3D Image Warping in Architectural Walkthroughs

Matthew M. Rafferty, Daniel G. Aliaga, Anselmo A. Lastra

Department of Computer Science

University of North Carolina at Chapel Hill

{ *rafferty* | *aliaga* | *lastra* } *@cs.unc.edu*

ABSTRACT

We are investigating methods to accelerate rendering of architectural walkthroughs. In this paper, we improve upon a cells and portals framework by using image-based rendering techniques. We first store a few reference images of the view through each portal. At run time, we replace portals with these images warped to the current viewpoint. We begin with a well-known scheme for handling the complexity of a model, whereby the boundaries of enclosed spaces (cells) are used to divide the total space, and views of geometry beyond the currently occupied space are limited to the openings (portals) by walls. Our system improves upon the replacement of portals with conventional textures because the warping removes the popping effect when switching between image samples and significantly reduces the number of image samples needed.

Keywords: geometry, images, image-based rendering, plenoptic warping, cells, portals, interactive.

1. Introduction

High quality architectural walkthroughs require large and complex models with many geometric primitives. Recently, algorithms have been presented to take advantage of the structure of an architectural model by subdividing it into cells and portals [Airey90, Teller91, Luebke95]. These methods compute which cells (or rooms) are visible from the current location by finding the visible portals (windows, doors, etc.) to adjacent cells. We can use the information to cull parts of the model that are not visible.

As models increase in complexity, even this portal culling does not reduce the amount of rendered geometry enough to maintain interactivity. Our previous research explored the use of conventional textures at portals as replacements for the geometry of the view they represent [Aliaga97]. The rendering burden is substantially reduced by this method, since the portal textures can be sampled once and reused. The main problem is that a portal texture is only correct from a given viewpoint. If we want to maintain accuracy and provide motion parallax as the user moves, we have to use multiple portal textures. If we don't use enough portal textures to represent each portal, the user notices a “pop” from one texture to the next. The method of portal textures can require a large number of samples to reduce popping to an acceptable degree. This demands more texture memory (or the use of a large amount of main memory and copying to texture memory as necessary).

To combat this problem, we have applied image-based rendering techniques [Chen93, McMillan95c, Gortler96, Levoy96, Mark97] to warp the portal textures to the current viewpoint, thereby achieving smooth transitions. We are thus able to greatly reduce the number of images required per portal. There are problems with the warping algorithms such as *exposures*, which are gaps that appear in the warped image because unsampled areas of the scene should become visible. In this paper, we describe our solutions to these problems to produce an interactive system for architectural walkthroughs.

The following section presents an overview of portal culling and the use of portal textures to replace geometry. Section 3 describes portal warping and the problems encountered, along with our solutions. In section 4 we describe the implementation. Section 5 shows some results we have obtained using the system. Finally, section 6 describes some future work and section 7 presents some conclusions.

2. Portal Culling and Portal Textures

2.1 Portal Culling

Architectural models can be subdivided into cells and portals. Each cell contains a list of portals to adjacent cells. The cells and portals form a connectivity graph. At run-time, the system, starting with the cell containing the viewpoint (*view cell*), recursively traverses the connectivity graph by visiting all adjacent cells connected to the current cell by a visible portal.

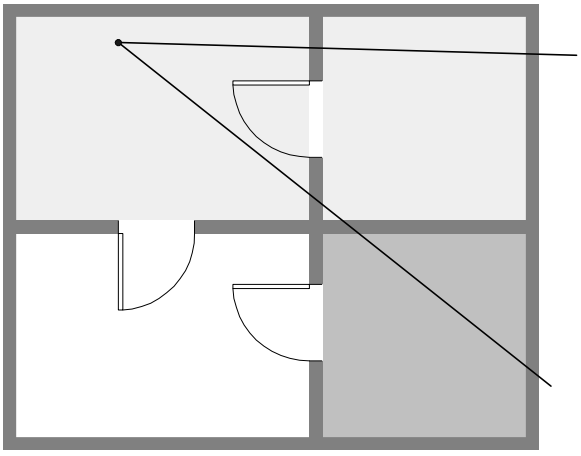


Figure 1. Portal Culling. The light gray cells are visible and must be rendered. The medium gray cell, although it lies in the view frustum, is not rendered because it did not appear during the recursive traversal.

