Multiple Image Warping for Remote Display of Rendered Images

Thomas C. Hudson and William R. Mark

Abstract:
We address the problem of rendering a scene in one location and displaying it in another, remote, location, where the remote user controls the viewpoint. Network latency makes it impossible to accept a new viewpoint, render the complete scene, and return the generated image to the user fast enough to provide immersion. Our render server constructs images and depth maps ("depth images") and sends them over the network to a remote client. The client warps and composites several depth images to form the image displayed to the user. Using multiple images greatly reduces the visual artifacts produced by image warping. The client runs entirely in software and displays images at 6 frames per second.

1 Introduction

Systems that provide remote viewing of three–dimensional data with interactive viewpoint control must confront two key problems: high latency and low bandwidth. The straightforward approach of transmitting and displaying rendered images results in a delay of at least one round–trip time between change in viewpoint and update of display; the time required to transmit an image over a low–bandwidth network can be significantly higher.

Image–based rendering presents one possible solution to this problem. Rather than distributing the viewpoint control loop over a transmission line with delays, we can "close the loop" locally. A remote render server that has a complete description of the scene can render color images augmented with depth (or disparity) at each pixel, then transmit them across the network to a client which can warp and display them in response to local changes of the viewpoint[MMB97].

However, image–based rendering has drawbacks. The client in the remote rendering system briefly described in [MMB97] only warped depth images from one center of projection to form each output image. This led to unsightly exposure errors, where the motion of the viewpoint revealed regions of the model that were not sampled in the original depth image.

By warping together several sets of depth images, we are able to fill in most of the exposure errors seen in [MMB97]. Our splat–like method runs in real time, unlike the mesh–based approach to combining several depth images that was separately discussed in that work.

As the viewpoint moves, the client must request new sets of depth images from the remote renderer. However, the latency in transferring images expected from even high–speed network connections induces large errors in traditional prediction methods. We present an algorithm for choosing locations at which to request new depth image sets that works well in practice and has optimal worst–case performance.

2 The Problem

Across a wide–area network, latency is unavoidable because of the distances involved. When we send messages over a communications link, the delay caused by the propagation speed of the signal is in itself significant, but can be swamped by the cost of routing, especially along congested paths. Added to this is the rendering time of the image to be sent and any overhead in the server. Limited bandwidth can increase the time required for an image to be transmitted, and is also a second–order obstacle since it restricts the ways in which we can attack the latency problem. Historically, aggregate network bandwidth has increased rapidly, but bandwidth across the "last mile" to endsystems increases slowly, and latency improves more slowly still.

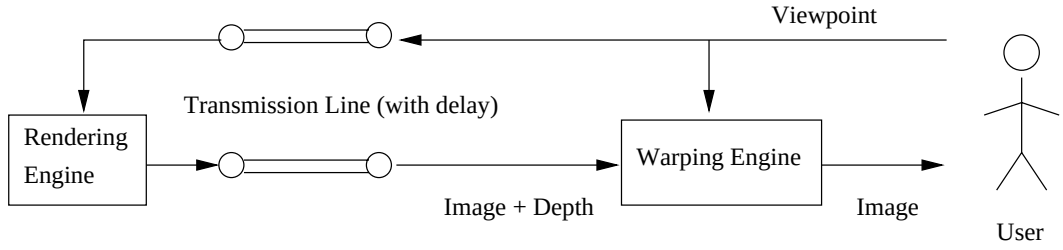

Figure 1: Remote 3D Display

The Image Warp Solution

The image warp [MB95] has the important property that its cost is proportional to the display resolution. Most geometric renderers have instead a cost proportional to the scene complexity. Scenes are rapidly becoming more complex, while display resolutions are increasing relatively slowly; this makes image warping an increasingly attractive alternative to geometric rendering. Image–based warpers can serve as a generalized frame buffer for renderers, allowing the results of a rendering to be seen from different viewpoints, in the same way that frame buffers free the graphics hardware from having to regenerate the entire sceen with every screen refresh and allow computers to display pictures that require more than one refresh time to render.

Exposure Errors

Image–based representations are cheaper than complex scene descriptions, but nothing is free: a complex scene description usually has complex occlusion relationships between objects in the scene, and image–based representations tend to break down when dealing with occlusion. Image–based representations can not sample every part of the scene. As the viewpoint moves, regions of the scene not sampled in the representation come into view; these exposure errors show up as "tears" or "rips" in the scene and are distracting artifacts.

