Appeared: Computer Graphics & Applications, November/December, 1998.

Images for Accelerating Architectural Walkthroughs

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

Department of Computer Science

University of North Carolina at Chapel Hill

{ *rafferty* | *aliaga* | *popescu* | *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 images to replace geometry visible through portals (doors and windows) in a three-dimensional model. We present several solutions: images texture-mapped onto the portal, 3D warping of single reference images, and 3D warping of layered depth images. We have achieved significant speedups and present the results of applying the techniques to large architectural models.

Keywords: interactive, geometry, images, cells, portals, image-based rendering, 3D image warping, layered depth images.

1. Introduction

High quality architectural walkthroughs require large and complex models with many geometric primitives. Algorithms have been presented to take advantage of the structure of an architectural model by subdividing it into cells and portals [1][2]. 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. We have explored the use of images, in several forms, at the portals as replacements for the geometry seen through the portal.

A simple approach we have explored is to use one or more images as a conventional texture [3]. The rendering burden is substantially reduced by this method, since the portal textures can be created once and reused. A problem is that a portal texture is only correct from a given viewpoint. If we want to maintain high visual fidelity and provide motion parallax as the user moves, we have to use multiple textures per portal. If we don't use enough portal textures to represent each portal, the user notices a "pop" when we switch from one texture to the next. The method of portal textures can require a large number of texture 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 [4][5] 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. Unfortunately, there are problems with the warping algorithms. For example, gaps that appear in the warped image because areas of the scene that should have become visible were not sampled in the reference image.

We implemented two solutions. The first is to warp multiple images made from different viewpoints. However, this not only leads to redundant work, but also would require *z*-buffering to resolve visibility. The second, more comprehensive solution, uses layered depth images (LDIs) [6][7] to store, on one image, multiple samples that become visible as the image is warped.

Our overall approach is a specialization of the use of *impostors* introduced by [8]. Portions of a general 3D model can be replaced with representations that are faster to render, namely 2D textures. Solving the general problem of deciding where to place textures in order to improve rendering performance is difficult [9,10,11,12]. The cells and portals framework allows us to formulate a set of concrete and efficient algorithms for replacing geometry with images.

In this paper, we describe our image-based solutions to produce a system for the interactive walkthrough of large architectural models. The following section presents an overview of portal culling and the use of portal textures to replace geometry. Section 3 describes 3D image warping applied to portals and the problems encountered. In section 4, we describe layered depth images. Section 5 describes the implementation and section 6 shows some results we have obtained using the system. Finally, section 7 summarizes future work and section 8 presents conclusions.

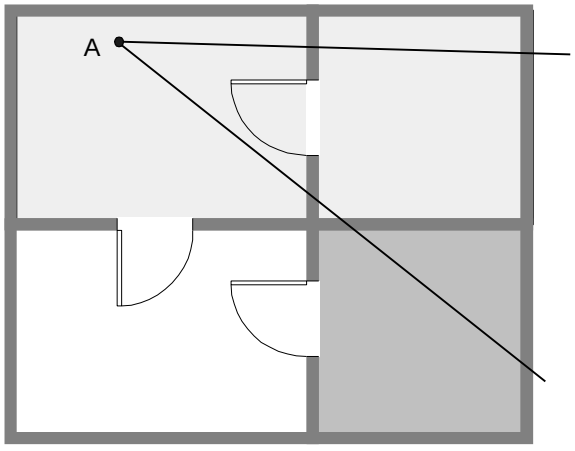

Figure 1. *Portal Culling. From viewpoint A, the light gray cells are visible and must be rendered. The medium gray cell, although it lies in the view frustum, is not rendered because it did not appear during the recursive traversal.*

2. 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. Starting with the cell containing the viewpoint (*view cell*), the system recursively traverses the connectivity graph at runtime by visiting all adjacent cells connected to the current cell by a visible portal (Figure 1).

2.2 Replacing Portals with Textures

Our first system renders the view cell normally but renders all visible portals as textures. Substituting textures for geometry has the advantages that a texture can be rendered in time independent of the geometric complexity it represents, and that 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. The adjacent cell becomes the view cell once the viewpoint enters it, and the previous cell will now be rendered as a texture.

