Portal Textures:
Texture Flipbooks in Architectural Models

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

November 4, 1996

Abstract

This paper outlines a method to use (compressed) flipbooks of textures to represent portals in 3D models. A portal is a generalization of windows and doors. It connects two adjacent cells (or rooms). This method accelerates portal visibility culling by reducing the depth of portal sequences to one. Each portal of the current cell that is some distance away from the viewpoint is rendered as a texture. The portal texture returns to geometry when the viewpoint gets too close to the portal. This way all portal sequences (not too close the viewpoint) have a depth complexity of one. The size of each texture and distance at which the transition occurs is configurable for each portal.

1. Introduction

Large and complex 3D models are required for applications such as virtual environments, architectural walk-throughs and flight simulators. It is not possible to render all the geometry of these arbitrarily complex scenes at highly interactive rates, even with high-end computer graphics systems. This has led to extensive work in 3D model simplification methods. This paper outlines a method based on texture-based simplification.

The key behind texture-based simplification is that portions of the 3D model can be replaced with faster-to-render representations, namely 2D textures. The textures will display imagery that is an approximate representation of the underlying geometry. In fact, we can control exactly how accurate we wish the textures to be by regulating how often textures are resampled or how many textures are precomputed. This provides us with a mechanism to control the accuracy of the images we are seeing. Three major issues need to be considered when replacing geometry with textures:

- Where do we place the textures in the model?
- What is the error introduced by the texture-based representation (thus allowing us to bound the maximum error)?
- How do we manage simultaneously having rendered geometry and geometry approximated by a texture?

Answering these questions in general is hard. We have restricted ourselves to the visualization of large 3D architectural models which have been divided into cells and portals [1][2][3]. This subclass of 3D models allows us to formulate a set of concrete and efficient answers to the above questions. While there are other ways to simplify models using textures [4][5][6], this approach is interesting and provides good results.

The following section will describe how portal textures are placed in a cell-partitioned model. Section 3 will first overview the error introduced by the portal texture representation. Then, it will describe how multiple textures are sampled and how the best texture is dynamically chosen. Section 4 will explain how smooth transitions between portal texture and cell geometry are accomplished. Section 5 will describe the portal texture extension, called *pfPortalTextures*, to pfPortals [3]. Finally, Section 6 will end with some conclusions.

2. Portal Textures in a Cell-Partitioned Model

When viewing a 3D model from a specific viewpoint, the user typically only interacts with the local geometry. We can take advantage of this fact and render the near geometry accurately and use faster representations for the farther geometry. In the cells and portals framework, this corresponds to rendering the cell containing the viewpoint (view cell) as normal geometry and replacing the (transparent) portals from the view cell to all adjacent cells with textures. Each portal texture represents the geometry of the adjacent cell from a sample viewpoint but can be rendered much faster. The errors introduced are proportional to the distance of the current viewpoint from the texture sample points. These errors will be described in the next section.

Since the model contains the location of all portals, we precompute the portal textures. At run-time, we render the view cell normally. All portals of the view cell are rendered as textures and no adjacent cells are actually rendered, despite being visible. As the viewpoint approaches a portal, we (smoothly) 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. We have effectively reduced the rendering complexity to that of the view cell.

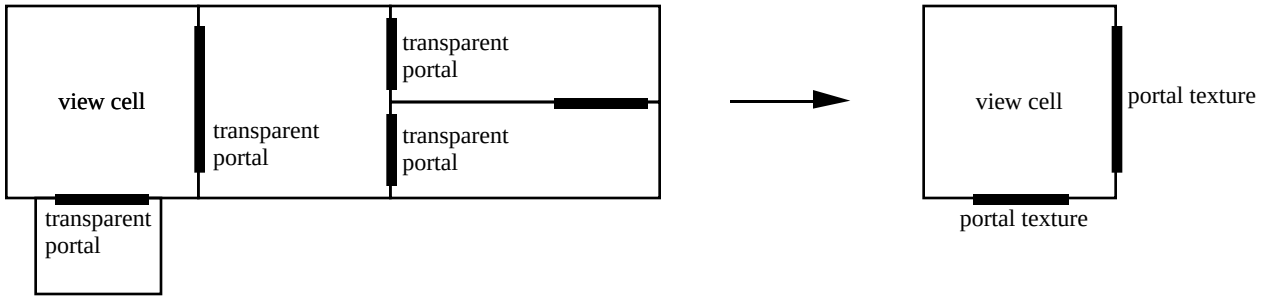


Figure 1: (a) Left - view cell and transparent portals, (b) Right - view cell and adjacent cells represented by portal textures

3. Choosing the Best Portal Texture

We will first present a method for computing the global representation *error* of a portal texture. This error is introduced once the viewpoint has been displaced from the current portal's texture sample point. Then, we will describe how we sample multiple portal textures and dynamically choose which portal texture to render.

