Walkthroughs of Complex Environments using Image-based Simplification

Lucia Darsa

16532 NE 36th Ct. #LL303, Redmond, WA 98052, luciad@msn.com

Bruno Costa

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

Amitabh Varshney

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

We present an image-based technique to accelerate the navigation in complex static environments. We perform an image-space simplification of each sample of the scene taken at a particular viewpoint and dynamically combine these simplified samples to produce images for arbitrary viewpoints. Since the scene is converted into a bounded complexity representation in the image space, with the base images rendered beforehand, the rendering speed is relatively insensitive to the complexity of the scene. The proposed method correctly simulates the kinetic depth effect (parallax), occlusion, and can resolve the missing visibility information. This paper describes a suitable representation for the samples, a specific technique for simplifying them, and different morphing methods for combining the sample information to reconstruct the scene. We use hardware texture mapping to implement the image-space warping and hardware affine transformations to compute the viewpoint-dependent warping function.

1 Introduction

In contrast to a conventional geometry-based rendering pipeline image-based renderers use pre-rendered images of the scene as the basic primitives to render a scene—in place or in conjunction with usual 3D models. This makes it possible to achieve much higher levels of realism, not only when modeling with real world photographs, but also when using synthetically generated imagery, since much more complex rendering algorithms can be used in the pre-navigation step. Image-based rendering also decouples the navigation frame rate from the complexity of the models, since only the fixed resolution images of the models are used, making this approach potentially faster than traditional ones as the model complexity continues to increase.

The trade-off of geometry complexity for images is not new. Texture mapping was introduced in 74 by Catmull [8], and has been used extensively since then. Blinn and Newell [7] used pre-rendered images to map the surrounding environment on reflective surfaces.

Image-based rendering has been used with two different but closely related aims: enable the navigation of environments modeled by real-world photographs and accelerate the navigation of synthetic environments. In the first case the scene is not modeled on the computer and parameters such as depth information are not easily available. In the second case, all the information necessary for an image-based renderer can be retrieved from the original model.

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

- reducing the rendering complexity by using texture mapping [7,6], and by using various levels of complexity in shading and illumination models [4].
- reducing the geometric complexity of the scene, by using level-of-detail hierarchies [46,37,36,11,24,17,23], and by visibility-based culling [2,44,22,29,20].
- exploiting frame-to-frame coherence with one of the above [5,49].

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-based rendering has been used to navigate (although with limited freedom of movement) in environments modeled from real-world digitized images [9,43,33]. More recent approaches [19,26] generalize the idea of plenoptic modeling by characterizing the complete flow of light in a given region of space. This can be done for densely sampled arrays of images (digitized or synthetic) without relying on depth information, resulting in a large amount of information that limits its applicability to current environments without further research. Promising results towards making these approaches feasible in real time appear in [42].

Combining simple geometric building blocks with view-dependent textures derived from image-based rendering [16,3,30] has resulted in viable techniques for navigation in environments that can be described by those simple blocks. Also, [25] derived simple 3D scene models from photographs that allowed navigation in a single image. They use a *spidery mesh* graphical user interface, enabling the user to specify a vanishing point, background and foreground objects in an existing picture easily. An interesting use of image-based rendering for distributed virtual environments has been presented in [31]. In this approach only a subset of the scene information that can not be extrapolated from the previous frame is transmitted in a compressed form from the server to the client, thereby dramatically reducing the required network bandwidth. The potential of image-based rendering specifically for the navigation in generic synthetic environments on single graphics machines however has been investigated in fewer instances [10,40,32].

We have presented a conceptual discussion and an implemented system for the problem of image-based rendering using image-space simplification and morphing [14]. In this paper, we discuss the technique in more detail, present the image-space simplification algorithm used, and compare the results of different blending techniques. 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 [10,9] through morphing.

In section 2, we present an overview of the image-based rendering area; in section 3, we discuss the relation between the morphing problem and image-based rendering. Section 4 describes the image-space simplification technique in detail, and sections 5 and 6 discuss the navigation problem, comparing various forms of node combination. Some results are presented in section 7.