3 Related Work

Our work attempts to apply techniques from a new body of literature (image–based rendering) to remote display. We build directly on the work in [MMB97], which was further developed in other direcions in [Mar99].

3.1 Remote Display

Most systems that want a "remote" display of some scene take one of two approaches: either they transmit raw video from the renderer to the client, or they send the scene data to the client and make the client render for itself.

The best–known examples of database replication are based on the Distributed Interactive Simulation (DIS) standard[ZPF+93, Kat94]. DIS systems require that the geometry of the database be completely replicated on the client, and transmit new vehicle locations (database updates) and other database state changes. For small number of clients DIS is an efficient user of bandwidth, but it maintains a complete copy of the database on all participating machines and allows only certain types of easily–described scene changes.

With the development of VRML[VRM96] and other consumer–oriented distributed 3D graphics protocols for the Internet, commercial attention has increasingly been paid to the problem of distributing views of a database which is not fully or a priori replicated. VRML's standard approach is to replicate the entire database on each client when it first attempts to view the scene. Some extensions allow "streaming" the database, where the client begins rendering before it has received the entire database. Others, such as the proposed Living Worlds multiuser standard[Liv97], divide the database up into small Zones and only require the client to download the geometry of a subset of the zones at any one time.

3.2 Image–Based Rendering

Our warping algorithm and basic approach derive from work by McMillan and Bishop[MB95]. They warped from a single source image to an output image, achieving real–time performance by incrementally evaluating the warping equation and by avoiding Z–buffering through warping pixels in an occlusion–compatible order.

Levoy and Hanrahan[LH96] developed Light Field Rendering, which in a preprocess takes a very large number of images of the scene to be rendered and from them builds a database describing the distribution of radiance through the scene. From this database they can rapidly render their input scene from arbitrary nearby viewpoints. In densely occluded scenes, many Light Fields are needed to capture a complete scene description. The Lumigraph[GGSC96] is a similar approach.

3.3 Exposure Errors

Our system uses multiple source images, each rendered from a different viewpoint, to provide more information about the scene and reduce exposure errors. Several different approaches have been taken to the same problem of reducing exposure errors.

Multiple–Center–Of–Projection Images[RB98] take as input an image from a strip camera rather than the conventional pinhole–model camera. Since every row of pixels in the strip camera has a different center of projection, the algorithm can sample a surface from many different directions in a single image, avoiding exposure errors.

Hanson and Wernert[HW98] use also use distorted multiperspective images to capture a scene, with fairly strong limitations on the geometry of scenes they can capture and camera positions from which they can permit reconstruction. By constraining initial scene geometry and then distroting it they can avoid exposure errors. Their work is aimed at providing a compact representation for animation sequences.

Layered Depth Images[SGwHS98] are generalized images that can have multiple (color, depth) pairs at every (x, y) location in the image. These images can be warped to a new image plane where (color, depth) pairs that were occluded from the original viewpoint (and thus would not have been represented in a standard source image) become visible. If the image plane on which the LDI was originally generated defines one face of a cube, the viewpoint can move to any point within this cube and see a correct image of anything in the original view frustum of the LDI. The LDI has a single center of projection, and thus can undersample surfaces which are nearly parallel to rays through the COP.

Many of these approaches work without explicit depth, which can be an advantage when using graphics hardware that does not give fast read access to the Z–buffer.

Mark, McMillan, and Bishop[MMB97] addressed post–rendering warping. This work is the direct descendant of the ideas they put forward there. In [MMB97], they focused on local display of images; their software simulator for post–rendering warping concentrated on producing high–quality output images and ran offline. They introduced the idea of using image based rendering for *remote* display, with a prototype that warped a single depth image set. Our depth image set maintenance algorithm and compositing of multiple depth image sets in real time are entirely novel.

4 Algorithms

We use McMillan's image warp[MB95]. A pixel at (x, y) in the source image with depth z will be warped to (x', y') in the output image:

$$x' = \frac{ax+by+c\delta+k}{gx+hy+i\delta+m} \quad y' = \frac{dx+ey+f\delta+l}{gx+hy+i\delta+m}$$

δ is a scale factor times $\frac{1}{z}$, and $a \ldots m$ are constants that depend on the source and destination view parameters.