2.2 Replacing Portals with Textures

Our previous system renders the view cell normally but renders all visible portals as textures. Substituting textures for geometry has the advantage that a texture can be rendered in time independent of the geometric complexity it represents, and texture mapping is often supported by graphics hardware. 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 texture.

The major problem with this approach is the number of portal textures required to adequately represent the adjacent cells. The simplest case is to only use a single texture, say sampled from directly in front of the portal. The problem is that as the user moves, the single texture appears more like a large painting hanging on the wall than like another room. To provide the desired 3D effect, we can use multiple textures [Aliaga97] and switch between them as we move. However, if we don't use enough of them, we see a very objectionable popping effect as the viewpoint moves and the system switches textures. In the accompanying video, we see that to get smooth transitions, we use a total of 120 textures sampled across a portal. On many machines that's just too much storage.

In order to smooth the transitions, we use image-based rendering techniques, specifically the one described in [McMillan95c], to warp one or more reference images to the current viewpoint. The next section describes the methods used to perform the warping.

3. Warping of Portal Images

3.1 Formulation of the Image Warping

We use the McMillan and Bishop warping equation (best described in [McMillan95b and McMillan97]) formulated as

$$x_2 = \delta(x_1)P_2^{-1}(c_1-c_2) + P_2^{-1} P_1 x_1$$

where

- x_1 is a set of coordinates for a reference image point,
- x_2 is a set of coordinates locating the corresponding point in the desired image,
- c_1 is the center of projection (COP) of the reference image,
- c_2 is the COP of the desired image,
- P_1 is the transformation matrix of the reference image,
- P_2^{-1} is inverse of the transformation matrix of the desired image,
- $\delta(x_1)$ is the *disparity* of the reference image pixel at x_1.

The disparity term is related to the classical stereo disparity measure, and is proportional to the distance from the COP of the reference image to a pixel, divided by the range to the surface represented by that pixel. Thus, the disparity is inversely proportional to distance and measures how far a pixel will flow as the viewpoint changes — closer objects will move farther. In section 4, we discuss how we compute disparity from the z buffer of the reference image.

Since the reference image is on a regular grid, many of these computations are incremental, thus fast. The amount of work is similar to that required by traditional texture mapping.

Note that the results of this warp are not one to one: multiple points in the reference image may be warped to a single point in the desired image. This raises the issue of visibility resolution: we must somehow ensure that when multiple pixels from the reference image warp to the same pixel in the desired image, the one representing the closest of the points to the current viewpoint is the one that “wins”. We could use z-buffering to resolve visibility, but in our case it's faster to use the back-to-front occlusion-compatible order described in [McMillan95a].

This algorithm is similar to a painter's algorithm. We first determine the projection of the COP of the desired image in the reference image. We use that point to divide the reference image into a number of *sheets.* There are four, two, or one, depending on whether both, one, or neither of the coordinates of the projected intersection lie in the image domain. We then determine whether we must warp the pixels in the sheets towards or away from the projected point, depending on whether the desired COP is in front of, or behind the reference COP.

Since the sheets can be warped and rendered independently with correct occlusion guaranteed, we can parallelize the implementation of the warp as described in section 4.

3.2 Reconstruction

In our laboratory, we've used two methods for resampling of the desired image, bilinearly interpolated surfaces and splatting [Westover91]. [McMillan97] includes a good discussion of the reconstruction issues involved in image warping. We decided that surface patches would be too expensive to evaluate, therefore we used a splat. Through a software switch, we can decide whether to compute an approximation to the projected size of each pixel (for a more accurate splat) or to use a fixed-size footprint. Since the fixed-size splat is cheaper to compute and provides a visually pleasing result, we usually use a three by three footprint in preference to the more accurate solution. We note that in [Gortler97] the authors also decided to use a fixed-size kernel.