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. Image-based rendering can be described as a process consisting of generally three steps:

- Sampling - samples from the scene model are obtained at discrete viewpoints and viewing directions;
- Reconstruction - samples are organized into data structures that allow evaluation through some kind of interpolation;
- Resampling - sampled data is reprojected from a new viewpoint and direction, creating new views of the scene in real time.

This process can have a feedback, as in the case of "Talisman" [45], "Hierarchical Image Caching" [40] and "Post-rendering 3D warping" [32], where the original scene is periodically resampled as the navigation occurs. In these cases, however, there won't be a complete decoupling from the 3D scene complexity, as new samples from the scene model will still have to be generated in run-time, even if at a lower frame rate.

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 three different levels based on freedom allowed in user-movement:

(i) Discrete viewpoints, discrete view directions
(ii) Discrete viewpoints, continuous view directions
(iii) Continuous viewpoints, continuous view directions

The first group is the simplest approach, providing a very limited immersive experience and interaction. The sample 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 [27], and the same concept has been used at the consumer level more recently [12].

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 [33] 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 [9], for instance.

The last group gives the user the highest degree of freedom to continuously translate and rotate. In “View Interpolation for Image Synthesis” [10], 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” [33], 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, required closely spaced nodes and user input for proper matching. As they developed an ordering algorithm that ensured pixels were drawn back to front, they avoided depth comparisons in the warping and blending steps. “Post-Rendering 3D Warping” [32] uses the same image warping algorithm, but composite from different warped reference images to obtain the final images. As more than one input image is used, they have to perform depth comparisons to resolve hidden surfaces.

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 [1]. 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 the third category 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. Our technique samples the original scene from a set of fixed viewpoints, associating a *node* with every selected position, consisting of an extended environment map with depth and color information for every direction, and also the camera parameters. This is essentially a sampling of the plenoptic function, that associates depth information to each direction, in addition to color.[1] We

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

reconstruct the depth data, simplifying the scene in image-space, generating a view dependent simplified scene.

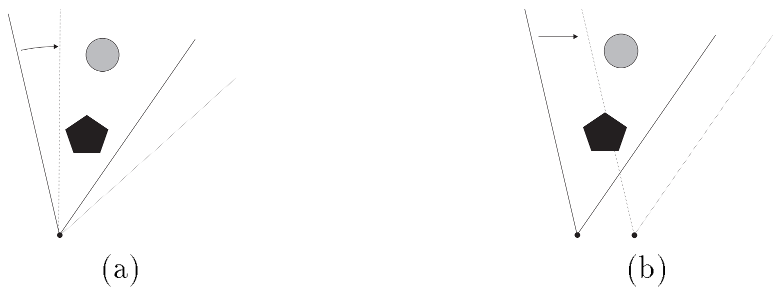


Fig. 1. Visibility: (a) Rotation (b) Translation.

Each node provides limited information about the world, which is not sufficient to determine the view from an arbitrary viewpoint and direction in the reprojection phase. 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 [18]. The particular case of image morphing is extensively discussed in [48]. 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 $I^z : U \subset \mathbf{R}^2 \to C \times \mathbf{R}$, 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:

$$R : \mathcal{S} \times \mathcal{P} \rightarrow \mathbf{I}^z, \tag{1}$$
$$\mathcal{P} = \{\mathcal{P} : \mathbf{R}^3 \rightarrow \mathbf{R}^3\}. \tag{2}$$

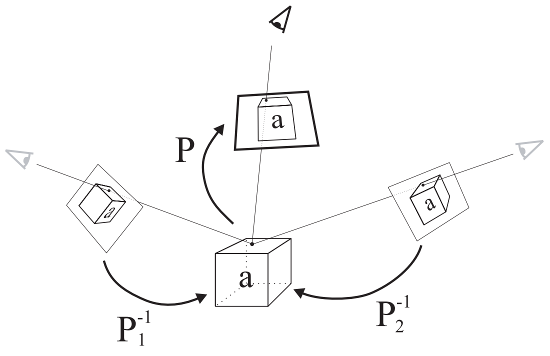


Fig. 2. Environment map morphing.

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 the left side of 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].

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.