A very low-cost approach is to use only one texture per portal, say sampled from directly in front of the portal. However, as the user moves, the single texture appears more like a large painting hanging on the wall than like another room.

To approximate the 3D effect, we can use multiple textures [3] and switch between them as we move. Since we are designing an architectural walkthrough system, we assume the eye height remains approximately constant and create the textures from viewpoints along a semicircle in front of each portal (Figure 2). Unfortunately, if we don't use enough of them, we see a very objectionable *popping* effect as the viewpoint moves and the system switches textures. We found that to get smooth transitions, we needed to use approximately one texture per degree. On many machines that's just too much storage.

In order to smooth the transitions and reduce the number of images per portal, we use 3D image warping [5] to warp reference image(s) to the current viewpoint. The next section describes the methods used to perform the warping operation.

3. Warping of Portal Images

3.1 Formulation of the Image Warping

We use the McMillan and Bishop warping equation [5]. This equation is a 3D transformation reformulated to take advantage of image-space coherence to reduce computational costs. The equation is

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

where

- x_1 is a set of coordinates for a reference image point,

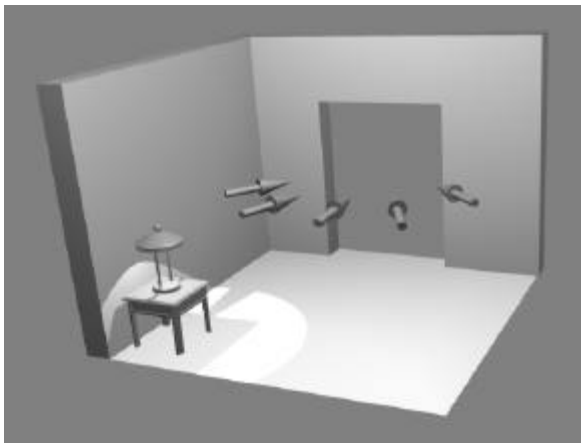

Figure 2. *Constrained portal texture viewpoints lying on a semicircle in front of the portal at eye height.*

