Navigating Static Environments Using Image-Space Simplification and Morphing

Lucia Darsa* Bruno Costa Silva† Amitabh Varshney‡

State University of New York

Abstract

We present a z-buffered image-space-based rendering technique that allows navigation in complex static environments. The rendering speed is relatively insensitive to the complexity of the scene as the rendering is performed *a priori*, and the scene is converted into a bounded complexity representation in the image space. Real-time performance is attained by using hardware texture mapping to implement the image-space warping and hardware affine transformations to compute the viewpoint–dependent warping function. Our proposed method correctly simulates the kinetic depth effect (parallax), occlusion, and can resolve the missing visibility information by combining z-buffered environment maps from multiple viewpoints.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - Display Algorithms, Viewing Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation, Texture; I.4.8 [Image Processing]: Scene Analysis - Range data.

Additional Keywords: image-based rendering, synthetic environments, image morphing, visibility.

1 Introduction

Traditional approaches to graphics acceleration for the navigation of a three-dimensional environment have involved:

- reducing the geometric complexity of the scene, by using level-of-detail hierarchies (Turk, 1992; Schroeder, Zarge and Lorensen, 1992; Rossignac and Borrel, 1993; Cohen et al., 1996; Hoppe, 1996; DeRose, Lounsbery and Warren, 1993; He et al., 1996), and by visibility-based culling(Airey, 1990; Teller and Séquin, 1991; Greene and Kass, 1993; Luebke and Georges, 1995; Greene, 1996).

- reducing the rendering complexity by using texture mapping (Blinn and Newell, 1976; Blinn, 1978), and by using various levels of complexity in shading and illumination models(Bergman et al., 1986).

- exploiting frame-to-frame coherence with one of the above (Bishop et al., 1994; Xia and Varshney, 1996).

However, as the complexity of the three-dimensional object-space has increased beyond the bounded image-space resolution, image-based rendering has begun to emerge as a viable alternative to the conventional three-dimensional geometric modeling and rendering, in specific application domains. Image-space-based rendering has been used to navigate (although with limited freedom of movement) in environments modeled from real-world digitized images (Chen, 1995; Szeliski, 1996; McMillan and Bishop, 1995). More recent approaches (Gortler et al., 1996; Levoy and Hanrahan, 1996) generalize the idea of plenoptic modeling by characterizing the complete flow of light in a given region of space. A clever combination of simple geometric building blocks with view-dependent textures derived from image-based rendering (Debevec, Taylor and Malik, 1996) has resulted in a viable technique for navigation in environments that can be described by those simple blocks. The potential of image-based rendering specifically for the navigation in generic synthetic environments, however, has been investigated in fewer instances (Chen and Williams, 1993).

In this paper we present a conceptual discussion and an implemented system for the problem of image-based rendering using image-space simplification and morphing. Given a collection of z-buffered images representing an environment from fixed viewpoints and view directions, our approach first constructs an image-space simplification of the scene as a pre-process, and then reconstructs a view of this scene for arbitrary viewpoints and directions in real-time. We achieve speed through the use of the commonly available texture-mapping hardware, and partially rectify the visibility gaps ("tears") pointed out in previous work on image-based rendering (Chen and Williams, 1993; Chen, 1995) through morphing.

2 Image-based Navigation

Image-based rendering uses images as the basic primitive for generating other images, as opposed to the more traditional approach that renders directly from geometric models. Images are a sampled version of the scene, viewed from a certain position and direction, and not a full representation of the actual scene. Since the images used as the basis for the rendering are generated and viewed from different points of view, they represent view-dependent information only for the originally generated positions. Thus, image-based rendering methods have an inherent difficulty in dealing with view-dependent effects, such as specular highlights, reflection, and refraction. However, view-independent effects that are usually very expensive to simulate – such as diffuse reflections, soft shadows and caustics – can be used with image-based rendering without any additional runtime cost.

Navigation in an environment using image-based rendering can be classified into four different levels based on freedom allowed in user-movement:

*luciad@msn.com, Recently graduated from the Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400

†brunos@microsoft.com, Microsoft Corp., One Microsoft Way, Redmond WA 98052-6399

‡varshney@cs.sunysb.edu, Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400

1. Discrete viewpoints, discrete view directions
2. Discrete viewpoints, continuous view directions
3. Continuous viewpoints, discrete view directions
4. Continuous viewpoints, continuous view directions

The first group is the simplest approach, providing a very limited immersive experience and interaction. The images are rendered or digitized for selected positions and selected viewing directions and during navigation the one that is closest to the desired is displayed. One early instance of this situation is described in (Lippman, 1980), and the same concept has been used at the consumer level more recently (Cyan, 1994).

The second group uses one image, or a series of images stitched together, to represent the environment around a certain point of view, which is equivalent to providing a complete sample of the plenoptic function (McMillan and Bishop, 1995) for that point. This form of information enables the simulation of a rotation of the observer, around the original point of view, to look in any direction by reprojecting the given images to the new viewing frustum. To allow observer limited translation (at discrete viewpoints), the solution is to have a set of environment maps, each computed for a different viewpoint. If these points are carefully selected and not very far from each other, it is possible to simulate movement by selecting the closer environment map. This jump between discrete viewpoints around which one can rotate almost freely allows for a quality of simulation that can be considered acceptable for some situations, as shown by current applications of Quicktime VR (Chen, 1995), for instance.