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

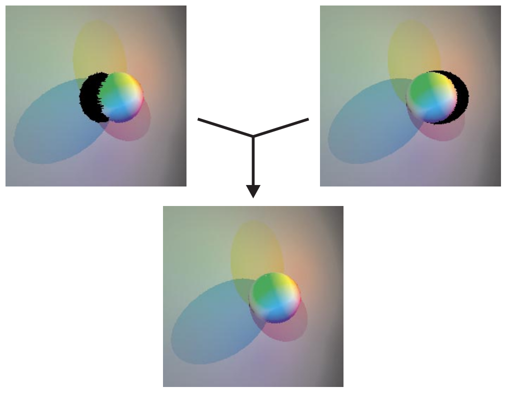


Fig. 3. Blending from two nodes (visibility gaps in black).

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 2. 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 4(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 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 [34].

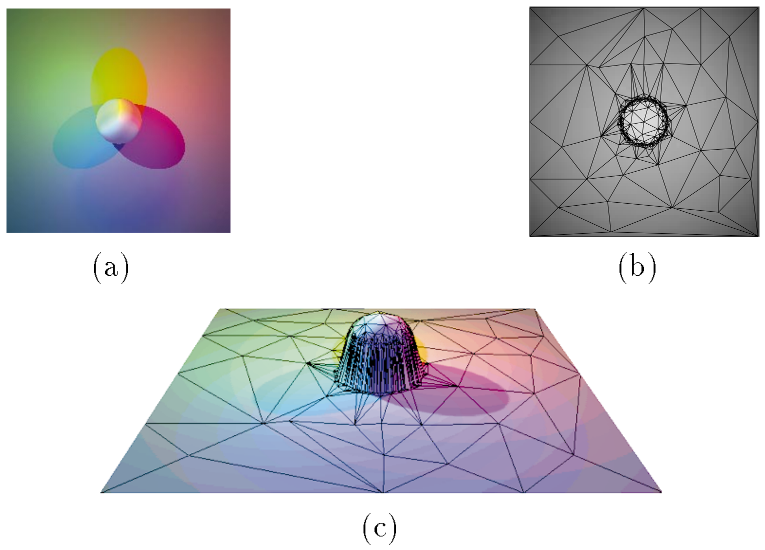


Fig. 4. Range image triangulation: (a) Input image; (b) 2D texture coordinates; (c) 3D triangulation textured by input image

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 4(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 [21] and cylindrical maps [9,33]. 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 and have discontinuities at the edges, 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 5). The construction of cubical maps directly from photographs is difficult, requiring complicated alignment and expensive 90^o degree lenses, but it can be done by stitching on an intermediate cylinder that is reprojected onto a cube. More-

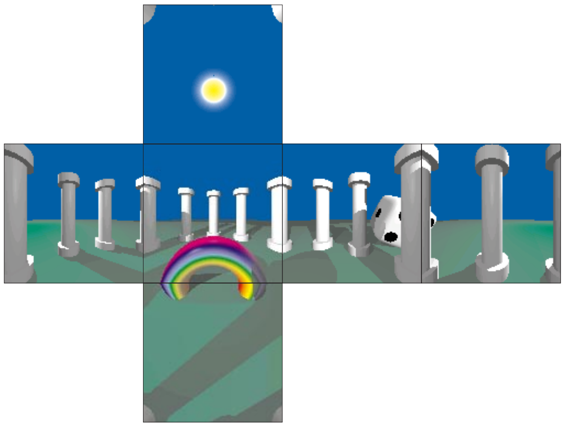


Fig. 5. Unfolded cubical environment map.

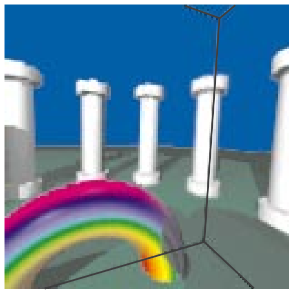


Fig. 6. Cube reprojection in a given viewing direction.

over, each of the sides of a cube 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 5 is shown for a given view direction in Figure 6; the seams of the cube are highlighted to indicate the new 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.
- to refine more finely closer objects, where the parallax effect is more noticeable.

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.