3.3 Exposures

A typical image represents an environment from only a single viewpoint. Therefore, there is information about only a single surface at each pixel, the one nearest to the COP (ignoring clipping by a hither plane). As we move in three-dimensional space warping a single reference image, we see areas of the environment that were not sampled in the image (note the example in figure 2). We have no information about the exposed surfaces, so we don't know what to render. The effect of these *exposures* in warped images is illustrated in color figure A. If nothing is done to correct for the problem, the exposures appear as “tears” or sharp shadows in the images.

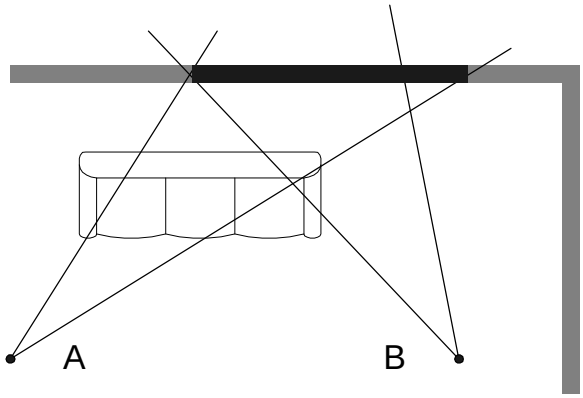


Figure 2. If the reference image is taken from viewpoint A, and we warp to point B, we have no information about the section of wall (shown in black) behind the sofa or about the side of the sofa. Since parts of these surfaces are visible from B, we must obtain this information from another reference image.

The simplest solution is to increase the number of reference images per portal. Thus when the viewpoint moves, the reference image being warped is close to the desired image and the widths of the tears are proportionally reduced. However, it is not practical to use this solution, because it exacerbates the problem we were trying to solve by warping, namely the large number of reference images.

A better solution is to warp multiple reference images, expecting that surfaces exposed in one reference image will have been sampled in another image. We implement this solution by warping the two nearest reference images. We don't decide explicitly which warped pixels are best. Rather, we warp the second nearest reference image first, then warp the closest reference image into the buffer over the first. In practice, this seems to provide an excellent solution. Color figure B shows the result of warping the two images nearest to the same viewpoint used to render color figure A. Notice that the exposed regions are now filled and the image looks quite good. For comparison, we provide an image in color figure C that is rendered from geometry. The only part that is wrong in the warped image is visible through the doorway on the left. Apparently there was some detail that was not visible from either of the two reference images. Presumably this was visible only from a narrow angle.

We could warp more reference images in order to try to reconstruct the environment in greater detail. However this is not practical because of the time involved and not particularly rewarding given the small amount of detail that's lost. Also, since transitions are smooth during movement, the small artifacts are not particularly noticeable. In practice, warping the two nearest reference images seems to give very acceptable quality and good performance.

One more solution that we have tried is to allow previous images to persist in the buffer as we warp new ones [McMillan, personal communications]. We accomplish this trivially, by leaving the warp buffer uncleared (which has the benefit of also reducing memory access). Although this is a “hack”, it tends to fill the tears with a plausible color, as long as the warped image is not too far from the reference images, and the movement is smooth. If we warp a single reference image too far or the viewpoint changes are large, the effect is rather hallucinatory. In practice, it works well.

4. Implementation

4.1 System

We implemented our system on a Silicon Graphics Onyx (250 MHz R4400, 2GB main memory) with Infinite Reality graphics (containing 64MB of texture memory) and on an Indigo2 (250 MHz R4400, 128MB memory) with Max Impact graphics (and 4MB of texture memory). The system is coded in C++, and uses the OpenGL graphics library.