The third group has been bypassed in the literature and its results are subsumed in the fourth group in which the user is allowed freedom to continuously translate and rotate. In "View Interpolation for Image Synthesis" (Chen and Williams, 1993), the mapping function and the depth are obtained from the camera model and the rendering. The mapping is applied as an image warping transformation, and a binary combination of the images is performed based on depth—the front most pixel wins. That technique is not actually based on environment maps, but on single images with depth information. The movement of the observer had to be restricted, though, to achieve the desired performance. In "Plenoptic Modeling" (McMillan and Bishop, 1995), the nodes are represented by cylindrical maps, which the authors describe as a complete sample of the plenoptic function. They focus on the image-based modeling aspect of the problem, concentrating on the techniques for reconstruction of a complete sample of the plenoptic function from a set of overlapping partial samples from non-computer-generated sources (photographs or video frames). The navigation, however, was limited and required closely spaced nodes and user input for proper matching.

When the environment is modeled from photographs or video, the depth information has to be inferred from the disparities induced by translations of the camera. This is an important problem in computer vision to which a considerable effort has been dedicated (Aggarwal and Nandakumar, 1988). However, the driving problem for our work is smooth navigation in complex computer-generated virtual environments, that are slow to render, and for which we have access to z-buffered images at fixed viewpoints and view-directions. Thus, our work falls in category four listed above.

3 Environment Mapping and Morphing

Changes of visibility that occur as an observer moves freely in an environment can be simulated by using precomputed views of the scene at selected viewpoints. A *node* is associated with every selected viewpoint and consists of an environment map with depth and color information for every direction. This is essentially an extension to the plenoptic function, that also associates depth information to each direction, in addition to color.[1]

Figure 1: Visibility: (a) Rotation (b) Translation.

Each node provides limited information about the world, which is not sufficient to determine the view from an arbitrary viewpoint and direction. If the desired visualization parameters differ only in the direction, there is no parallax and no new areas can become visible (see Figure 1). If they differ in the position, however, parallax is introduced and the restricted information provided by a single node becomes evident.

This problem can be overcome by combining the information from neighboring nodes—through node morphing—to create an image for any viewpoint and direction. Morphing two nodes involves two warpings to register the information, followed by a combination (Gomes et al., 1995). The particular case of image morphing is extensively discussed in (Wolberg, 1990). The depth information and the visualization parameters allow the determination of the mapping functions between the original views and the new arbitrary view. After applying these mappings, the warped information can be combined with local control, used to determine the predominant information at each region.

A more detailed discussion of this form of morphing is presented in the next sections. We will focus on a simpler case of two planar z-buffered images, although it can be directly applied to environment maps.

3.1 Environment Map Warping

An adequate mathematical model for a *continuous* image with depth is a function that relates points in a subset of the Euclidean plane to colors in a color space and to depths. A z-buffered image can be considered as a function $I^z : U \subset \mathbf{R}^2 \rightarrow C \times \mathbf{R}$, where C is a color space.

The class of rendering processes that are relevant to our application are those that are able to yield a z-buffered image. Each of those processes can be seen as a function R that maps a scene, $\mathcal{S}$ (the collection of models and information that define a scene) and a projection $\mathcal{P}$ (a transformation derived from a set of visualization parameters) into a z-buffered image:

$$R : \mathcal{S} \times \mathcal{P} \rightarrow \mathbf{I}^z, \tag{1}$$

$$\mathcal{P} = \{P : \mathbf{R}^3 \rightarrow \mathbf{R}^3\}. \tag{2}$$

Given a z-buffered image I_1^z and the projection transformation P_1 that originated this image, we are interested in applying a reprojection using a new set of visualization parameters described by P to obtain a new image I^z. Our specific case is depicted in Figure 2. This is a transformation of the domain of definition of the original image, or a warping transformation $W = P \circ P_1^{-1}$, that essentially reprojects the image to a different point of view[2].

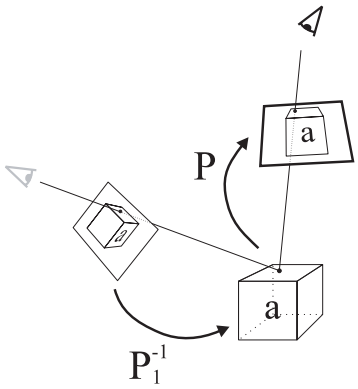


Figure 2: Environment map warping.

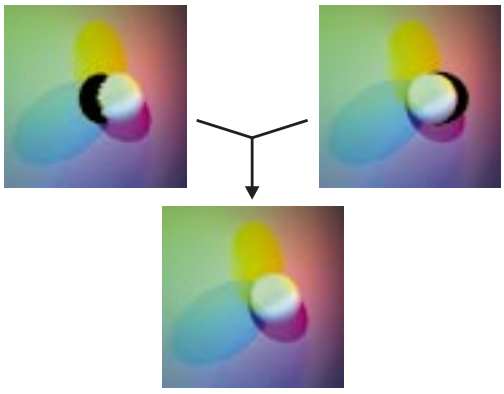


Figure 3: Blending from two nodes (visibility gaps in black).

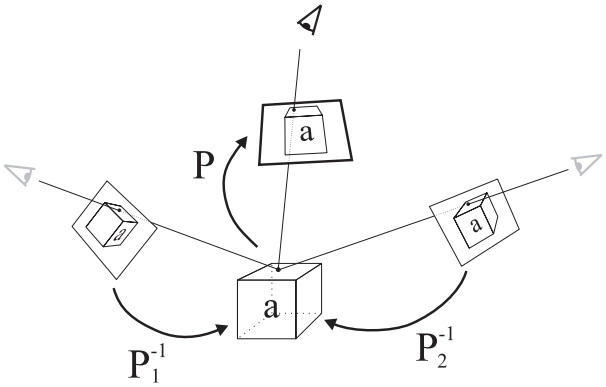


Figure 4: Environment map morphing.