Adaptive Top-down Sampling

The implemented algorithm constructs an image-space Delaunay triangulation using a Voronoi diagram to adaptively sample the image based on the depth component. The image-based simplification is similar to previous work by the authors [13], but with a different objective.

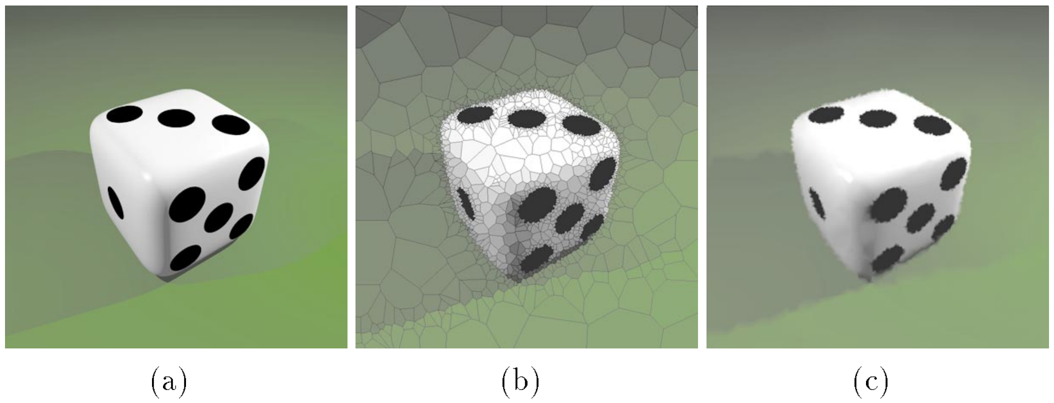


Fig. 7. (a) Conventional image; (b) Voronoi cells (2,000 samples); (c) Interpolated Delaunay triangles (2,000 samples)

In [13], the goal was to represent color images minimizing the number of samples required, by using adaptive sampling and reconstructing the images using Gouraud shading. Figure 7(a) shows a conventional representation of an image, with 640,000 samples organized in a regular grid. The adaptive form of sampling and reconstruction is illustrated in Figures 7(b) and (c), which show two different reconstruction methods for an 800×800 image sampled at 2,000 positions (0.3% of the final resolution).

In our current approach, we are interested in minimizing the number of samples in a depth image, which represents an underlying 3D scene. Each sample represents the depth of the object with absolute fidelity at that point. To be able to decide new sample positions, more complex information will be required, such as adjacency relations and distances, implying a structure for the samples. A natural geometric concept is the area of dominance of each sample, which is well captured by the Voronoi polygon of that sample, and will be the basis of our representation.

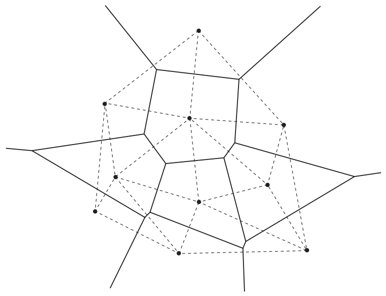


Fig. 8. Voronoi diagram and its dual Delaunay triangulation

Given a set S of n points in a plane, the set of points p, such that p is closer to one of the given points p_i than to any other p_j of S is called the *Voronoi polygon* of p_i in S (see Figure 8). In this way, each of the Voronoi polygons is the intersection of the half planes that contain p_i defined by the medians between the point p_i and each of the other points in S. The union of the Voronoi polygons of all the points in S is a planar subdivision called *Voronoi diagram* [35], such as the example in Figure 8.

There are various algorithms for the construction of Voronoi diagrams, but, in this application new samples should be added incrementally to an existing structure. The addition of a point to a Voronoi diagram is a local operation, although it may affect the entire diagram in a worst case scenario. On the average, for randomly distributed points, it can be shown that a Voronoi polygon has only six sides, or equivalently, that each vertex of a Delaunay triangulation has six incident edges [41]. A complete implementation of incremental construction of Voronoi diagrams is presented in [28].