- x_2 is the set of coordinates in the desired image to which x_1 warps,
- 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. We compute the disparity values from the OpenGL *z-buffer* of a reference image as follows:

```
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.

The amount of work for image warping is proportional to the number of pixels in the image and independent of scene complexity. Furthermore, since the reference image is on a regular grid, many of these computations are incremental and fast. The 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 [5].

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. If the desired COP is behind the reference COP (i.e. farther away), the pixels must be warped towards the projected point. The amount by which a pixel is displaced is proportional to its disparity value. It is difficult to create a raster-scan ordering of the circular pattern of pixels moving towards or away from the projected point. Thus, dividing the reference image into a number of sheets generates an occlusion-compatible ordering of the pixels. There are four, two, or one, depending on whether both, one, or neither of the coordinates of the projected point lie in the image domain. The pixels of each sheet are warped 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 this warp as described in Section 5.

3.2 Reconstruction

In our laboratory, we've used two methods for resampling of the desired image: bilinearly interpolated surfaces and splatting [13]. 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 use either a two-by-two or a three-by-three footprint in preference to the more accurate solution.

3.3 Limitation of Single-Image Warping

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 3). We have no information about the exposed surfaces, so we don't know what to render. The effect of these missing samples in warped images is illustrated in color figure A. If nothing is done to correct for the problem, the lack of information appears as "tears" or sharp shadows in the images.

A partial solution, not correct but visually pleasing, is to allow samples from previous images to persist as we warp new ones [McMillan, personal communications]. We accomplish this trivially by warping on top of the previous image. 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.

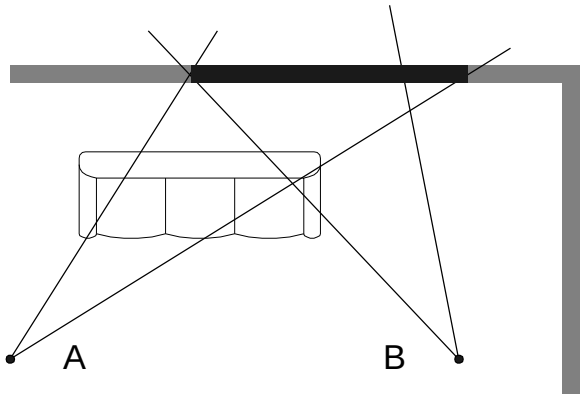

Figure 3. *If the reference image is taken from viewpoint A, and we warp to point B, we have no information about the section of wall (shown in black) behind the sofa or about the side of the sofa. Since parts of these surfaces are visible from B, we must obtain this information from another reference image.*

Another 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 not visible in one reference image will have been sampled in another image. We implemented this solution by warping the two nearest reference images.

When using two images, we encounter one of three visibility cases for each pixel: when both reference images contain information about the same geometry, when one image contains information absent from the other image, and when information about some geometry is missing from both images. The first and third cases are straightforward (although in the third case, we will be missing some information). The second case becomes a problem if the last warped image is the one that is missing data from a certain location, and it contains information from a different depth that obscures correct detail by warping to the same location. This problem manifests itself as a flashing effect, as correct detail appears and disappears under incorrectly overlaid imagery. A solution would require a decision process to choose the best source for each pixel in the composite image. This is a very difficult problem without computationally expensive *z*-buffering.