3.2 Environment Map Morphing

The information from a single node is not sufficient to generate an arbitrarily different view, that is, this particular form of warping is not described in general by an onto function. Therefore, to cover the domain of the resulting image I^z it is generally necessary to combine the information from two or more nodes.

This combination is exemplified in figure 3, which shows an image obtained in real-time from the combination of two nodes. The top left node was originally generated as a top view of the sphere; the top right node, as a side view. Notice how the visibility information that is missing from the top view is completely filled by the second node. Similarly, the visibility gaps in the second node are covered by the first.

By applying the warping process described in the previous section to each node individually, we get two different z-buffered images $I_1^{z'}$ and $I_2^{z'}$—from P_1 and P_2, respectively—as illustrated in figure 4. What remains is the combination of $I_1^{z'}$ and $I_2^{z'}$, a range transformation B which, for each point (x, y), depends solely on the values of the z-buffered images at that position, resulting in the image $I_f^z = B(I_1^{z'}, I_2^{z'})$. Different forms of this blending function are described in section 6.

4 Image-Space Simplification

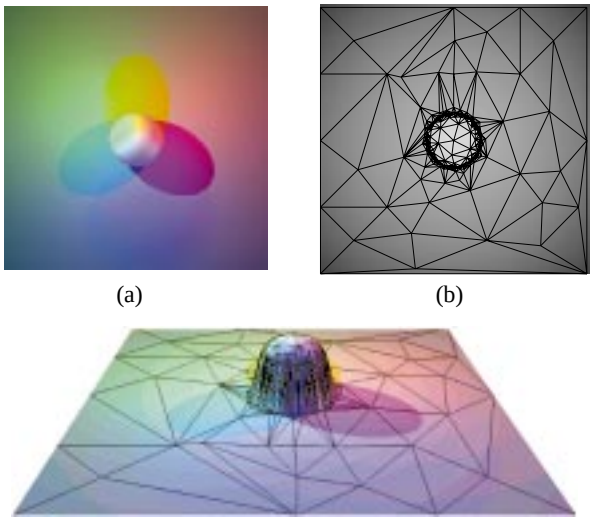
Given an environment map with depth and color information at a viewpoint, we have seen that it is possible to create views from new positions and directions by appropriately warping the environment map. To generate environment maps for viewpoints intermediate to the previously selected nodes, we morph neighboring environment maps into an intermediate one.

Our solution to the image-space-based rendering problem simplifies the environment, as seen from a given viewpoint, by linear polygons. This polygonal mesh is created by triangulating the depth information associated with the environment map, as shown in the example in Figure 5(b). Each triangle in this mesh represents an object (or part of an object) at a certain depth.

The parallax effect can then be correctly simulated by warping each of these triangles appropriately. Since image warping can be efficiently performed with hardware assistance through texture mapping, we determine the appropriate projective transformation which is then applied to this mesh textured by the environment

[1] This implies indirectly that the modeled world is opaque.

[2] It also fills in the gaps in the domain of definition with a background color/depth, and solves foldovers using z-buffering.



(a) (b)

(c)

Figure 5: Range image triangulation: (a) Input image; (b) 2D texture coordinates; (c) 3D triangulation textured by input image

map colors. The hardware z-buffer is used to resolve occlusions, or mesh *foldovers*. Multiple nodes are used to fill in the gaps resulting from mesh *tears* by combining z-buffered images from various nodes using alpha blending and the stencil or the accumulation buffer (Neider, Davis and Woo, 1993).

The polygonal mesh derived from the depth information is in fact a 3D triangulation that, when viewed from the original viewpoint, will look exactly like the flat image. The triangulation can be reprojected to any other arbitrary viewpoint in space by using standard viewing transformations, such as in the side view shown in Figure 5(c).

4.1 Choice of the Environment Map Geometry

Although spherical maps are the most natural way to represent the environment information, they are not necessarily the most convenient or efficient. Other representations have been used, such as cubical (Greene, 1986) and cylindrical maps (Chen, 1995; McMillan and Bishop, 1995). Spheres are difficult to represent digitally without significant variation in the information density, whereas cylinder-based techniques have the problem of limiting the field of view to avoid dealing with the caps. Cylindrical maps are convenient for generating panoramic images—by stitching together several partially overlapping photographs from planar rotations of the view direction.

Although cubes do not represent texture information homogeneously, the cube representation is the easiest to obtain for synthetic images and can be stored as six conventional rectangular images, which can be output by virtually any rendering software (see Figure 6). Moreover, each of the sides can be considered independently during most of the process. These reasons led us to use cubical environment maps. A reprojection of a cube texture-mapped by the environment of Figure 6 is shown for a given view direction in Figure 7; the seams of the cube are highlighted to indicate the new viewing direction.

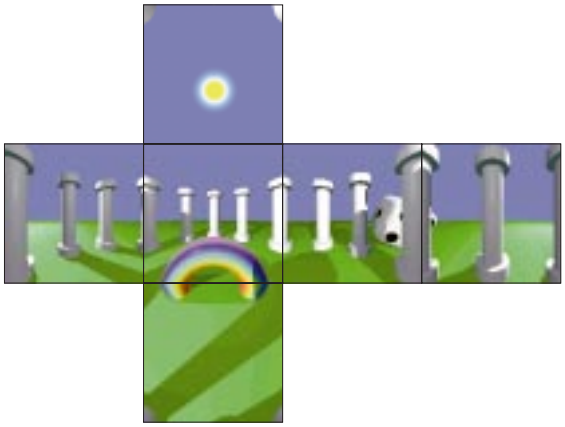


Figure 6: Unfolded cubical environment map.