3.1 Measuring the Global Pixel-to-Pixel Error

Consider a viewer looking at a portal that has been replaced with a texture. If the viewpoint is the same as when the texture was sampled, the viewer will not perceive any difference in the image (except for lighting issues, but for now we will defer this). Rotations about the viewpoint will also yield an apparently correct image. But, once the viewpoint translates in any direction, the picture displayed by the portal texture will no longer be accurate. The error will be proportional to the deviation of the viewpoint from the texture sample point and will also be proportional to any subsequent rotations. The portal texture will show a coherent image but will have incorrect occlusions and locations for the geometry of the adjacent cell.

Defining an exact objective pixel-by-pixel measurement of how the portal texture deviates from the correct image is difficult. A more practical measurement is to define a conservative bound on the error introduced.

An outline of how to perform a pixel-by-pixel error measurement is given here: each texel of the texture must also store its depth value relative to the portal texture plane (in addition to its RGB value). Now we consider each texel as a small particle (or polygon) for which we can compute its actual 3D position. Subsequently, during each frame we project this 3D particle onto the portal texture and measure its distance (in pixels) from its corresponding position on the sampled portal texture. We then sum up the distances of all the texel pairs to obtain a global error measure.

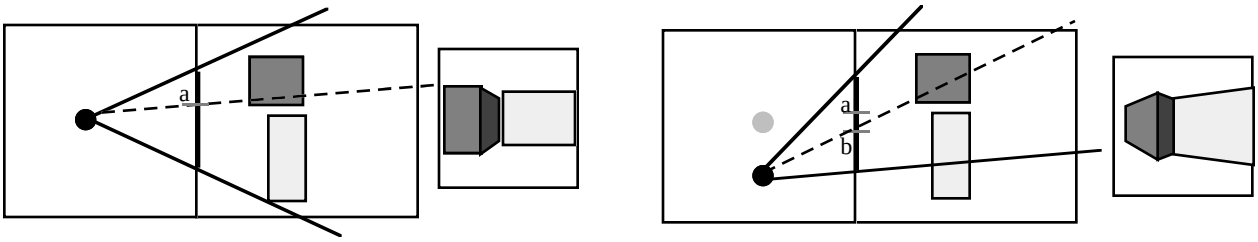


Figure 2: Example Pixel-by-Pixel Error Measurement, (a) Left - viewpoint same as texture sample point, (b) Right - viewpoint translated from texture sample point.

Figure 2a corresponds to viewing the adjacent cell from the texture sample point. The corner of the darker cube appears at location **a** on the texture (see dashed line). Figure 2b is the same cell viewed from a translated and rotated viewpoint. The corner of the same cube should appear at **b**, but instead it still appears at **a** on the portal texture. The error can thus be measured as |**b**-**a**|. Figure 2 also shows the image as viewed through the portal. Notice how the geometry changes shape and occlusion. These changes do not properly occur with the sampled texture. The error measurement given above measures the deviation in geometrical shape, but does not properly account for changes in occlusion. In general, it is hard to define an objective metric to measure inaccuracy due to occlusion changes.

3.2 Best Fit Selection

An alternate way of measuring the error is to compute the deviation of the viewpoint from the texture sample point. Furthermore, by placing a bound on the velocity and acceleration at which the viewpoint can change, you can establish how long a portal texture will be valid until a certain error threshold is reached. Specifying a maximum distance from the texture sample point will bound the maximum error of the portal texture representation to a specific angular deviation [5]. When this threshold is reached, the portal texture can be resampled, another precomputed texture sample can be used or the geometry can be displayed.

