements, the texture can be sampled at a resolution lower than the framebufferís 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 pre-filtered to achieve apparent high-quality antialiased imagery (in addition to MIP mapping the texture).

3. The texture 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 ìbehindî 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. We employ the geometry warp operation over several frames to change the geometry behind the texture plane (that has not been culled) and the surrounding geometry to match the texture. The intermediate view frustums are created using viewpoints along the line between the current viewpoint and the texture-viewpoint. At the end of the transition, the texture is displayed instead of the warped geometry.

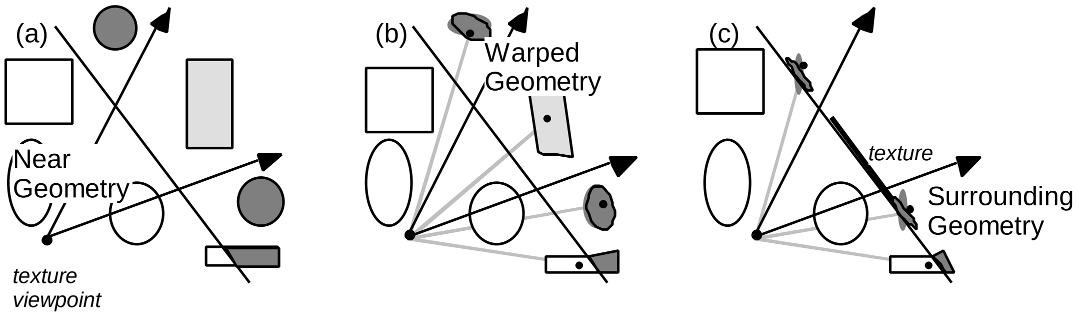


Figure 6: Warping Sequence. A geometry-to-texture transition goes from (a) to (c). At the end of the transition, the texture is introduced. A texture-to-geometry transition goes from (c) to (a). From the texture-viewpoint, the objects look the same at all times.

3.2 Texture-to-Geometry Transition

1. The texture geometry is reintroduced into the model. The vertices are set to their projected position on the texture plane.

2. The texture geometry and surrounding geometry is warped from their projected position on the texture plane to their correct position over several frames (note that if the texture plane is currently not in the view frustum, an instantaneous transition can be performed).

Once a texture has been computed it might undergo various geometry-to-texture and texture-to-geometry transitions. As mentioned before, all subsequent transitions (after the first geometry-to-texture transition) will generally be from viewpoints other than the texture-viewpoint. Thus the surrounding geometry is gradually warped from its correct 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 warped.

4. Portal Textures

This system takes advantage of the fact that we can divide architectural models into cells based on the location of walls and other opaque surfaces [1][22]. Each cell contains a list of portals, each of which defines an opening through which an adjacent cell may be seen. Figure 7(a) shows the top view of a cell-partitioned model. The viewpoint is inside the *view* cell. Since the view frustum only intersects a subset of the portals of the view cell, the cells attached to each visible portal are recursively traversed to compute all of the visible cells.

Since the model contains the location of all portals, we can compute textures to be placed in the location of the otherwise transparent portal openings (Color Figure 2) [3]. At run-time, we render the view cell normally. All visible portals of the view cell are rendered as textures and no geometry from adjacent cells is actually rendered, despite being visible. Figure 7(b) illustrates the reduced set of cells that need to be rendered. As the viewpoint approaches a portal, we switch to rendering the geometry of the cell behind the portal. Once the viewpoint enters the adjacent cell, it becomes the view cell and the previous cell will now be rendered as a portal texture.