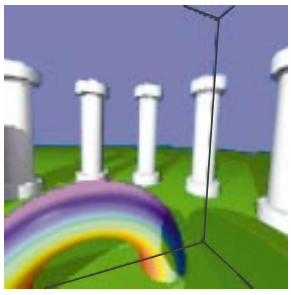


Figure 7: Cube reprojection in a given viewing direction.

4.2 Image-space Triangulation

This step of the algorithm corresponds to the inverse projection that takes the z-buffered image space into the object space (such as P_1^{-1}, in Figure 2). The goals of the triangulation step are:

- to match the object silhouettes, which correspond to depth discontinuities in the range images, as accurately as possible;
- to detect the areas in the depth information that are almost linear, and approximate them by triangles, which effectively corresponds to a view-dependent simplification of the object models.

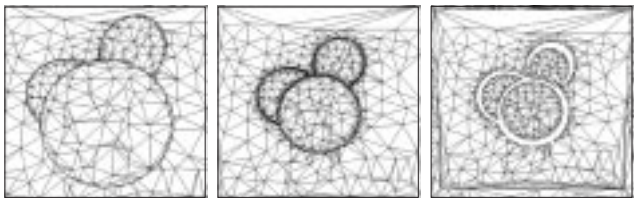
Since this must be done while minimizing the error in the scene representation, it is important to subdivide the non-linear areas of the objects that are away from discontinuities as well, so that the geometry representation is more faithful, and the parallax effect *within* the objects can be simulated. Also, due to the perspective projection, the more distant an object is, the less relevant it is to the observer, and the less noticeable is its parallax effect. In this way, the mesh should approximate the object edges, and its sampling density should be inversely proportional to the depth.

The implemented algorithm constructs an image-space Delaunay triangulation using a Voronoi diagram to adaptively sample the image based on the depth component, similar to previous work by the authors (Darsa and Costa, 1996). Figure 8 shows an image, its depth image and the corresponding triangulation (viewed from the original point of view), in which the farthest objects are sampled more sparsely, and the areas near the edges are sampled finely.

Another triangulation created by inverse projecting depth information to 3D is shown in the sequence in Figure 9, where an ob-

(a) (b) (c)

Figure 8: Triangulation using depth and discontinuity.



(a) (b) (c)

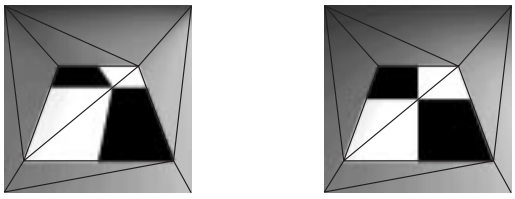
Figure 9: Handling Visibility Changes in Translation.

server is moving away from the spheres—9(b) shows the view from the position where the node was generated. Note how the visibility changes are correctly handled, with the closer sphere covering a greater part of the other spheres (9(a)) and the visibility gaps appearing where no information was available (9(c)). Methods to fill-in these visibility gaps using information from neighboring nodes are presented in Section 6.

The projection transformation that relates the triangulation and the z-buffered image, when applied to each vertex of the 3D triangulation, yields the texture coordinates that have to be assigned to that vertex.

4.3 View-dependent Texture Mapping

Simply assigning the texture coordinates, however, does not result in the desired behavior, since the graphics engine generally interpolates the interior pixels of each triangle using perspective correction. This texture mapping correction, essentially a divide by z, is performed on a per-pixel basis for the texture coordinates (Segal et al., 1992). In our case, however, the texture maps already have the necessary perspective distortion in them. Letting the hardware perform perspective correction results in an incorrect double perspective effect. Figure 10(a) shows such double perspective effect for an oblique view of a checkerboard pattern mapped to a square. Disabling perspective correction is necessary to obtain the correct effect shown in (b). To achieve this effect in OpenGL we transform the texture coordinates according to the depth of the corresponding vertices so that the automatic perspective correction is nullified (Segal, 1996). Details of this transformation appear in the Appendix.



(a) (b)

Figure 10: Handling Perspective: (a) Double Perspective; (b) Compensated.

(a) (b)

Figure 11: Triangulation qualities from two different nodes.

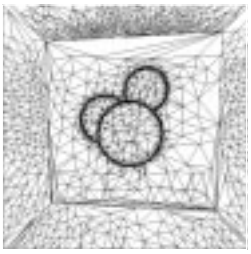


Figure 12: Node triangulation viewed from within the node.

4.4 Triangle Quality Measure

Each triangle of the mesh carries a certain amount of texture information, which we will measure by a triangle quality factor. This quality is related to the angle that the normals of the triangles make with the view ray, i.e., how oblique is the triangle in relation to the image plane for a perspective projection. Triangles with greater angles are projected to proportionally smaller areas in 2D and thus, less pixels will be texture mapped to it. The quality that is assigned to each triangle can be calculated therefore as the dot product between the normal to each triangle and the ray direction:

$$w = \|N.V\| = |\cos(\theta)|$$

The quality of the triangles is a static property, that is computed before navigation for the observer in the original position. It denotes the proportion of pixels from the texture map that are used in the representation of this triangle. When the observer moves, this quality indicates how much a triangle can be warped without noticeable error. If the quality of a triangle is low, a modification in the observer position can cause it to become more visible, and the low quality of its texture would become apparent. In this case, we combine or replace it by a better quality triangle, from a triangulation of another node. Figure 11 shows the qualities—indicated as gray levels with white being the best—of the triangles of two different nodes viewed from the same position.