In our first warping system [14], we did not decide explicitly which warped pixels were best. Rather, we warped the second nearest reference image first, then warped the closest reference image over the first. 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 D 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.

Warping a second image just to display a few samples (e.g. 10% of the pixels) that were not visible in the first is very wasteful. Ideally, we could identify the necessary samples and warp only those. Our next system, which used layered depth images did just that.

4. Layered Depth Images

Layered Depth Images (or LDIs) [6][7] are the best solution to date for the visibility errors to which 3D warping is prone. They are a generalization of images with depth since, like regular images, they have only one set of view parameters but, unlike regular images, they can store more than one sample per pixel. The additional samples at a pixel belong to surfaces, which are initially invisible, along the same ray from the viewpoint.

Figure 4 shows construction of an LDI using two methods. The first, using raycasting, just stores multiple samples hit by the rays of each pixel of the LDI. The second method consists of warping secondary images to the viewpoint of the primary image, then selecting samples that have not already been stored in the LDI (this is an approximate determination).

Whenever the LDI is warped to a view that reveals the (initially) hidden surfaces, the deeper layer samples will not be overwritten anymore and they will naturally fill in the gaps that would otherwise appear. As will become clear in the next subsection, most of the work is done during preprocessing when the LDIs are constructed.

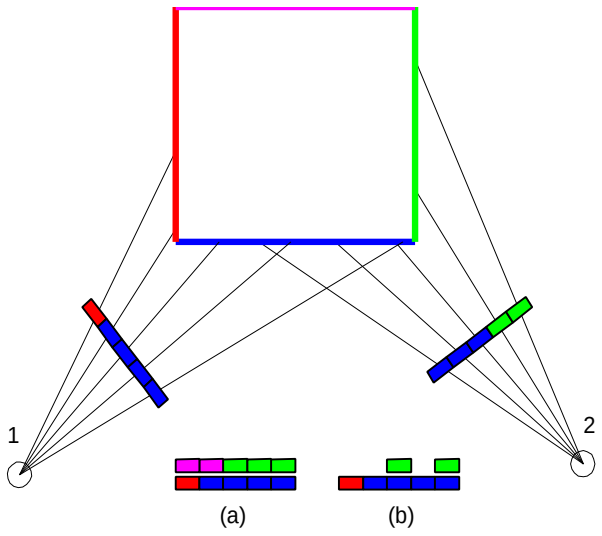

Figure 4. *Construction of a Layered Depth Image. The LDI is constructed from viewpoint 1. Each sample of the LDI stores not only color, but also disparity (depth). (a) shows a LDI built by raycasting. The rays sample surfaces beyond those that would be seen from viewpoint 1. (b) shows a LDI constructed by warping a second image from viewpoint 2 to viewpoint 1. Redundant samples are discarded, others are stored at deeper layers. Note that the undersampling of the green surface from viewpoint 1 is not a problem because it's never seen from that direction.*

Layer	Samples	% of total
1	120556	72.07
2	32458	19.40
3	9714	5.80
4+	4567	2.73
Total	167295	100

Figure 5. Samples in layers of the LDI shown in color Figure F.

An additional benefit that the single set of camera parameters brings is that, as demonstrated by [7], LDIs can be warped in McMillan's occlusion compatible order [5]. The next subsections describe the construction and warping of LDIs.

4.1. Construction of LDIs

In our system, we construct LDIs by taking advantage of the architectural domain. We use images that are evenly distributed on a semicircle in front of each portal to create its LDI. Our main concern is to collect samples from all surfaces that are visible in at least one view of the portal.

We take advantage of conventional rendering hardware to create the images. There are several reasons for choosing this approach:

- the samples in the LDI are all potentially visible (keeping the depth complexity of the LDI low),
- the samples are taken at the proper resolution (appropriate level of detail), and
- the construction of the LDI takes a reasonable amount of time (of special interest when LDIs are constructed on the fly).

Of course there is no guarantee that all potentially visible surfaces are captured in the LDI. However the more images used to construct the LDI, the lower the probability of missing a sample. In our implementation we used thirteen images per portal and the exposure errors were virtually eliminated (Color figure C). The basic LDI construction algorithm is:

