Dynamic Simplification using Textures

Daniel G. Aliaga
aliaga@cs.unc.edu
Computer Science Department
University of North Carolina at Chapel Hill

Abstract

We are investigating methods for simplifying large geometrical models for interactive walkthroughs 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.

Motivation

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][2]. 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 [3][4], which typically require 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 [5]. 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 applications. The error introduced is proportional to the distance from the original texture sample point. The system can bound the error by resampling the texture as needed.

We encountered two major problems in developing this rendering system. First, since a texture represents a 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 [6]). Consequently, the geometry surrounding the texture does not match the geometry sampled in the texture. This is especially noticeable at the border between the geometry and the texture. We present a solution to the boundary 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 the model.

Second, if the viewpoint has changed from the texture sample point, a straightforward transition to geometry will cause a sudden jump in the image (as in [5]). Therefore, we need to define transitions to smoothly change geometry into texture and texture back into geometry. We present smooth transition operations to convert geometry into textures (and vice versa).

Texture-based Simplification

We have devised a simplification method which dynamically replaces arbitrary portions of a model with textures and morphs the near 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 difficult to apply.

Our simplification method has a preprocessing phase and a run-time phase. The preprocessing phase statically partitions the model into a 3D grid of cells. Space partitioning and view frustum culling are used so that the amount of work to be done for transitions to texture or back to geometry is proportional to the number of visible primitives.

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 three sections describe the major elements of the run-time phase:

Continuous Border

At run-time, a distant (and visible) subset of the model is replaced by a texture. Once the viewpoint moves, additional geometry becomes visible. In order to maintain a continuous border between the texture and the geometry that became visible 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 [6] in which the sampled texture has depth information and is warped every frame to match the appropriate reprojection. 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 advantageous because: (a) it allows texturing hardware to be efficiently used, (b) the texture does not need to be warped every frame, (c) it does not introduce visible artifacts as the viewpoint changes as may be the case with image warping, and (d) all of the work is performed at one time (at geometry-to-texture transition time or as a precomputation).

pipes un-morphed (wireframe)

pipes morphed (wireframe)

Figure 1: Geometry-Texture border. Texture and un-morphed geometry (left). Texture and morphed geometry (right).

Geometry-To-Texture Transition

Replacing a distant (and visible) subset of the model with a texture corresponds to a geometry-to-texture transition and is a straightforward operation. The following series of events occur:

1. The rendered image of the geometry is copied into texture memory. A texture object (a texture-mapped quad covering the subset of the model currently visible) is added to the model. To reduce texture memory requirements, the texture can be sampled at a resolution lower than the framebuffer's resolution.
2. The geometry represented by the texture is culled out of the model. Since the model is space partitioned, this is a fast operation. All cells currently contained in the viewing volume beyond the texture plane distance are removed from the active set of cells of the model. The cells that intersect the viewing volume can be partitioned or not culled at all.
3. The geometry in front of the texture plane is rendered normally. The geometry behind the texture (that has not been culled) and the geometry surrounding the texture plane object are morphed to match the geometry represented by the texture (this is the continuous border problem). The surrounding geometry is projected onto the texture plane (space partitioning and view frustum culling make this operation proportional to the visible geometry).

After a geometry-to-texture transition, the following are true:

- The user cannot walk forward beyond the texture plane (without performing a texture-to-geometry transition).
- Geometry near the viewpoint is rendered normally. Surrounding geometry maintains a continuous border with the texture and near geometry but is not rendered completely accurately. This is not a bad tradeoff for the improved performance.

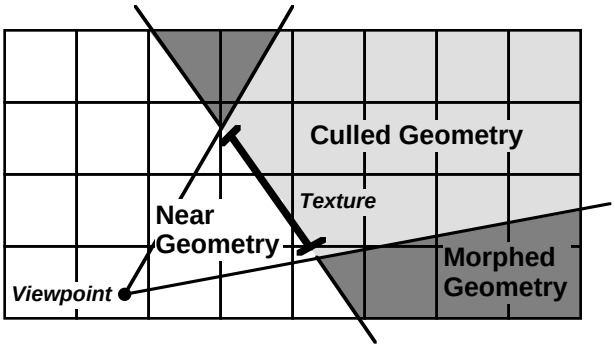

Figure 2: Model partitioning strategy. Each box corresponds to a space-partitioning cell. The cells are classified: near, culled, and morphed. Intersecting cells can optionally be partitioned.

Texture-to-Geometry Transition

A texture is sampled from a specific viewpoint (and view direction). If the viewpoint at the time of the texture-to-geometry transition is the same as the sample viewpoint, the transition is instantaneous. In general, this is not the case. Thus a smooth transition is gradually performed over the next few frames. A texture-to-geometry transition consists of the following steps:

1. The geometry represented by the texture is reintroduced into the model at its projected position on the texture plane.
2. The vertices of the geometry are morphed from their projected position and orientation on the texture plane to their correctly projected position (note that if the texture plane is not currently in the viewing volume, an instantaneous transition can be performed).
3. The vertices of the surrounding geometry are also morphed to their correctly projected position. Since space partitioning and view frustum culling are used, only the visible geometry is actually morphed.

Results

We implemented this method on a graphics workstation using OpenGL. The method 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 and the geometry behind the texture was culled. For a better

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

view of what the simplified models look like, including the transitions, please refer to the video.

unity with texture outlines

Figure 3: Church model with two textures (outlined in red) and surrounding geometry morphed to match the textures.

Conclusions

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 (i.e. portals), but if the geometry inside one room is still very complex there is inherently a large amount of geometry in the viewing volume. 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 algorithm 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 and can render scenes quickly regardless of visual complexity. 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 methods to decide automatically when to perform the transitions. A cost-benefit style function determines when and where in the scene the transitions should occur in order to maintain an interactive frame rate. This will enable us to walkthrough complex models while automatically "caching" distant geometry into texture-based representations.

References

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

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

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

[4] Rossignac J., Borrel P., *"Multi-resolution 3D Approximations for Rendering Complex Scenes"*, IBM T.J. Watson Research Center, Yorktown Heights, NY.

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

[6] McMillan L., Bishop G., *"Plenoptic Modeling: An Image-Based Rendering System"*, pp. 39-46, SIGGRAPH '95, 1995.