5 Single Node Navigation

A node consists of a cubical environment map and its triangulation as discussed in Section 4. An example of this is shown in Figure 12. The navigation inside a node involves projecting these triangulations for a given viewpoint and viewing direction. The projection and the subsequent z-buffering correctly handle the visibility and the perspective for regions where adequate information is available.

The six sides of a node are not normally all visible at once. A cube divides the space into six pyramidal regions that are joined by

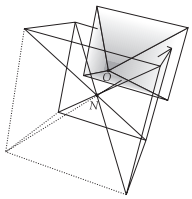
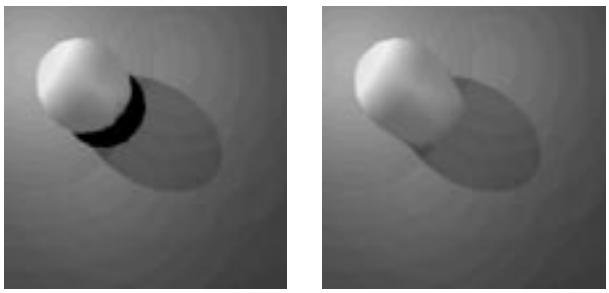


Figure 13: View frustum culling.



(a) (b)

Figure 14: Visibility gaps: (a) black; (b) filled by linear interpolation.

their apices at the center of the cube . We cull the triangulations in large batches, by computing the intersection of the viewing frustum with the six pyramidal regions to determine which sides of the node can possibly take part in the view. In Figure 13, for instance, just the triangles from the highlighted sides are sent through the graphics pipeline.

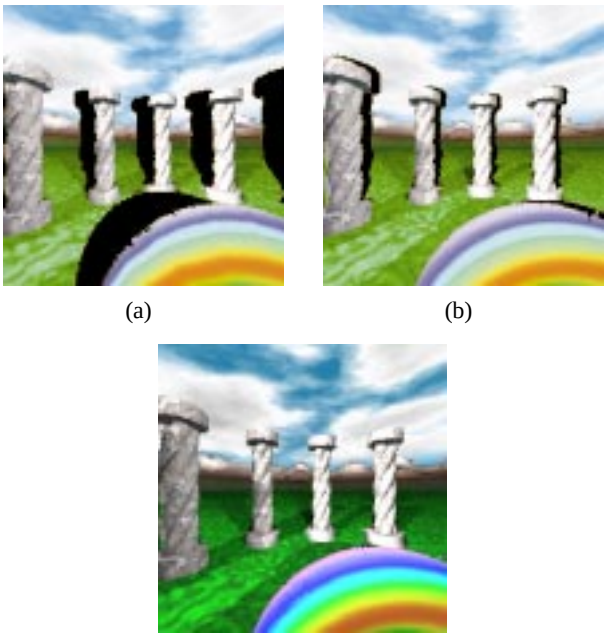
When an observer translates, regions not visible from the original viewing parameters appear. These visibility gaps can be either shown in a background color or can be filled by a linear interpolation of the colors of their vertices. Both options are shown in Figure 14, where the observer has moved down and to the right from the original position, which was directly above the sphere. In an image-based navigation system, the visibility information from the original viewpoint is projected to a 2D plane and any obscured objects are "lost". Therefore, the system at this stage does not have any information to fill uncovered areas and an interpolation is just a crude approximation that is acceptable for very small gaps. Nevertheless, some systems rely on this type of technique. We shall next discuss an approach that uses information from other nodes to fill-in the missing visibility information where possible.

6 Multiple Node Navigation

Given a set of nodes, solving the visibility problem for a certain viewpoint and direction involves two subproblems: selecting the appropriate nodes and combining the information from these nodes. If the nodes are uniformly distributed, the selection of the nodes that are closer to the observer is a simple solution that yields acceptable results. This is the approach that we have implemented. The remainder of this section discusses the combination of information from two nodes. Combination of information from three or more nodes proceeds in the same manner if we are iteratively combining information from two nodes at a time, until all or most of the visibility gaps are filled.

The information from two different nodes has to be merged to form a new view of the scene in real-time, combining or replacing triangles based on their quality (see section 4.4). We next some ways to perform such merging.

6.1 Mesh Layering



(a) (b)

(c)

Figure 15: Mesh Layering

This node merging technique begins by projecting the triangles of the visible triangulations from the node that is closest to the observer. Clearly, if the observer is not at the center of the node, visibility gaps can appear. The next closest node is then reprojected, and the process can be repeated until all the visibility gaps are filled, or a subset of the neighboring nodes has been used. A z-buffer is employed for hidden surface removal. Although this simple scheme fills most of the visibility gaps, it suffers from the drawback that, for triangles with similar z-values but different qualities, the winner is determined solely by the depth ordering, which can cause parts of low quality triangles to dominate over high quality ones that are slightly farther. Figure 15 shows the visibility gaps (in black) from two different nodes. Notice the the discontinuities, especially in the rainbow in Figure 15(c) generated using this method.

6.2 Binary Merge

To allow the quality measure to play a more important role, we store the quality of each triangle on a pixel-by-pixel basis to the stencil or alpha buffer during its rasterization. The combination of the two images is now performed by first comparing the z-values. If the z values are sufficiently different, the pixel that is closer to the observer wins; otherwise, the pixel that is output to the final image will be the one that has a higher quality. Although this function is not a continuous blend it yields good results. Figure 16 shows a combination using this technique (see Figure 11 for the triangle qualities).