At run-time, our visibility algorithm determines which portals are visible. Then, we make sure the reference images for warping are created. We chose a nominal image size of 256x256 pixels. All visible portal images are warped into a common warp buffer of the same aspect ratio as the main window. Then, the warp buffer is copied to the frame buffer; finally all visible geometry is rendered on top (leaving holes at the location of the portals through which we see the warped images).

For our application, the COP of the desired image typically projects onto the reference image, producing four roughly equal-sized sheets. We can take advantage of this to parallelize the warp. We accomplished this by using *fork-join* multiprocessing directives. Our system employs up to four processors to warp the sheets in parallel.

The top-level visibility algorithm is described below:

```
Visibility(cell, frustum) {
   Mark cell visible
   Cull cell to frustum
   Foreach portal {
      Cull portal to frustum
      if (portal is visible) {
         if (portal is image) {
            Choose best reference image(s)
            Warp reference image(s)
         } else
            Visibility(portal’s adjacent cell,
                       culled frustum)
      }
   }
}
```

4.2 Reference Images

A portal image, its dimensions, and its camera model parameters are grouped into a single data structure. The image’s depth buffer is also retained, but the depth value for each pixel must be converted to a disparity value by the formula

```
disparity(u,v) = 1 - z(u,v) * (f - n) / f
```

where

- z(u,v) is the OpenGL z-buffer value for a pixel at u, v,
- f is the distance from the reference viewpoint to the far clipping plane, and
- n is the distance from the reference viewpoint to the near clipping plane.

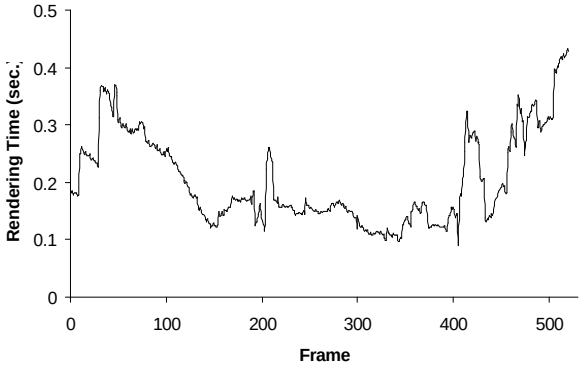


Figure 3. Rendering time of a path through the Brooks House model using only portal culling.

We need to define a set of reference images for the warping algorithm. As in [Aliaga97], our reference images are sampled along a semicircle in front of each portal located at the typical viewing height. The semicircle does not need to cover the full halfspace in front of a portal, but only the span from which the portal will be seen. In our examples, we typically generate a reference image every 10 or 20 degrees in front of the portal over an angular range of 60 or 120 degrees.

5. Results

We tested our system with two architectural models. The first model, named *Brooks House*, is that of a one-story radiosity-illuminated house. The house has 19 cells, 52 portals and 528,000 polygons. The second model, named *Haunted House*, is a smaller two-story house with 214,000 polygons, 7 cells and 12 portals.

We recorded a 520-frame path through Brooks House. We ran the path twice, warping one and two reference images per portal. For this path, we used 256x256 reference images and a 640x512 common warp buffer. When warping one image per portal, we obtained speedups of up to 4.9 over portal culling alone (overall speedup of 1.87). Figure 3 shows the results of portal culling alone. Figure 4 shows the frame times when image warping is enabled. If we time the same path, but warp two images per portal, our maximum speedup is 3.17 (overall 1.25). Note that an advantage to the use of portal images is that the portal warping time is independent of the amount of geometry. Thus, we should see higher speedups for more complex models.

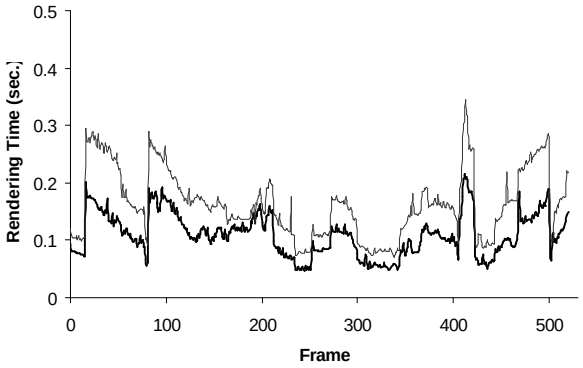


Figure 4. Rendering time of the same path but warping one and two reference images. Note the improvement in performance and the reduced variability in frame time.