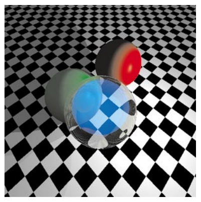
Selection of the Samples Positions

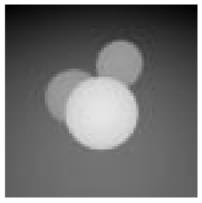
To sample adaptively, it is necessary to devise a criterion that determines where new samples may be needed. The heuristics for positioning of the samples are different for images and z-buffers, as the depth images need to be sampled based on object proximity, and the silhouettes have to be maintained for the parallax effect. We start from a basic set of samples, that can be just a few points that cover the domain of interest of the image. The information necessary to make further decisions will be based solely on the current set of samples, as we assume that no other information about the continuous depth image is available.

One way to decide where samples may be needed is a simple stack, as proposed in [15] for sampling parametric curves. The stack is initialized with the endpoints of the curve. The pair at the top of the stack is probed to test if inserting a new sample between them is actually necessary. If the two samples and the probe are reasonably collinear, the probe is discarded and that sample pair need not be considered again. Otherwise, two new segments will be pushed onto the stack: one containing the new sample and the last endpoint, the other containing the first endpoint and the new sample. Obviously, this process is not immune to aliasing, and if applied for curves with localized high frequency detail, it will fail to detect it. Note that this scheme generates the samples in the order they occur along the curve. In our case, we want to generate samples in the interest area first, which corresponds to the use a breadth first strategy, or a queue, instead of a depth first strategy.

This concept is extended to a priority queue that gives a higher preference to the more important areas of the depth image. We used a simple priority scheme that takes three factors into consideration: the difference in depth between adjacent samples, the size of the Voronoi cells, and the proximity to the observer. Between adjacent samples with high difference in depth, there is probably an object silhouette and therefore one should sample higher in that region. Large Voronoi cells need more samples too, since there is a high probability that significant detail in them is lost. We could have used an object identification information from the renderer, but we chose not to, so that fewer restrictions were imposed on the set of renderers that could be used with our system.

The above criteria can be easily evaluated by using the Voronoi diagram or the Delaunay triangulation to structure the samples. Each Voronoi edge corresponds to a pair of adjacent samples. If the priority is evaluated as samples are added, and stored at each Voronoi edge, the edges can be placed in the priority queue. The first edge of the queue is then probed repeatedly, possibly adding more edges to the queue. Once an edge to be broken is selected, the probe position will be chosen randomly between the endpoints of the edge. The rationale behind this decision is that endpoints of Voronoi edges are equidistant from the neighboring samples, yielding a better sample distribution.


(a)


(b)

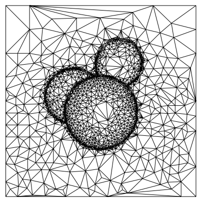

(c)

Fig. 9. Triangulation using depth and discontinuity.

Figure 9 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 10, where an observer is moving away from the spheres—10(b) shows the view from the position where the node was generated. Note how the visibility changes are correctly handled, with

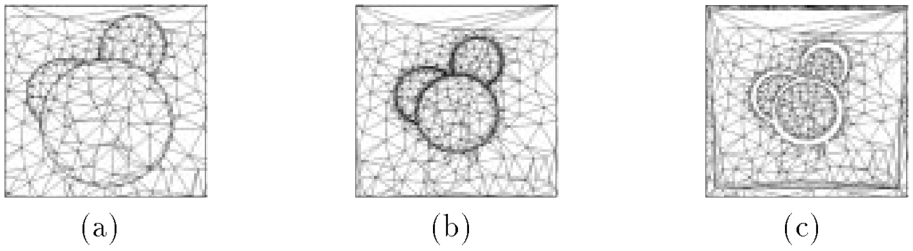


Fig. 10. Handling Visibility Changes in Translation.

the closer sphere covering a greater part of the other spheres (10(a)) and the visibility gaps appearing where no information was available (10(c)). Methods to fill-in these visibility gaps using information from neighboring nodes are presented in Section 6.

Obtaining a Triangulation Directly

When using polygonal data as input, it is possible to construct a trinagulation that matches the edges of the original objects exactly. This can be done by using an object space hidden surface algorithm such as [47] to eliminate invisible polygons, and keep the visible part of the remaining polygons. This polygon mesh, however, is more closely dependent on the complexity of the original scene. A mesh simplification algorithm, taking in consideration the depth information coherence can be used to further simplify it.