Figure 16: Binary merge.

Figure 17: Weighted blending.

6.3 Simple Weighted Blending

This method is an extension of the binary merge approach with the difference that for close z-values, the output pixel is an interpolation of the pixels from different nodes, with the alpha factor based on the relative quality values. Since the quality values of two triangles do not, in general, sum to 1, we choose the output pixel to be a quality-weighted average of the input pixels:

$$p = \frac{q_1 p_1 + q_2 p_2}{q_1 + q_2}$$

The result using this computation is shown in Figure 17, where the combination produced a smoother image.

6.4 Positional Weighted Blending

The weighted average technique can be further refined by considering the position of the observer: the closer is the observer to the center of a node, the more should be the influence of that node on the resulting image. This is achieved by multiplying the qualities stored in the buffers by a factor proportional to the distance d_i of the observer from the center of the node i:

$$p = \frac{t q_1 p_1 + (1-t) q_2 p_2}{t q_1 + (1-t) q_2}, \quad t = \frac{d_2}{d_1 + d_2}.$$

This solution produces a smooth morphing between the nodes (see Figure 18). When the observer is exactly at the center of a node, the resulting image is exact, and it becomes a combination of the two nodes as the observer moves. Although the accumulation buffer of OpenGL can produce a weighted average of two images it cannot be used here directly, since q_1 and q_2 do not sum to 1. For this, q_i must be normalized on a pixel-by-pixel basis which makes this approach impractical for OpenGL. However, in other systems it might be possible to directly execute this solution in the graphics pipeline.

Figure 18: Positional weighted blending.

Figure 19 shows frames of an animation obtained by blending two nodes using the weighted average technique in software. The center of the first node is directly in front of the rainbow torus, and the second is to the its left and front of the first center. In the navigation, the observer starts at the center of the first node, is translated to the left, then rotates to the right in the last two frames.

7 Results and Conclusions

We have described an image-based rendering technique for navigation of 3D environments by using viewpoint-dependent warping and morphing. The method relies on a set of cubical environment maps that are pre-rendered from a collection of fixed (and preferably uniformly distributed) viewpoints within the virtual model. Every given viewpoint is represented by a node that consists of a cubical environment map and an associated 3D triangulation of the six faces of the map. We have described the construction of such triangulations of the faces and discussed navigation by combining information from multiple nodes. Our scheme relies heavily on, and derives its speed from, hardware texture mapping facilities.

We have tested our implementation on a model of the Stonehenge generated by us. The initial model was ray-traced and two cubical environment maps, each consisting of six 256×256 images (with depth) were generated. From these $786K$ data points, we obtained a simplified representation consisting of a total of $30K$ texture-mapped triangles using a top-down approach to generate a Delaunay triangulation. We have tested our system on a single SGI Challenge R10000 processor with – one Raster Manager, Infinite Reality with 64 MB of texture memory, and 2 MB of secondary cache. We compared mesh layering, binary merge, weighted blending, and positional weighted blending schemes for the same navigation path consisting of 250 frames between the two nodes. For mesh layering we achieved an average frame-rate of 9.59 frames per second, for binary merge 4.27 frames per second, for weighted blending 3.58 frames per second, and for positional weighted blending 3.53 frames per second. Our current implementation does not use triangle strips; from our past experience with triangle strips, the above frame-rates should roughly double with triangle strips. The results are shown in Figures 16, 17, 18, and 19. The differences amongst these figures although present are subtle and not very obvious at the scale at which these figures have been reproduced in this paper. The difference between mesh layering and positional weighted blending, for instance, is more obvious in Figure 20 in the spurious triangles near the left side of the sphere in the mesh layering approach.

In our current implementation, reprojection of the second node is done for the entire image. However, to speed-up the results, a mask can be created to determine the visibility gaps exactly and the secondary projections restricted to the missing parts of the scene. This can be done by using a binary stencil and modifying the viewing