4.1 Compositing

To avoid the expense of Z–compares, McMillan rendered in an occlusion–compatible order. Given the epipolar geometry of a single source image, we can determine an order to warp the pixels such that the painter's algorithm gives correct visibility[MB95, McM95]. However, the epipolar geometry of one im-

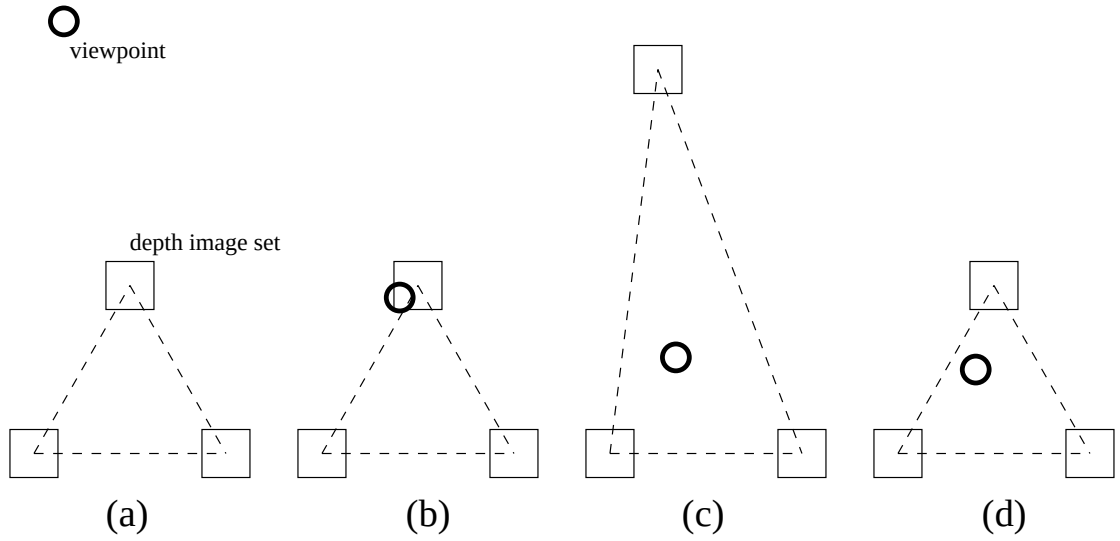

Figure 2: Cases of the algorithm for choosing the next depth image set to request: (a) viewpoint far from triangle formed by three current depth image sets, (b) viewpoint close to one of the current depth image sets, (c) viewpoint close to large or lopsided triangle, (d) viewpoint close to well–formed triangle.

age does not necessarily give us the information we preserve visibility when compositing it with another image. In the absence of another visibility algorithm, we need to use a Z–buffer to combine multiple images. (We can gain a little efficiency by warping the first image in occlusion–compatible order, writing a Z–buffer for it but not doing any Z compares; only the second and subsequent images need pay the overhead of Z comparisons.)

4.2 Depth Image Set Maintenance

Our rendering client always tries to keep three sets of depth images to warp together to form the output. A depth image set is a set of depth images (color plus Z or disparity) with a common center of projection but different viewing parameters. For our architectural walkthrough application, we take four images at 90–degree angles in the horizontal plane, forming four faces of a a cube.

We try to maintain these depth image sets in a triangle around the viewpoint. In the figures that follow, each depth image set is a small box (representing the four image planes of its component depth images) and the viewpoint is a heavy circle. The three depth image sets that are closest to the user define a triangle. For viewpoints within the triangle, warping from these three depth image sets should provide an output image with few exposure errors. However, when the viewpoint leaves the triangle output image quality will probably degrade.

Because of the network's latency and limited bandwidth, there is usually a significant delay between when the client requests a new depth image set from the render server and when it is available to warp. This same delay makes conventional prediction difficult to use. We present the following algorithm to maintain useful depth image sets around the viewpoint; it appears to work well in practice and is guaranteed to form a reasonably–sized triangle within three round trips after the viewpoint becomes stationary[1].

Our algorithm has four cases:

1. The viewpoint is relatively far from the triangle defined by the three depth image sets (figure 2a).

[1] Any algorithm must have this as its worst case.

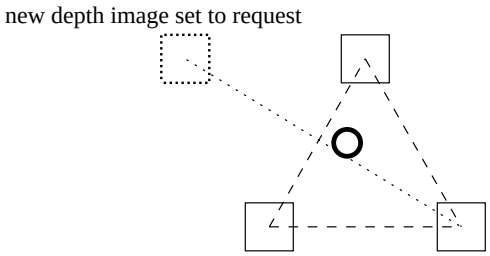

Figure 3: Choosing the next viewpoint for case 4.