4.3 View-dependent Texture Mapping

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.

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 [39]. In our case, however, the texture maps already

Fig. 11. Handling Perspective: (a) Double Perspective; (b) Compensated.

have the necessary perspective distortion in them. Letting the hardware perform perspective correction results in an incorrect double perspective effect. Figure 11(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 [38]. Details of this transformation appear in the Appendix.

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 q. 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 average ray direction, i.e., the ray connecting the barycenter of the triangle and the observer position:

$$q = \|N.R\| = |\cos(\theta)|$$

(a) (b)

Fig. 12. Triangulation qualities from two different nodes.

This is equivalent to the ratio between the orthogonal projected area in the ray direction and the original area. Note that no triangles will have $\theta > 90^o$, as the triangulation contains only triangles that are visible from the original observer position. When θ is close to 0 degrees, the triangle is almost perpendicular to the observer, and its quality will be close to 1, meaning that its texture information is well represented. For a large angle, closer to 90 degrees, the face is projected to a small area, and its quality will be close to 0.

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 12 shows the qualities—indicated as gray levels with white being the best—of the triangles of two different nodes viewed from the same position.

It is interesting to note that the triangle qualities can be automatically generated by any renderer that uses a Lambertian diffuse illumination model. This is done by placing a point light source with no attenuation in the observer position, and re-rendering the entire scene with no ambient illumination using a white dull material for all objects. The equivalence is due to the Lambertian diffuse component being based on the cosine of the normal with the light source direction.

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 13. The navigation

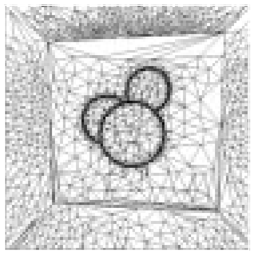


Fig. 13. Node triangulation viewed from within the node.

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 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 14, for

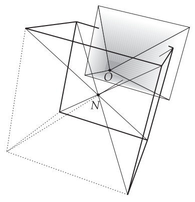


Fig. 14. View frustum culling.

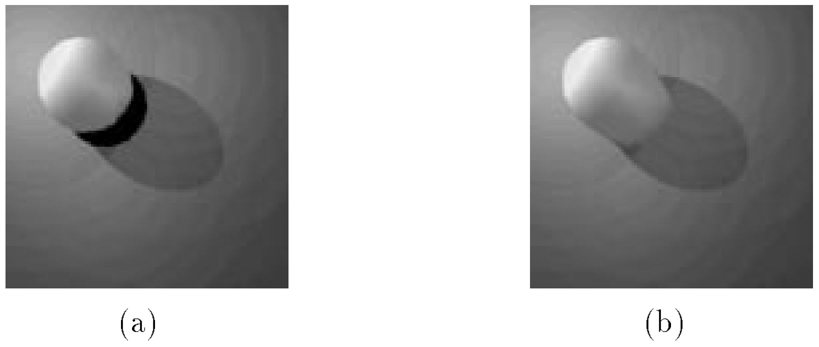


Fig. 15. Visibility gaps: (a) black; (b) filled by linear interpolation.

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 15, 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

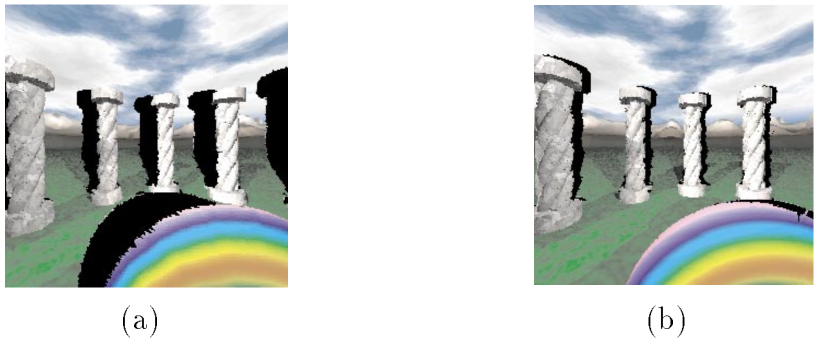

(a) (b)

Fig. 16. Views from two different nodes.