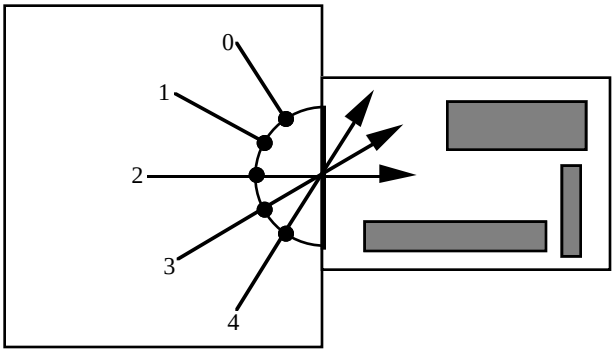


Figure 3: Portal Texture Sample Locations

The pfPortalTextures system stores a set of textures with each portal. Each texture corresponds to the view through the portal from a specific distance and viewing angle. More precisely, imagine placing a hemicircle in front of the portal. Each texture corresponds to a sample from a position on this hemicircle. According to what *pie-section* the viewpoint is in (or what portal texture sample angle is closest to the current incident angle to the portal texture), the corresponding portal texture is selected (Figure 3).

3.3 Lighting Error

If we use standard lighting models with the geometry and textures, the textures will be incorrectly illuminated. Each portal texture contains a sample of the shaded and illuminated geometry of the adjacent cell. The per-texel color information is preserved but the normals are lost (unless the texturing hardware supports per-texel normals). Thus, when rendering the portal texture, lighting should be turned off.

Furthermore, if the scene is rendered using a light moving relative to the model (i.e. the light follows the viewpoint), the illumination of the geometry in the texture will appear incorrect. Thus, it is more advantageous to have static lights (i.e. lights fixed relative to the model). Specular highlights will appear, though incorrectly for both view-centered and object-centered lights.

The overall lighting error can be removed by using precomputed diffuse illumination models (i.e. radiosity illumination models) or flat shading.

4. Geometry-to-Texture Transitions

In general texture-based simplification [4][5], the simplification process must also be concerned with performing smooth transitions when switching between geometry and texture (and vice versa) [6]. Furthermore, the boundary between the geometry and texture must maintain at least positional continuity (i.e. zero-order continuity). In the case of portal textures, the boundary between geometry and texture is not visible (since it is hidden by the *wall* surrounding each portal). Thus, we do not need to worry about the continuous border problem.

In order to achieve a smooth transition from the portal texture to geometry (for example, when the viewpoint gets too close to the texture and the system decides to restore the geometry) or vice versa, the geometry of the adjacent cell is gradually morphed. The morphing operation will morph the adjacent cell's geometry from its projected position on the portal texture plane to its actual position (or vice versa when morphing to a portal texture). Note that not all portals must have a portal texture, so sometimes the morphing operation is not straightforward (example in next paragraph). The morphing operation occurs over the next few frames after the start of the transition operation. The number of frames the morph operation covers is specified as a parameter. During a morph operation, the visual error is never greater than the error when just the portal texture is displayed.

A typical portal sequence is: portal A to adjacent cell A to portal texture B. When the near portal (portal A) is at the distance where it should switch to a portal texture, adjacent cell A and portal texture B will be morphed onto the plane of portal A. If a longer portal sequence occurs (for example, an intermediate portal does not have a portal texture representation), the farther cells are also morphed according to their portal state (either texture or geometry). Consider the portal sequence: portal A to adjacent cell A to portal B to adjacent cell B to portal texture C. The adjacent cell A, adjacent cell B and portal texture C will all be morphed onto the plane of portal A.

5. Implementation

5.1 pfPortalTextures in the perfPort Application

The runtime executable system is called *perfPort*. It combines pfPortals and pfPortalTextures with the standard IRIS Performer perfly application. The following sections will briefly highlight how portal textures are managed in the perfPort application.

Specifying Portal Textures

Each portal has a set of textures associated with it. A texture corresponds to the view of the geometry behind the portal (i.e. the adjacent cells) from a specific viewpoint and view direction. The range of view directions and sampling factor for each portal is specified by a portal texture parameter file. The input file defines the desired size of the texture (thus implicitly also the distance from the texture sample point to the portal itself), the rotation range (about an axis also specified) and the rotation step factor. Portals not specified in the file will not have a texture representation.

Recording Portal Textures

The system will automatically cycle through all portals cited in the portal texture parameter file and record the corresponding textures. Each portal texture sample is obtained by displacing the eyepoint along

the portal normal from the center of each portal (a typical height for a person walking through each portal). The amount of the displacement is computed so as to produce a texture of the specified size for that portal.