```
1. Create empty LDI with view parameters of
   the central construction image
2. For every subsequent construction image
      for each pixel (sample)
         warp to the LDI
         inspect all samples of the LDI
         location
            if there is a sample at similar
            depth, resolve conflict
            else install sample in a new
            layer at that location in
            correct depth order
```

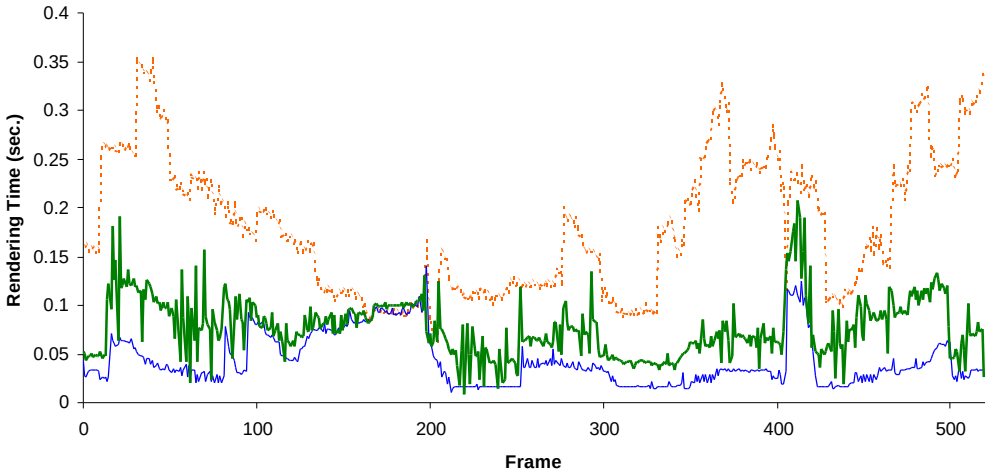

Figure 6. *Rendering time of a path showing portal culling (upper plot, red dashed-line), LDI warping (middle, green line), and a portal texture (lower, blue line).*

The reason we decide at step one to give the LDI the camera parameters of the central image is that it is the most important view of the portal. Indeed, that is the view often used during the walkthrough when one wants to explore the adjacent room. The warping errors are smaller when the desired image has the view parameters close to the view parameters of the reference image.

Step two is the most laborious and one can see that the total work is proportional to warping all of the images. If new samples are encountered, they are stored in the LDI. A sample is new if it warps to an empty LDI location or if it is far enough away in depth from the other samples at the same location. We establish the threshold below which a sample is considered to be at the same depth by a trial and error method, once per model.

The most delicate part of the algorithm is resolving the conflicts when two samples collide in the LDI. Eliminating one of the two avoids the redundant work that was the major drawback when two or more images were warped. However, eliminating too many will lead to undersampling problems when the portal is viewed at acute angles. We decide which one to eliminate by estimating the area of the 3D scene surface patch seen by the two samples in their images of origin. The sample with the smaller area is preferred. One exception is that if two samples come from the same construction image, both are kept. This prevents undersampling of surfaces that are nearly parallel to the rays of the LDI. Also, the samples coming from the central image are in the first layer of the LDI and cannot be pushed back or eliminated by any other samples; this ensures a high quality image when the desired view is similar to the central image (a common case for this application). The resulting LDI is similar to a wide field-of-view image of the portal with the exception that some locations store more than one sample. Color Figure F shows the first 3 layers of an LDI. Figure 5 shows the number of samples at each depth. The construction images were 256 by 256 pixels in size.

4.2 Warping the LDI

The main concern in warping the LDI is efficiency. During construction, we worked to include the best available samples of the surfaces seen through the portal. From some views, some areas are actually oversampled, causing us to warp more samples than strictly necessary. However, better sampling ensures that there will be enough samples for every visible surface from all views. It also enables us to use a small, constant-sized stamp for reconstruction via splatting. We can use a stamp as small as 5 pixels: Center, East, South, West and North.

A more direct way of improving the efficiency of the warping is to eliminate some of the samples of the LDI that will clearly not warp to the desired view of the portal. We use a fast recursive-clipping algorithm that conservatively eliminates the columns from the left and right side of the LDI based on their minimum and maximum disparities [15]. On our test paths, the clipping algorithm reduced the number of samples to be warped by 50%, on average.

5. Implementation

We implemented our system on Silicon Graphics Onyx and Onyx^2 computers with Infinite Reality graphics and on an Indigo2 with Max Impact graphics. 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).

The COP of the desired image is projected onto the reference image to determine the visibility preserving ordering of the reference image pixels. In our first attempt

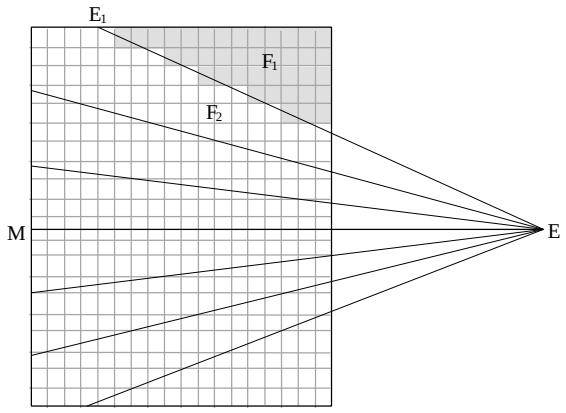

Figure 7. *In this case, the projection E of the COP falls outside the reference image (or the LDI). The horizontal line EM divides the image in two sheets. Using 3 lines, each of the sheets is divided in 4 fragments (one for each available processor) of equal area. The pixels lightly shaded form an example fragment F_1 to be handed off to a single processor.*

to parallelize the warp, the resulting one, two or four sheets were assigned to different processors but the load balancing was often poor. We solved the problem by splitting each sheet into *p* equal-area fragments where *p* is the number of available processors [15]. The sheets are split along lines emanating from the projected COP (as in Figure 7).

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) {
            Warp reference image(s)
         } else
            Visibility(adjacent cell,
                       culled frustum)
      }
   }
}
```