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. Figure 16 shows the visibility gaps (in black) from two different nodes that will serve as a testbed to compare the different combinations.

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

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. This approach is the only one currently implemented entirely in hardware, and uses the hardware z-buffer for hidden surface removal. The combination of the two images is then performed by comparing the z-values of the two corresponding input pixels p_1 and p_2:

$$p = \begin{cases} p_1, \; z_1 < z_2 \\ p_2, \; \text{otherwise} \end{cases}$$

Although this very 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. As the triangle qualities are not considered, parts of low quality triangles can dominate over high quality ones that are slightly farther. Notice the discontinuities, especially in the rainbow in Figure 17 generated using this method.

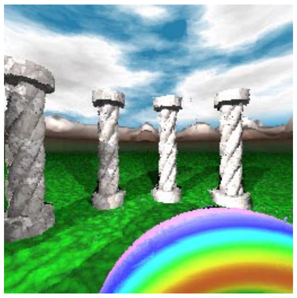


Fig. 17. Mesh Layering

6.2 Binary Merge

To allow the quality measure to play a significant role, we must be able to obtain, for each pixel, the quality of the triangle it belongs to. The resulting pixel is now computed by first comparing the z-values of the input pixels, as follows:

$$p = \begin{cases} p_1, \ z_1 < z_2 - \delta, \ \text{or} \\ \qquad 0 \leq |z_1 - z_2| \leq \delta \text{ and } q_1 \geq q_2 \\ p_2, \text{otherwise} \end{cases}$$

If the z values are sufficiently different, according to an arbitrary δ value, the pixel that is closer to the observer wins; otherwise, the pixels are two different representations for the same surface, and therefore the pixel that is output to the final image will be the one that has a higher quality.

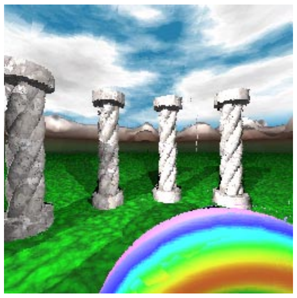


Fig. 18. Binary merge.

Although this function is not a continuous blend it yields good results. Figure 18 shows a combination using this technique (see Figure 12 for the triangle qualities). To implement this scheme, we store the quality of each triangle on a pixel-by-pixel basis in the stencil or alpha buffer during rasterization. We retrieve color and depth information from the warped images in the frame buffer, blend the values, and write the resulting image back to the frame buffer. Note that if alpha buffer is used, Gouraud shading can be used to vary the pixel-by-pixel qualities smoothly inside each triangle.

6.3 Simple Weighted Blending

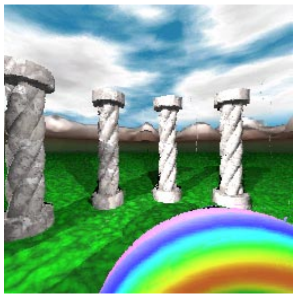


Fig. 19. Weighted blending.

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:

$$p = \begin{cases} p_1, & z_1 < z_2 - \delta \\ p_2, & z_2 < z_1 - \delta \\ \dfrac{q_1 p_1 + q_2 p_2}{q_1 + q_2}, & \text{otherwise} \end{cases}$$

The result using this computation is shown in Figure 19, 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:

$$p = \begin{cases} p_1, & z_1 < z_2 - \delta \\ p_2, & z_2 < z_1 - \delta \\ \dfrac{t q_1 p_1 + (1-t) q_2 p_2}{t q_1 + (1-t) q_2}, \quad t = \dfrac{d_2}{d_1 + d_2}, & \text{otherwise} \end{cases}$$

This solution produces a smooth morphing between the nodes (see Figure 20).

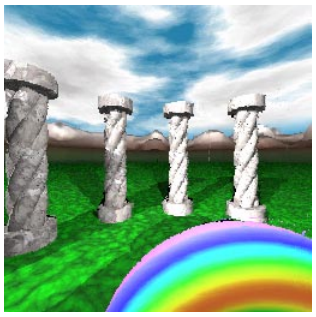


Fig. 20. Positional weighted blending.