Since we require few reference images per portal, we precompute the images and store them in host memory. Thus, the total number of reference images stored per portal does not affect performance — only the number of reference images actually warped per frame affects performance. We have experimented with computing the images on the fly. We obtain the same speedups except for sporadic spikes whenever a reference image is created. We feel that the amount of storage needed to store the reference images is not unreasonable (for example, in the Brooks House model there are 52 portals for a total of 312 images).

We also timed how fast our algorithm can warp and draw a reference image. On our workstation, we can warp a single 256 by 256 reference image into the 640 by 512 buffer in 0.036 seconds (including the time to copy to the frame buffer). A 512 by 512 reference image can be warped in 0.101 seconds.

6. Future Work

The work that might yield the most immediate benefit is an investigation into the placement of reference images. We chose the semicircle in front of the portal only because it was the same pattern that we used in our previous system. Perhaps a regularly spaced grid of sample points might work best, especially if we were to work on models with larger rooms.

For the same case of larger spaces, we might have to look more closely at reconstruction. The farther we get from reference images, the more important it becomes do a better job of resampling.

We have considered the use of inverse warping [McMillan97] as a second pass in order to render the portal image. We could forward warp from the best reference image, then inverse warp to fill in any remaining samples. Similarly, it may be fruitful to investigate representations of reference images with multiple depths per sample [Max95, Gortler97].

Our models use only diffuse lighting (with a precomputed radiosity solution). High specularity, in particular, might force us to use more reference images. We could investigate deferred shading of the warped samples, although the cost might be prohibitive.

Finally, we are investigating specialized hardware to compute the warped image more rapidly. This would allow us to not only increase our frame rate, but would also enable us to generate more detailed portal images.

7. Conclusions

We have presented a system for interactive rendering of architectural walkthroughs that uses image-based techniques to increase performance while maintaining high

quality and smooth motion. To our knowledge, this is the first practical application of warping techniques, such as that of McMillan and Bishop, that take depth at each pixel into consideration.

We employ multiple processors to accelerate the 3D-image warping and use a simple reconstruction kernel. We also investigated how to reduce the exposure events that occur in portal image warping when previously occluded regions suddenly become visible. Furthermore, we are able to transition smoothly between portal images using an order of magnitude fewer of them than with the portal textures method alone. Since so few reference images are required, it's feasible to precompute them as we do in our production walkthrough system.

8. Acknowledgments

We would like to thank Leonard McMillan for the warping code from which we began our development, Wolfgang Stuerzlinger for code and advice, and Bill Mark for insight into image-based rendering.

The Brooks House model is courtesy of many generations of students and members of the UNC Walkthrough team. Dave Luebke and Mike Goslin created the Haunted House model.

This research was supported in part by grant number RR02170 from the National Institutes of Health National Center for Research Resources, the Defense Advanced Research Projects Agency under order number E278, and grant number MIP-9612643 from the National Science Foundation.

References

[Airey90] John Airey, *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision*, Ph.D. Dissertation, University of North Carolina (also UNC Computer Science Technical Report TR90-027), 1990.

[Aliaga97] Daniel G. Aliaga and Anselmo Lastra, "Architectural Walkthroughs using Portal Textures", *to appear in Proceedings of IEEE Visualization '97*.

[Chen93] Shenchang Eric Chen and Lance Williams, "View Interpolation for Image Synthesis", *SIGGRAPH 93*, 279-288, 1993.

[Gortler96] Steven Gortler, Radek Grzeszczuk, Richard Szeliski and Michael Cohen, "The Lumigraph", *SIGGRAPH 96*, 43-54, August 4-9, 1996.