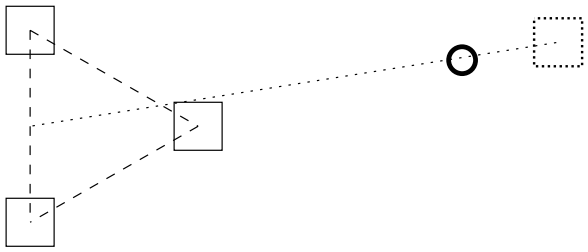

Figure 4: Choosing the next viewpoint for case 1.

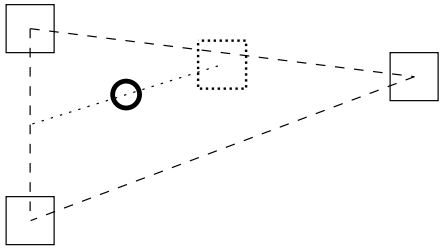

Figure 5: Choosing the next viewpoint for case 3.

2. The viewpoint is relatively close to one of the depth image sets or is outside the triangle and close to one of the lines from the center of the triangle through the depth image sets (figure 2b).

3. The viewpoint is close to the triangle and is not close to any of the depth image sets, but the triangle is large or highly asymmetric (figure 2c).

4. The viewpoint is close to the triangle, is not close to any of the depth image sets, and the triangle is reasonably small (figure 2d).

There are two parameters to this decomposition into cases: the definitions of "close". We define an ideal size s for the triangle, such that each of the depth image sets should be a distance s from the center; we define the viewpoint to be close to the triangle if it is within $1.5s$ of the center of the triangle, and close to a depth image set if it is within $0.3s$ of the center of projection of that depth image set. At any time, the algorithm is attempting to build a triangle of depth image sets of size s, but the triangle actually being used may have some other size $s\prime$ – the average distance from the depth image sets to the center of the triangle.

Cases 1 and 3 are handled as variants of case 4, so we address 4 first. We find the edge connecting the closest two vertices of the triangle; the next depth image set to request from the render server lies s along the perpendicular to the edge through its midpoint (figure 3).

If in case 1 we requested that the next depth image set lie along the perpendicular to the edge connecting the closest two vertices of the triangle, the triangle could become distorted, and would not be guaranteed to include (or even come close to) the viewpoint. Instead, we request the next depth image set s beyond the viewpoint along a line through the midpoint of the edge (figure 4).

For case 3, we need to shrink the triangle toward size s and to make it approximately equilateral. This can be handled by the same procedure as case 1 (figure 5).

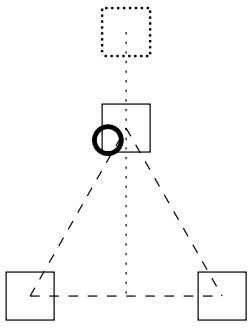

Figure 6: Choosing the next viewpoint for case 2.

In case 2, the viewpoint is outside the triangle, or nearly so, which means that we expect to want to use data from the new depth image set as soon as it arrives (assuming we will continue to move out of the triangle in the current direction). However, which of the other two depth image sets is closer is not a good predictor of what direction we are moving, because they are nearly equally far away. We request the next depth image set from a point $s\prime$ along the line from the center of the triangle through the depth image set closest to the viewpoint (figure 6).

If the next depth image set we want from the renderer server is close to a depth image set we already have, we can wait until the viewpoint has moved so that we want a different depth image set[2]

This algorithm was presented in the plane, since we have only applied our approach to walkthrough scenarios with a constant eye height. The algorithm can be generalized to three dimensions, using four depth image sets in a tetrahedron. We have not implemented this generalization.

5 Implementation

5.1 Rendering Server

For this experiment, our image generator is an SGI Onyx with Infinite Reality graphics. A simple server built on top of SGI Performer renders the model (four images at 90 degree angles from each viewpoint requested by the client), captures the color and depth buffers, converts the depth to disparity, estimates $\frac{\partial \delta}{\partial x}$ and $\frac{\partial \delta}{\partial y}$, and sends three arrays (color, disparity, and disparity derivatives) over the network. Our network is 100Base-T Ethernet.