Using Portal Textures

Each portal will be automatically replaced with a portal texture when the distance to the portal is greater than the sampling distance of its portal textures (the closest portal texture sample according to the view position and view direction is automatically selected). The distance at which the transition occurs is implicitly defined by the size of the portal textures. A larger portal texture will allow the viewpoint to be closer before the transition occurs. In order to get apparently larger textures without the memory cost, the texture hardware's bilinear interpolation can be used to magnify the texture (note: the first time a portal texture is used, all the texture samples of that portal are loaded into texture memory, causing a small glitch or flicker to appear).

Geometric Morphing of Adjacent Cells

In order to minimize the jump produced when switching from portal textures to cell geometry or vice versa, morphing of the geometry of the adjacent cells can be enabled. The number of frames the morph operation covers is specified for each portal in the texture parameter file. The morphing operation uses the multiple processors (if available). During the first morph of a portal, intermediate data is computed and cached. Subsequent morphs will be faster.

5.2 pfPortalTextures API

The pfPortalTextures extension uses the pfPortals library as well as a slightly modified version of the fltLoader library (all are included in a ftp'able package). The below functions are the API available for a perfPort-level application program.

- **`pfInitPortalTextures(void)`**

Initialize the pfPortalTextures extension. This should be called immediately before the pfInitPortals() call. Both are called after pfConfig() but before loading any scene geometry.

- **`pfEnablePortalTextures(void)`**
- **`pfDisablePortalTextures(void)`**
- **`pfQueryPortalTextures(void)`**

These functions are used to enable/disable or query the current state of the pfPortalTextures extension.

- **`pfStartPortalTexturesRecord(char *parmfile, char *fltfile )`**
- **`pfEndPortalTexturesRecord(void)`**

Start recording portal textures. The texture parameter filename can be explicitly specified by the *parmfile* parameter or the PFPORTALPARMS environment variable or it will be deduced from the *fltfile* parameter. The texture parameter file is read and parsed. Any previous portal texture data is freed and overwritten. This function will enable pfPortals (if not already enabled). This function will start automatically cycling through all portals for which textures should be created.

- **`pfReadPortalTextures(char *texfile, char *fltfile)`**
- **`pfWritePortalTextures(char *texfile, char *fltfile)`**

These functions should be used to read portal textures from a file or write the current portal textures to a file. If the portal for a portal texture read from the file does not exist in the current scene graph, an appropriate message is displayed and that portal is ignored. The naming convention is similar to that used for the texture parameter file. The texture filename can be explicitly specified by the *texfile* parameter or the PFPORTALTEXTURES or PFPORTALPARMS environment variable or it will be deduced from the *fltfile* parameter.

- **`pfEnableMorphCelltoTexture(void)`**
- **`pfDisableMorphCelltoTexture(void)`**

Enable or disable morphing of the cell behind each portal when switching from texture to geometry or vice versa.