[Gortler97] Steven Gortler, Li-we He, and Michael Cohen, "Rendering from Layered Depth Images", Microsoft Research Technical Report MSTR-TR-97-09, March 19, 1997.

[Levoy96] Marc Levoy and Pat Hanrahan, "Light Field Rendering", *SIGGRAPH 96*, 31-42, August 4-9, 1996.

[Luebke95] David Luebke and Chris Georges, "Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets", *Proc. Symp. on Interactive 3D Graphics*, 105-106, 1995.

[Max95] Nelson Max and Keiichi Ohsaki, "Rendering Trees from Precomputed Z-Buffer Views", *Proc. of the 6th Eurographics Workshop on Rendering*, June 1995.

[McMillan95a] Leonard McMillan and Gary Bishop, "Head-Tracked Stereo Display Using Image Warping", *Stereoscopic Displays and Virtual Reality Systems II*, Scott S. Fisher, John O. Merritt, Mark T. Bolas, ed., SPIE Proceedings 2409, (San Jose, CA), Feb 5-10, 1995, 21-30.

[McMillan95b] Leonard McMillan and Gary Bishop, "Shape as a Perturbation to Projective Mapping", UNC Computer Science Technical Report TR95-046, University of North Carolina, April 1995.

[McMillan95c] Leonard McMillan and Gary Bishop, "Plenoptic Modeling: An Image-Based Rendering System", *SIGGRAPH 95*, 39-46, August 1995.

[McMillan97] Leonard McMillan, *An Image-Based Approach to Three-Dimensional Computer Graphics*, Ph.D. Dissertation, University of North Carolina (also UNC Computer Science Technical Report TR97-013), April 1997.

[Mark97] William R. Mark, Leonard McMillan and Gary Bishop, "Post-Rendering 3D Warping", *Proc. Symp. on Interactive 3D Graphics*, 7-16, 1997.

[Teller91] Seth Teller and Carlo H. Séquin, "Visibility Preprocessing For Interactive Walkthroughs", *SIGGRAPH 91*, 61-69, 1991.

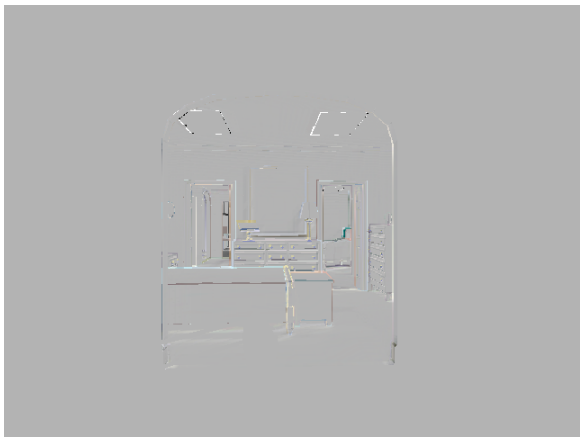
[Westover91] Lee Westover, *Splatting: A Feed-Forward Volume Rendering Algorithm*, Ph.D. Dissertation, University of North Carolina, 1991.

Note: some of the UNC references are available in electronic form on http://www.cs.unc.edu/~ibr/pubs.html.

Color Figure A: Exposure Events. This image shows a single reference image being warped (from a total of six sampled across the portal). The viewpoint is at the worst location for this reference image. Observe the black areas where we have no visual information.

Color Figure B: Two Reference Images. An image from the same viewpoint as A, but we are warping the two nearest reference images (from a total of six) to render the desired image. Most of the areas that were invisible from one are visible from the second.

Color Figure C: Geometry. An image from the same viewpoint as A, but rendered using the model geometry for purposes of comparison with figures A and B. The main difference is some detail through the left doorway. Apparently these were some features that were not visible from either reference image.



Color Figure D: Difference Image. This image is a signed difference image between Color Figure B and C. Gray equals zero difference.

Color Figure E: Haunted House. A screen shot from the Haunted House model. In this example, we are warping one image from a total of six reference images.