Computing disparity derivatives at the server increases the network bandwidth consumed. However, cost of this computation was much larger than the client's frame time. If we were to compute disparity derivatives at the client instead, we would have to compute disparity derivatives incrementally, reducing frame rate. Determining which process should compute disparity derivatives is sensitive to network bandwidth and the client's spare processor power.

[2] If we have plenty of local memory, we can instead request additional depth image set that are less likely to be needed but may still be useful.

5.2 Client

The client, which warps and displays the images, is a four–processor SGI Onyx2. One of the four processors is dedicated to communications: it sends viewpoint requests to the server and receives the depth images. The other three processors warp the depth images to the current position and orientation and composite them together. The warp is done entirely in software, as an X–windows application.

5.3 Warp Optimization

An important part of our system's warping implementation is that it clips regions in the depth images which are guaranteed to fall off–screen. Clipping avoids the warping algorithm's expensive per–pixel calculations for pixels that fall in these regions, thus providing a several–fold performance increase over naive implementations. Our clipping algorithm works as follows: The rendering server determines the extremal disparity values for each row of the depth image and passes these values to the client. For a given row in the depth image, the extremal disparity values and the location of the row can be used to construct a parallelogram in the projective two–space used by the warp algorithm. This parallelogram is then clipped against the view frustum conventionally. If the quadrilateral lies entirely outside the view frustum, we skip the entire row. This case is likely for rows in the sides of the cube that fall entirely behind the viewer. If the quadrilateral is partially clipped, then we determine from the clipped polygon's new vertices what portion of the row is potentially visible, and pass only that portion to the per–pixel warper. Finally, if the polygon is entirely visible, that entire row of the depth image needs to be warped pixel–by–pixel. ([PLAdON98] uses a similar technique.)

Parallelization across three processors is another important part of our warping implementation. Recent work by others (including [PLAdON98]) parallelizes the occlusion–compatible rendering order from [MB95] with good load balancing, but since we have multiple input images we instead use a Z–buffer.

5.4 Reconstruction

[MB95] does not discuss the implementation of their forward–mapping image reconstruction technique. Our implementation "splats" pixels into the final image. To have an acceptable output image, we must pay careful attention to the size of splats. The further the viewpoint moves from the center of projection used to capture the depth image, the more opportunity there is for a significant change in pixel size between the two images. Using the wrong size for a reconstructed pixel causes holes or blotches in the final image. To reduce these problems, we spend more CPU time computing reconstruction kernel size than we do splat position.

The expression we use for our axis–aligned reconstruction kernel size is:

$$xsize = \left|\frac{\Delta x'}{\Delta x}\right| + \left|\frac{\Delta x'}{\Delta y}\right| \approx \left|\frac{\partial x'}{\partial x}\right| + \left|\frac{\partial x'}{\partial y}\right| .$$

(ysize is similar.) The partial derivatives are all of the form:

$$\frac{\partial x'}{\partial x} = \frac{\frac{\partial \delta}{\partial x}(c - ix') + a - gx'}{gx + hy + i\delta + m}$$

It would be tempting to simplify the computation by assuming that $\frac{\partial \delta}{\partial x} = \frac{\partial \delta}{\partial y} = 0$, but this simplification leads to small holes in the destination image for large changes in viewing angle between the original depth image and the output image. We approximate the disparity derivatives by taking differences in disparity at adjacent pixels.

6 Results

For simple scenes, the frequency of update is limited by the bandwidth of the network. Using TCP over 100BaseT Ethernet, our system transmits new depth image sets of simple scenes to the client only about once every eight seconds. For more complex scenes, the rendering speed of the server can become the bottleneck, and the frequency with which new depth image sets are received can become arbitrarily low.

The client clips, warps, and composites twelve depth images (three nearby viewpoints with four images each) to produce a single output frame; it displays at about six frames per second, forty–eight times the frame rate we receive from the image generator for simple scenes. XXX RESOLUTION? XXX

Using multiple depth image sets greatly reduces the number of exposure errors, but also reduces the frame rate. If we were to only use a single depth image set, we would expect about ten to fifteen frames per second.

These frame rates were measured while warping along a 1750-frame path through the Brooks House model. XXX RUSS: ADD PICTURE OF MODEL XXX

7 Conclusions

In this paper we presented a system that uses image–based techniques for the remote display of rendered imagery. By using image warping on the display client, we closed the loop between viewpoint changes and an updated display. This lets us avoid the excessive latency and bandwidth suffered by a distributed control loop and present a stabler, more usable interface to the user. Our system warps and composites several depth images together to reduce exposure errors. We also presented the prediction algorithm we use to choose which new sets of depth images to request from the rendering server. Given the latency inherent in remote display, only by choosing reasonable sets of depth images in advance will we have the information we need to actually reduce exposure errors.