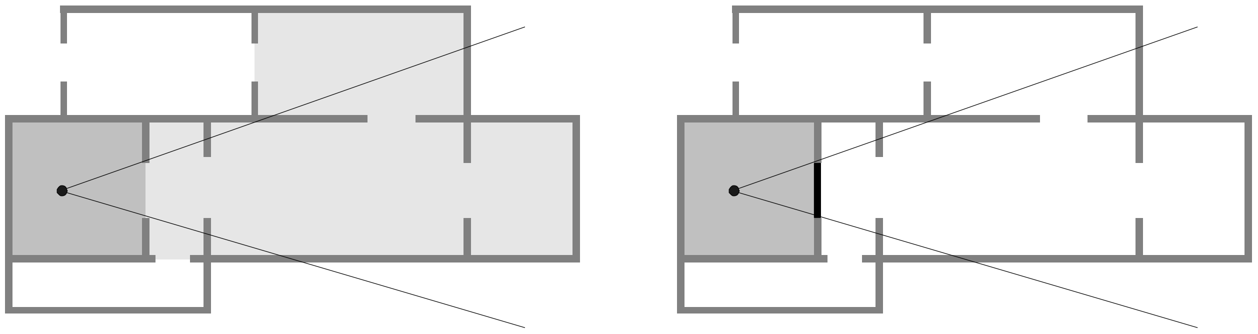


Figure 7(a-b): (a) Portal Culling. (b) Portal Texture Culling.

We can store a variable number of texture samples per portal. We chose to sample texture-viewpoints along a semicircle in front of each portal, at the typical portal viewing height. As we approach the texture and it switches to geometry we see an abrupt jump in the image, particularly for a single texture per portal (which might be the case if we have a very tight texture memory budget). We employ the geometry warp operation to provide smooth transitions when switching between a portal texture and the geometry of the cell behind the portal.

5. Performance Results

The systems are written in C/C++. All primitives are tessellated into triangles at start-up time. The first system, running on an SGI Onyx equipped with Reality Engine^2 graphics (16 MB of texture memory), was tested with three models: a procedurally generated pipes model, a radiosity-illuminated church[1] and an auxiliary machine room of a nuclear submarine[2]. The portal textures system, running on an SGI Onyx with Infinite Reality graphics (64 MB of texture memory), was tested with two models: a large single-story house and a smaller two-story house.

5.1 Models for User-Selected Textures

We used a separate program to procedurally generate a complex *pipes* structure. This program recursively created an array of pipes using replication and instancing. It is very hard to perform visibility culling (for examples, cells and portals) and level-of-detail computations on such a model. It is also representative of the complexity present in some ship models. The pipes model we used has 205,000 triangles.

We employed a radiosity-illuminated *church* as an example of a single room that is visually complex (158,000 triangles). Geometric-based LOD algorithms that 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.

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

Finally, the auxiliary machine room (*AMR*) is an example of a visually complex model with high depth complexity (273,000 triangles). The model contains the depth complexity of the pipes model and the shape detail of the church model. This model is actually a notional model, not that of an actual submarine.

5.2 Models for Portal Textures

We tested our system using two architectural models. The first model, named *Brooks House*, is that of a large one-story house modeled using 528,000 triangles. The second model, the *Haunted House*, is of a two-story house and consists of 214,000 triangles. Both of these models have been divided into cells and portals. The more complex Brooks House has 19 cells and 52 portals, while the Haunted House has 7 cells and 12 portals.

5.3 Recorded Paths

For each model, the location of several textures was predetermined by the user or fixed to be at the portals. We recorded various paths through the models (spline-interpolated paths and some viewpoints captured from actual user paths). Interactive performance is significantly improved when textures are introduced. The distortion caused by the warping is not very noticeable as you can see in the color figures (and in our demonstration video). Furthermore, for the user-selected textures system no discontinuities or ìcracksî are perceivable at the border between geometry and texture. Since we are using static lighting, the texture and geometryís colors should match at the border (we have observed a small disparity that we suspect is a consequence of the texture sampling or of exactly how the texturing hardware processes the texel colors).