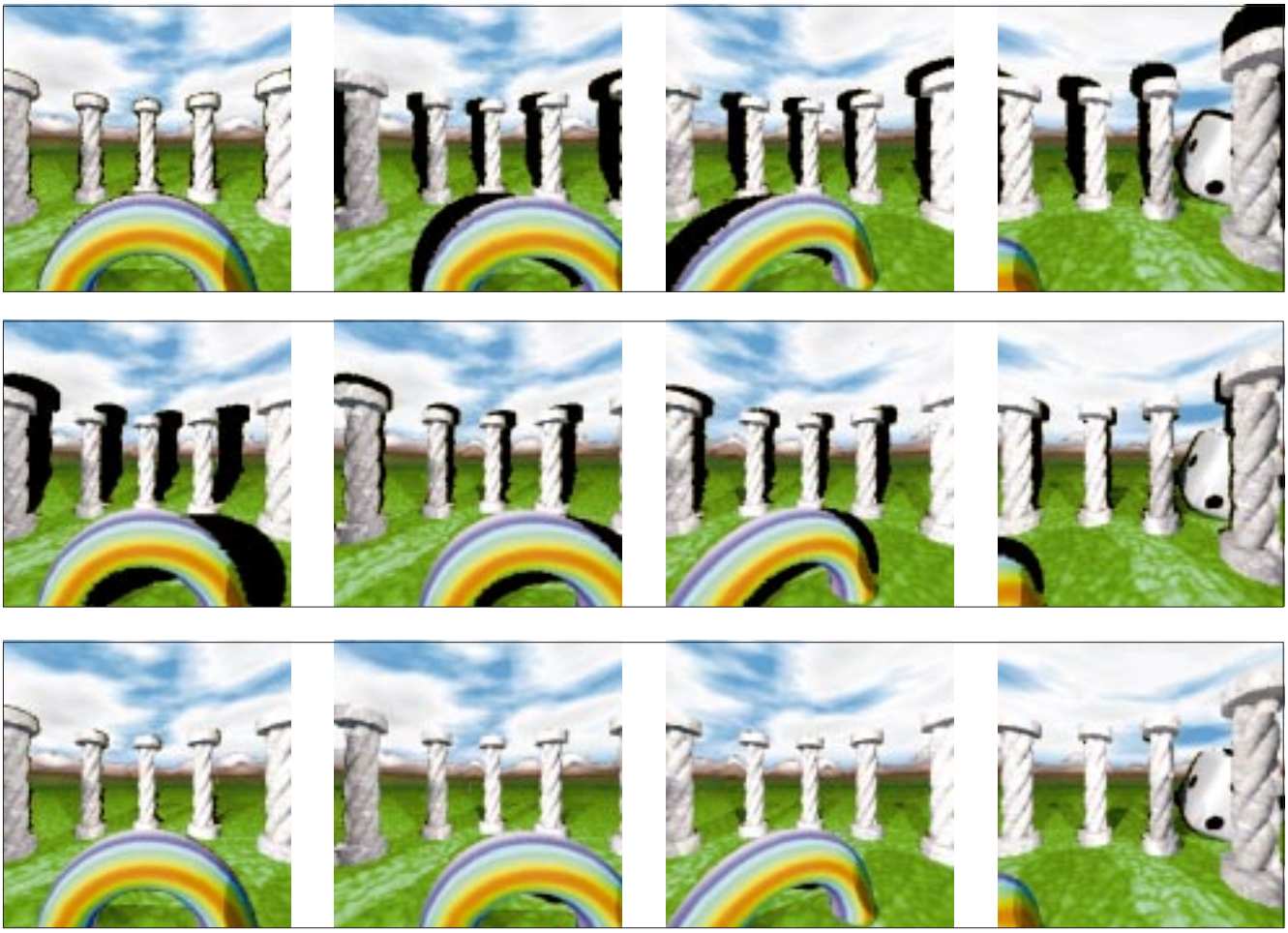
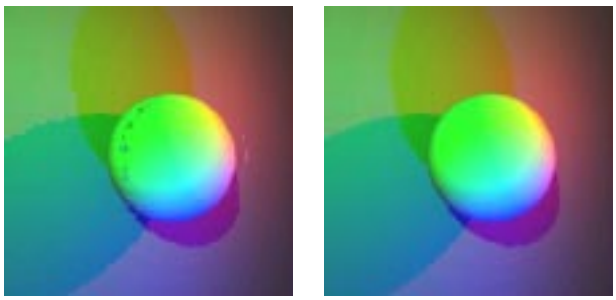


Figure 19: Navigating in Image-Space: Top Row – Frames from Node 1; Middle Row – Frames from Node 2; Bottom Row – Frames from Positional Weighted Blending of Node 1 and Node 2



(a) (b)

Figure 20: Mesh layering (a) versus Positional weighted blending (b).

frustum to be restricted to the bounding box of the visibility gaps. An approach similar to this has been shown to work well (Luebke and Georges, 1995).

The triangulation scheme that was used could take advantage of further improvements. Its current version is better suited when the sampling process is expensive, since it never discards samples. In many situations, such as with range images, the information is already fully sampled, and resampling it incurs a minimal cost. In these cases, it is better to use a technique that works bottom up, by trying to combine similar areas that can be approximated by a linear polygon, instead of top down, trying to guess sample positions. Also, the triangulation of each of the sides of the environment map could not actually be performed in an entirely independent way, to avoid cracks at the seams. A single integrated triangulation step may yield a better junction between the sides.

The use of multi-resolution images, as well as multi-resolution triangulations, could be useful, if the nodes are positioned sparsely.

The position of the nodes is currently determined manually by the user, as part of the scene modeling. Ideally, a minimum amount of nodes should be positioned in such a way so as to cover all the areas of interest in the scene. Also, the problem of selecting the subset of the nodes that will be combined at a given position during navigation must be solved in an efficient way, so that the node reprojections are kept to a minimum. There is a clear coherence in the node selection that adapts itself to a working set model, where the active nodes are cached, and a node replacement occurs sparsely.

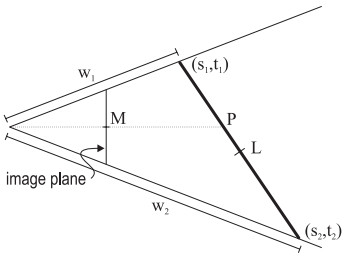


Figure 21: Linear and perspective corrected interpolations.

8 Acknowledgements

We deeply appreciate the thorough review and detailed and insightful comments by the anonymous referees. We would like to thank CNPq (Brazilian Council of Scientific and Technological Development) for the financial support to first two authors. This work has been supported in part by the National Science Foundation CAREER award CCR-9502239. This research was performed at the Visualization laboratory at SUNY Stony Brook.