6. Results

We tested our system with two architectural models. The first model, named *Brooks House*, is of a one-story radiosity-illuminated house. The house has 19 cells, 52 portals and 1,744,000 polygons. The second model, named *Haunted House*, is of a smaller two-story house with 214,000 polygons, 7 cells and 12 portals.

We recorded a 520-frame path through Brooks House (Figure 6). We ran the path using several methods: portal culling only, a single conventional texture, warping one reference image, and warping one LDI per portal. For this path, we used 256x256 reference images and a 640x512 common warp buffer. Note that an advantage to the use of portal images is that the portal warping time is independent of the amount of geometry beyond the portal.

When the portals were rendered by warping LDIs, using the improved parallelization scheme, the clipping algorithm and the smaller reconstruction stamps, the frame rendering times were not much greater than the ones obtained when only one image was warped (and always less than warping two images). On our workstation, we can warp a single 256 by 256 reference image into a 640 by 512 frame buffer in approximately 0.03 seconds. Warp of a LDI can range from 0.03 to 0.09 seconds. All of these timings include the copy to the frame buffer.

The rendering load is typically reduced to that of the current cell; that and the size of the images to be warped determine the total rendering time. Thus, we should see higher speedups for more complex models.

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 (not LDIs) 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 only 52 portals).

7. Future Work

The work that might yield the most immediate benefit is an investigation into the placement of reference images. 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 image COPs, the more important it becomes to do a better job of resampling. However, we are severely limited by the amount of computational power available.

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.

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.

8. Conclusions

We have presented several systems for interactive rendering of architectural walkthroughs that use image-based techniques to increase performance while maintaining high quality and smooth motion. For highest quality, we prefer the LDIs. However, conventional textures exhibit the highest performance and can be generated at run time.

To our knowledge, this is the first demonstrated application using the McMillan and Bishop Plenoptic Warping technique, which takes depth at each pixel into consideration. We employ multiple processors to accelerate the 3D-image warping and use a simple reconstruction kernel. We employ LDIs to reduce the errors that occur in portal image warping when previously occluded regions suddenly become visible. Since so few reference images are required, it's feasible to precompute them as we do in our production walkthrough system.

9. 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

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

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

[3] Daniel G. Aliaga and Anselmo Lastra, "Architectural Walkthroughs using Portal Textures", *IEEE Visualization '97*, 355-362, 1997.

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

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

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

[7] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski, "Layered Depth Images", *SIGGRAPH '98*, 231-242, July 1998.

[8] Paulo Maciel and Peter Shirley, "Visual Navigation of Large Environments Using Textured Clusters", *Symp. on Interactive 3D Graphics '95*, pp. 95-102, 1995.

[9] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose and John Snyder, "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments", *SIGGRAPH '96*, 75-82, 1996.

[10] Gernot Schaufler and Wolfgang Sturtzlinger, "A Three Dimensional Image Cache for Virtual Reality", *Eurographics '96*, 227-235, 1996.

[11] Daniel G. Aliaga. "Visualization of Complex Models Using Dynamic Texture-Based Simplification", *IEEE Visualization '96*, 101-106, 1996.

[12] Peter Ebbesmeyer, "Textured Virtual Walls - Achieving Interactive Frame Rates During Walkthroughs of Complex Indoor Environments", *VRAIS '98*, 220-227, March, 1998.

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

[14] Matthew Rafferty, Daniel G. Aliaga, and Anselmo Lastra, "3D Image Warping in Architectural Walkthroughs", *VRAIS '98*, 228-233, March, 1998.

[15] Voicu Popescu, Anselmo Lastra, Daniel G. Aliaga, and Manuel de Oliveira Neto, "Efficient Warping for Architectural Walkthroughs using Layered Depth Images", *IEEE Visualization '98*, to appear, 1998.

Color Figure A: Visibility Errors. 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 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. The large areas that were invisible from one are visible from the second.

Color Figure C: Layered Depth Image. The LDI for the portal captures almost all of the visible detail. In addition, it takes up less storage and can be rendered faster than two full reference images. Furthermore, in the architectural domain, we can construct high-quality LDIs by using reference images sampled along a semicircle in front of each portal.

Color Figure D: Geometry. An image from the same viewpoint as A, but rendered using the model geometry for purposes of comparison with figures A, B and C. The main difference is some detail through the left doorway. Apparently these were some features that were visible from only certain viewing angles.

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.

Color Figure F: Layers of a LDI. The images show, from top to bottom, the samples at depth 0, 1 and 2 of a LDI. The side walls are present in more than one layer not because of visibility, but rather because they are not adequately sampled by the central view; when the LDI is warped to a view where better sampling is necessary, the successive layers will unfold, ensuring better reconstruction.