Model	Average Speedup	Comments
Pipes	9.2	3 textures total
Church	3.3	3 textures total
AMR	6.5	2 textures total
Brooks House	3.3	vs. portal culling
Haunted House	2.6	vs. portal culling

Table 1: Performance Summary.

Table 1 summarizes the speedups we obtained. The first three entries were obtained using our user-placed texture system. At any time, the textures can be smoothly changed back to geometry (with the corresponding decreases in performance). The second set of numbers is from the portal textures system and are the speedups versus traditional portal culling. Textures were placed at all distant portals. We computed the speedups based on the average frame rates in different parts of the model. However, we do not regard the ìaverage frame rateî as necessarily the best means to measure the rendering performance because we may prefer to have a more even frame rate. 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 overall performance.

6. Conclusions and Future Work

We have presented a method to provide continuous borders between geometry and discretely sampled textures at the cost of (slightly) distorting the geometry. This method also provides smooth transitions when switching between geometry and textures.

We are currently exploring algorithms to decide automatically when to perform the transitions. A cost-benefit style function [10] 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 ìcachingî distant geometry into texture-based representations.

Furthermore, we are developing methods to (quickly) measure the perceptual error introduced by the warping 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 warping operations.

In addition, we are investigating ways to remove the restriction of using precomputed lighting. For example, by including per-texel normals [13] 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 help of Gary Bishop, Frederick J. Brooks, Bill Mark, Michael North, Mary Whitton and Peggy Wetzel. The Brooks House model is courtesy of generations of students and members of the UNC Walkthrough team. We would like to thank Dave Luebke and Michael Goslin for the Haunted House model. This work was supported in part by NSF MIP-9306208, NIH/NCRR P41RR02170-13, DARPA Order No. A410 and DABT63-93-C-0048 (ìEnabling Technologies and Application Demonstrations for Synthetic Environmentsî) and a UNC Dissertation Fellowship.

REFERENCES

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

2. Aliaga D., Visualization of Complex Models Using Dynamic Texture-Based Simplification, *IEEE Visualization*, 101-106 (1996).

3. Aliaga D. and Lastra A., Architectural Walkthroughs Using Portal Textures, *IEEE Visualization*, (1997).

4. Chen S. E. and Williams L., View Interpolation for Image Synthesis, *Computer Graphics (SIGGRAPH '93)*, 279-288 (1993).

5. Chen S. E, QuickTime VR - An Image-Based Approach to Virtual Environment Navigation, *Computer Graphics (SIGGRAPH '95)*, 29-38 (1995).

6. Clark J., Hierarchical Geometric Models for Visible Surface Algorithms, *CACM*, Vol. 19(10), 547-554 (1976).

7. Cohen J., Varshney A., Manocha D., Turk G., Weber H., Agarwal P., Brooks F. and Wright W., Simplification Envelopes, *Computer Graphics (SIGGRAPH '96)*, 119-128 (1996).

8. Coorg S. and Teller S., Real-Time Occlusion Culling for Models with Large Occluders, *Symposium on Interactive 3D Graphics*, 83-90 (1997).

9. DeHaemer M. and Zyda M., Simplification of Objects Rendered by Polygonal Approximations, *Computer Graphics*, Vol. 15(2), 175-184 (1991).

10. Funkhouser T. and Sequin C., Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environmentî, *Computer Graphics (SIGGRAPH '93)*, 247-254 (1993).

11. Hoppe H., View-Dependent Refinement of Progressive Meshes, *Computer Graphics (SIGGRAPH '97)*, 189-198 (1997).

12. Kumar S., Manocha D., Zhang H. and Hoff K., Accelerated Walkthrough of Large Spline Models, *Symposium on Interactive 3D Graphics*, 83-90 (1997).

13. Lastra A., Molnar S., Olano M. and Wang Y., Real-Time Programmable Shading, *Symposium on Interactive 3D Graphics*, 59-66, (1995).

14. Luebke D. and Georges C., Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets, *Symposium on Interactive 3D Graphics*, 105-106 (1995).