Our goal is to bring the visualization capabilities of graphics supercomputers to the remote desktop. The image generator used in this experiment was an SGI Onyx Infinite Reality, but we should also be able to use unique hardware, such as UNC's HP PixelFlow or SGI RealityMonster.

7.1 Future Work

In addition to architectural walkthroughs, we are interested in the remote display of scientific data, particularly height fields. If a height field is rendered from overhead in parallel projection, no part of the data set will be occluded. We can render with just a single depth image, no loss of data or exposure errors, and a high frame rate.

Inverse warping is often avoided because it can be inefficient. However, it may provide an efficient so-

lution to the problem of exposure errors: if we can find one depth image set to forward–warp that will correctly color 80% or more of the pixels in the output image, we can inverse warp those remaining uncolored pixels into one or more alternate depth image sets to to fill in the exposure errors. One forward warp and one-fifth or two-fifths of an inverse warp should be faster than the three complete forward warps we are currently carrying out.

We currently transmit completely uncompressed images over the network; with a reasonable compression scheme we could significantly increase the rate with which new depth image sets of simple scenes would arrive at the client.

8 Acknowledgements

The authors' support included NSF Grant ARI DMR-9512431 and NIH NCRR 5-P41-RR02170 Supplemental Grant as well as funding from DARPA and equipment support from Intel Corporation.

This paper was the continuation of work done in [MBM96, MMB97]. Gary Bishop convinced us that it was time to revisit some of the ideas in the technical report and see if we could build a next–generation real–time system, and is popularly attributed as the source of the comparison of post–rendering warping to frame buffering. Russell M. Taylor II hepled engineer the system and critique the paper. Discussion with Daniel Rohrer, Adam Seeger, and Michael North improved the viewpoint maintenance algorithm. The Brooks House model was used courtesy of the UNC Walkthrough group.

References

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of SIGGRAPH 96*, pages 43–54, New Orleans, LA, 1996.

[HW98] Andrew J. Hanson and Eric A. Wernert. Image-based rendering with occlusions via cubist images. In *Proceedings of IEEE Visualization 1998*, pages 327–334, Research Triangle Park, NC, 1998.

[Kat94] W. Katz. Military networking technology applied to location–based, theme park and home entertainment systems. *Computer Graphics*, 28(2), May 1994.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of SIGGRAPH 96*, pages 31–42, New Orleans, LA, 1996.

[Liv97] Living worlds specification draft 2, April 1997.

[Mar99] William R. Mark. *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping.* PhD thesis, University of North Carolina, April 1999.

[MB95] Leonard McMillan and Gary Bishop. Head-tracked stereo display using image warping. In *Stereoscopic Displays and Virtual Reality Systems II*, pages 21–30, San Jose, CA, 1995.

[MBM96] William R. Mark, Gary Bishop, and Leonard McMillan. Post–rendering image warping for latency compensation. Technical report, UNC Chapel Hill, Jan 1996.

[McM95] Leonard McMillan. Computing visibility without depth. Technical report, UNC Chapel Hill, 1995.

[MMB97] William R. Mark, Leonard McMillan, and Gary Bishop. Post–rendering 3d warping. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pages 7–16, Providence, RI, 1997.

[PLAdON98] Voicu Popescu, Anselmo Lastra, Daniel Aliaga, and Manuel de Oliveira Neto. Efficient warping for architectural walkthroughs using layered depth images. In *Proceedings*

of IEEE Visualization 1998, pages 211–215, Research Triangle Park, NC, Oct 1998.

[RB98] Paul Rademacher and Gary Bishop. Multiple–center–of–projection images. In *Proceedings of SIGGRAPH 98*, pages 199–206, Orlando, FL, 1998.

[SGwHS98] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *Proceedings of SIGGRAPH 98*, pages 231–242, Orlando, FL, 1998.

[VRM96] The virtual reality modeling language specification version 2.0, August 1996.

[ZPF+93] M. Zyda, F. Pratt, J. Falby, P. Barham, and K. Kelleher. Npsnet and the naval postgraduate school graphics and video laboratory. *Presence*, 2(3):244–258, 1993.