A Disabling perspective correction

A texture coordinate in OpenGL is specified by a tuple (s, t, r, q). After multiplying the texture coordinates by the texture matrix, OpenGL interprets the result (s', t', r', q') as homogeneous texture coordinates. This computation is performed at each triangle vertex and the information is interpolated by the rasterizer to yield a value for each pixel. If the interpolation used is linear, the results differ from those obtained from the actual computation at each pixel, as illustrated in Figure 21. For the computation of the texture coordinates at the pixel M, at the center of the image, linear interpolation yields $(s_1 + s_2)/2$, which is incorrect since those are the texture coordinates of point L. With perspective correction the computation at M must yield the coordinates of P. This is done by linearly interpolating $(s_1/w_1, t_1/w_1)$ and $(s_2/w_2, t_2/w_2)$ at each pixel where w_i is the homogeneous coordinate of the polygon vertex i. The r and q texture coordinates are also divided by w_i, and interpolated in the same way, although they are not needed for simple texture mapping. At each pixel, a division is performed using the interpolated values of $(s/w, t/w, r/w, q/w)$, yielding $(s/q, t/q)$, which are the final texture coordinates. To disable this effect, which is not possible in OpenGL directly[3], we transform, a priori, the original texture coordinates (s_i, t_i) into $(w_i s_i, w_i t_i, 0, w_i)$, so that at the end of the transformation performed by OpenGL, we have (s_i, t_i), at no performance cost. In our case, the 3D triangulation includes the depth of each vertex which is the required w value.

[3] In other APIs, such as Microsoft Direct3D, this feature can be disabled with a single command.

References

Aggarwal, J. K. and Nandakumar, N. (1988). On the computation of motion from sequences of images–a review. *Proceedings of the IEEE*, 76(8):917–935.

Airey, J. M. (1990). *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC 27599-3175.

Bergman, L. D., Fuchs, H., Grant, E., and Spach, S. (1986). Image rendering by adaptive refinement. In Evans, D. C. and Athay, R. J., editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 29–37.

Bishop, G., Fuchs, H., McMillan, L., and Zagier, E. (1994). Frameless rendering: Double buffering considered harmful. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 175–176. ACM SIGGRAPH.

Blinn, J. F. (1978). Simulation of wrinkled surfaces. In *SIGGRAPH '78*, pages 286–292. ACM.

Blinn, J. F. and Newell, M. E. (1976). Texture and reflection in computer generated images. *CACM*, 19(10):542–547.

Chen, S. E. (1995). Quicktime VR – an image-based approach to virtual environment navigation. In *Computer Graphics Annual Conference Series (SIGGRAPH '95)*, pages 29–38. ACM.

Chen, S. E. and Williams, L. (1993). View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288.

Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, Jr., F. P., and Wright, W. V. (1996). Simplification envelopes. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 119 – 128. ACM SIGGRAPH, ACM Press.

Cyan (1994). *Myst: The Surrealistic Adventure That Will Become Your World*. Broderbund Software.

Darsa, L. and Costa, B. (1996). Multi-resolution representation and reconstruction of adaptively sampled images. In *SIBGRAPI'96 Proceedings*, pages 321–328.

Debevec, P. E., Taylor, C. J., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 11–20. ACM SIGGRAPH, ACM Press.

DeRose, T. D., Lounsbery, M., and Warren, J. (1993). Multiresolution analysis for surface of arbitrary topological type. Report 93-10-05, Department of Computer Science, University of Washington, Seattle, WA.

Gomes, J., Costa, B., Darsa, L., Velho, L., Wolberg, G., and Berton, J. (1995). *Warping and Morphing of Graphical Objects*. SIGGRAPH '95 Course Notes #3.

Gortler, S. J., Grzeszczuk, R., Szelinski, R., and Cohen, M. F. (1996). The lumigraph. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 43–54. ACM SIGGRAPH, ACM Press.

Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE CG&A*, 6(11):21–29.

Greene, N. (1996). Hierarchical polygon tiling with coverage masks. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 65 – 74. ACM Siggraph, ACM Press.

Greene, N. and Kass, M. (1993). Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240.

He, T., Hong, L., Varshney, A., and Wang, S. (1996). Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184.

Hoppe, H. (1996). Progressive meshes. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 99 – 108. ACM SIGGRAPH, ACM Press.

Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 31–42. ACM SIGGRAPH, ACM Press.

Lippman, A. (1980). Movie maps: An application of the optical videodisc to computer graphics. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, pages 32–43.

Luebke, D. and Georges, C. (1995). Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *Proceedings, 1995 Symposium on Interactive 3D Graphics*, pages 105 – 106.

McMillan, L. and Bishop, G. (1995). Plenoptic modeling: An image-based rendering system. In *Computer Graphics Annual Conference Series (SIGGRAPH '95)*, pages 39–46. ACM.

Neider, J., Davis, T., and Woo, M. (1993). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1*. Addison-Wesley.

Rossignac, J. and Borrel, P. (1993). Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag.

Schroeder, W. J., Zarge, J. A., and Lorensen, W. E. (1992). Decimation of triangle meshes. In *Computer Graphics: Proceedings SIGGRAPH '92*, volume 26, No. 2, pages 65–70. ACM SIGGRAPH.

Segal, M. (1996). Personal communication.

Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., and Haeberli, P. (1992). Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):249–252.

Szeliski, R. (1996). Video mosaics for virtual environments. *IEEE CG&A*, pages 22–30.

Teller, S. and Séquin, C. H. (1991). Visibility preprocessing for interactive walkthroughs. *Computer Graphics: Proceedings of SIGGRAPH'91*, 25, No. 4:61–69.

Turk, G. (1992). Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):55–64.

Wolberg, G. (1990). *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA.

Xia, J. and Varshney, A. (1996). Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96 Proceedings*. ACM/SIGGRAPH Press.