15. Luebke D. and Erikson C., View-Dependent Simplification of Arbitrary Polygonal Environments, *Computer Graphics (SIGGRAPH '97)*, 199-208 (1997).

16. Maciel P. and Shirley P., Visual Navigation of Large Environments Using Textured Clusters, *Symposium on Interactive 3D Graphics*, 95-102 (1995).

17. McMillan L. and Bishop G., Plenoptic Modeling: An Image-Based Rendering System, *Computer Graphics (SIGGRAPH '95)*, 39-46 (1995).

18. Rafferty M., Aliaga D. and Lastra A., 3D Image Warping in Architectural Walkthroughs, UNC TR# 97-019, submitted for publication, (1997).

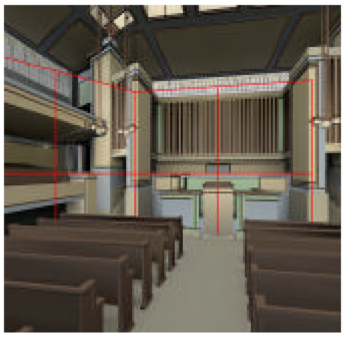
19. Rossignac J. and Borrel P., Multi-resolution 3D Approximations for Rendering Complex Scenes, Technical Report, IBM T.J. Watson Research Center, Yorktown Heights, NY, (1992).

20. Schaufler G. and Stuerzlinger W., Three Dimensional Image Cache for Virtual Reality, Computer Graphics Forum (Eurographics '96), Vol. 15(3), 227-235 (1996).

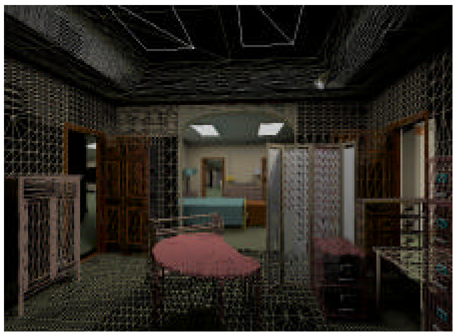
21. Shade J., Lischinski D., Salesin D., DeRose T., Snyder J., Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments, *Computer Graphics (SIGGRAPH ë96)*, 75-82 (1996).

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

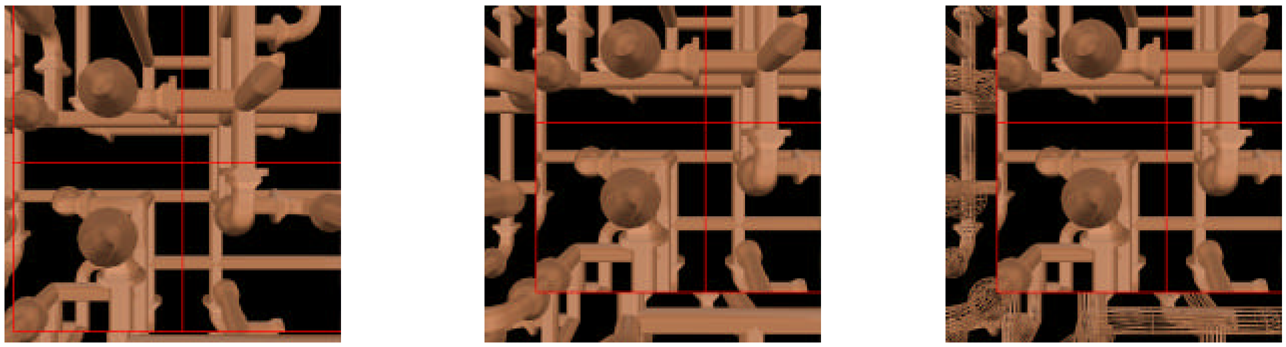
23. Zhang H., Manocha D., Hudson T. and Hoff K., Visibility Culling Using Hierarchical Occlusion Maps, *Computer Graphics* (*SIGGRAPH '97)*, 77-88 (1997).