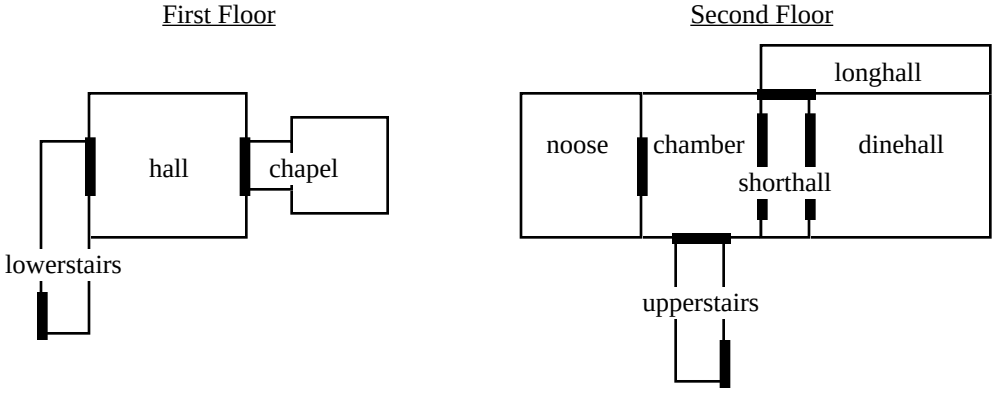
Mainloop

The functions pfQueryRecordingPortalTextures(), pfLoadPortalTextureMatrices(), pfRestorePortalTextureMatrices(), pfFirstPortalTexture(), pfNextPortalTexture(), and pfSavePortalTexture() are used in conjunction with pfStartPortalTexturesRecord() by the perfPort application mainloop. Below is an outline of this loop:

```
DrawFunc(pfChannel *chan, void *data)
{
   PreDraw(chan, data); /* e.g. - clear the viewport */
   if (pfQueryRecordingPortalTextures())
   {
         pfLoadPortalTextureMatrices(chan);
   }
   pfDraw(); /* Render the frame */
   if (pfQueryRecordingPortalTextures())
   {
         pfSavePortalTexture(chan);
         if (!pfNextPortalTexture())
            pfEndPortalTexturesRecord();
         pfRestorePortalTextureMatrices(chan);
   }
   PostDraw(chan, data); /* draw HUD, read GL devices */
}
```

5.3 Sample Model

The sample model of a two-story house (called *smallHaunted*) has been provided in the ftp'able package. This model is to be used with the perfPort application to visualize a sample usage of pfPortals and pfPortalTextures. All relevant datafiles are in the perfPort gzipped file.



The smallHaunted model consists of nine rooms (or cells) contained in seven .flt files. The radiosity-illuminated version is: hall.flt, chapel.flt, stairs.flt (upperstairs and lowerstairs cells), chamber.flt, noose.flt, winhall.flt (shorthall and longhall cells) and dinehall.flt. The standard shaded model is merged into a single file, smallHaunted.flt. To use this model, a typical invocation would be:

```
   % perfport -A -W 512,384 smallHaunted.flt
   or
   % setenv PFPORTALPARMS smallHaunted.parms
   % perfport -A -W 512,384 hall.flt chapel.flt stairs.flt chamber.flt
noose.flt winhall.flt dinehall.flt
```

Once perfPort has started, press “F1” to hide the GUI (in the current implementation, the GUI must be hidden in order for recording to properly work, otherwise the aspect ratio of the window is incorrectly computed), then press “R” to record portal textures (the smallHaunted.parms file is automatically used). Afterwards, portals will be enabled. Alternately, the portal textures can be read from a file. If so desired, the GUI can now be toggled on. Press “T”, “P”, “S” and optionally “m” to enable portal textures, portal bounds, small view frustum and optionally geometric morphing. Move through the house by using the Performer “fly” mode. Interesting locations are around the first floor (chapel and hall) and viewing the adjacent cells from the middle of the second floor (chamber). For example, place the viewpoint in the chamber or noose room and see the dinehall change between texture and geometry. Similarly, place the viewpoint in the dinehall and see the chamber, shorthall and noose room switch between geometry and texture.

The portals from the hall to chapel (and viceversa), chamber to noose room, shorthall to dinehall (and vice versa) have multiple portal texture samples. Other portals have one portal texture sample or no portal texture at all. The number of portal texture samples is determined by making reasonable assumptions about what typical viewing directions would be for a portal. To change the parameters, edit the smallHaunted.parms file.

The testportals.flt and testobs.flt models provided with pfPortals can also be used with pfPortalTextures. Sample testportals.parms and testobs.parms files are also in the ftp’able package.

6. Conclusions

We have outlined the benefits of using portal textures. This simplification approach allows you to reduce the rendering complexity to that of the view cell. The adjacent cell's are represented with an approximation. We have also outlined the errors introduced by the approximation. The errors can be bound thus we can always achieve a certain image quality.

We are still exploring other texture-based simplification approaches. Specifically, we believe we can incorporate incremental rendering ideas into portal texture-based simplification, whereby the accuracy of the textures are continuously refined whenever free frame time is available. The system can always guarantee to display a reasonable (perhaps precomputed) image.

References

[1] John Airey, John Rohlf, Fred Brooks, “Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments”, *Symposium on Interactive 3D Graphics*, pp. 41-50, 1990.

[2] Seth Teller, “Visibility Preprocessing For Interactive Walkthroughs”, *Computer Graphics (Proc. SIGGRAPH ‘90)*, Vol. 25, No. 4, pp. 61-69, 1991.

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

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

[5] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, John Synder, "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments", *Computer Graphics (Proc. SIGGRAPH ‘90)*, Vol. 25, No. 4, pp. 61-69, 1991.

[6] Daniel Aliaga, "Visualization of Complex Models Using Dynamic Texture-based Simplification", *IEEE Visualization ‘96*, pp. 101-106, 1996.