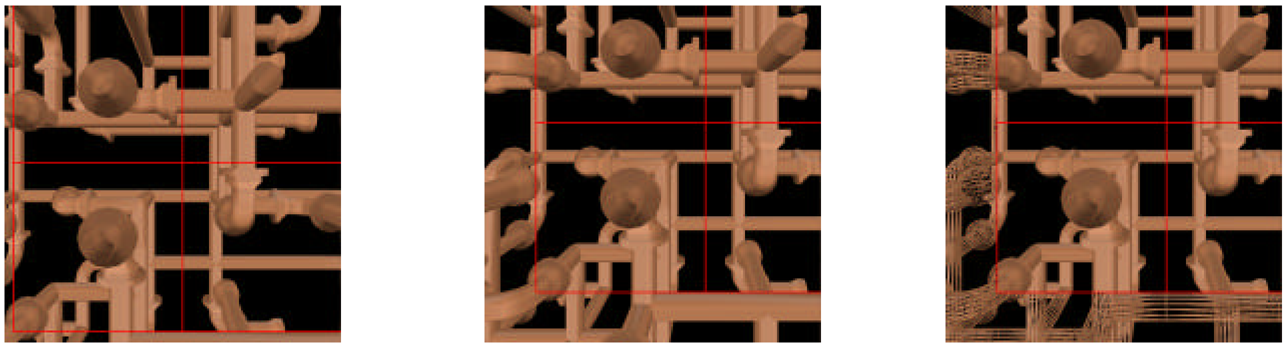
Color Figure 1: Church Model. Geometry around textures (outlined in red) is warped.



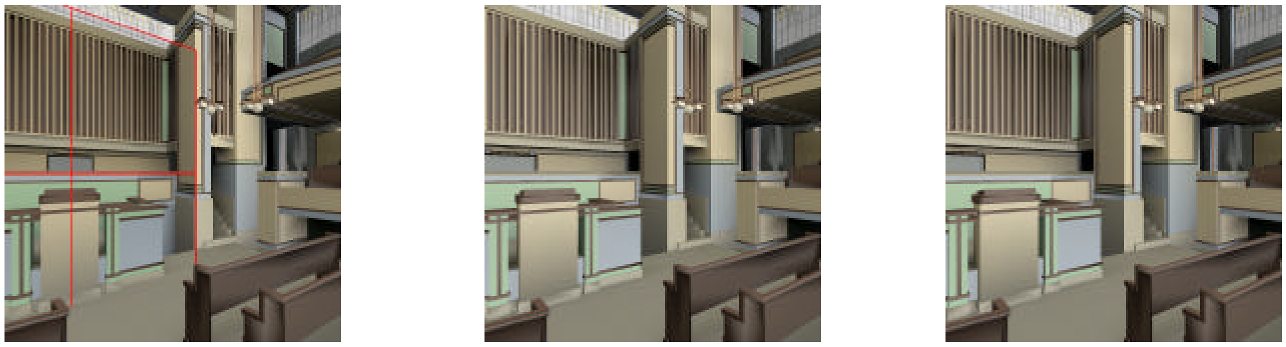
Color Figure 2: Brooks House Model. Portals have been replaced with textures.



Color Figure 3: No geometry warping (texture outlined in red). (a-b) Eye moves away from texture-viewpoint. (c) For clarity, geometry is rendered in wireframe.



Color Figure 4: Geometry warping. (a-b) Eye moves away from texture-viewpoint. Notice the geometric continuity at the boundary. (c) For clarity, geometry is rendered in wireframe.



Color Figure 5: Texture-to-Geometry Transition. (a) Texture (outlined in red) is about to be returned to geometry. (b) Midpoint of morphing sequence. (c) Geometry has been fully restored.