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.

7 Results

We have tested our implementation on a model generated by us. The initial model was ray-traced and two cubical environment maps, each consisting of six 512×512 images (with depth), were generated. From these $3M$ data points, we obtained a simplified representation consisting of a total of $30K$ texture-mapped triangles using the top-down approach described before to generate a Delaunay triangulation.

We have measured the performance in two reference systems: a single SGI Challenge R10000 processor with one raster manager, Infinite Reality with 64MB of texture memory and 2MB of secondary cache; and a Pentium Pro 200MHz with a Permedia NT graphics card accelerator, with 4MB of video memory. 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. In the first system, 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. In the second system, we obtained an average frame-rate of 8.03 frames per second for mesh layering. 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.

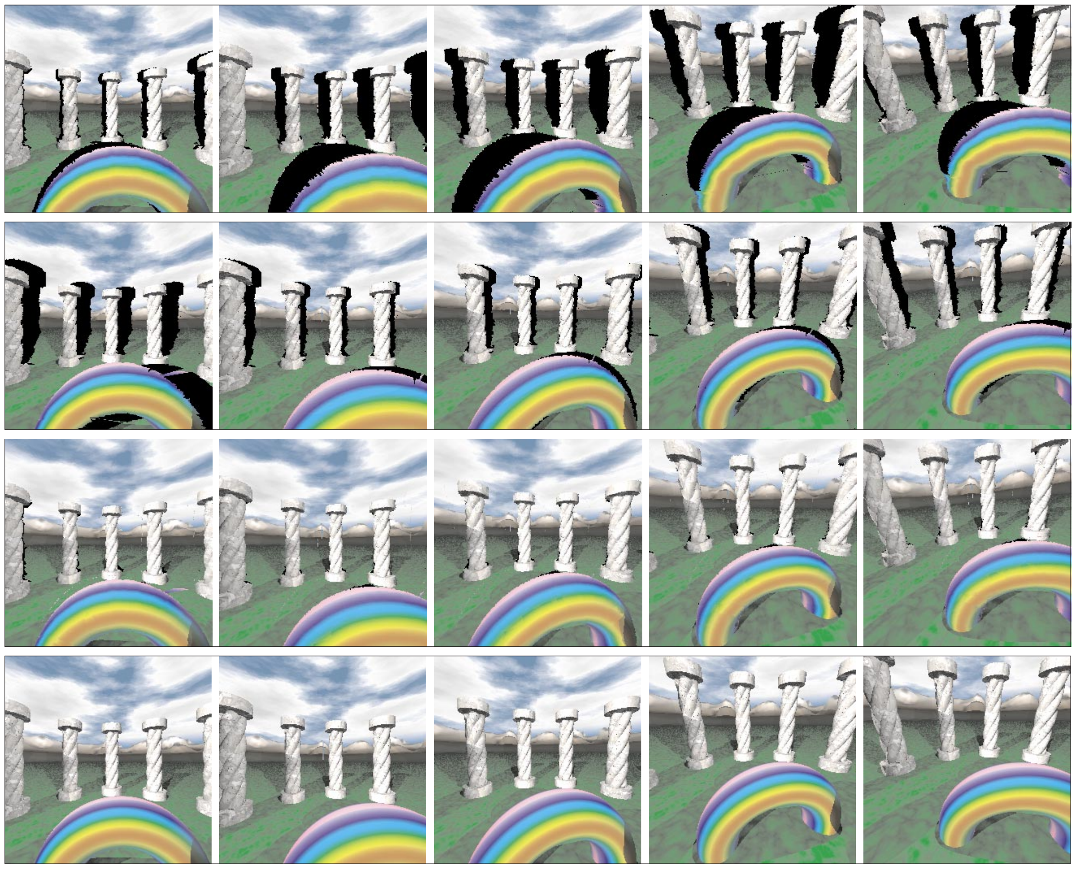


Fig. 21. Navigating in Image-Space: row 1 – node 1, row 2 – node 2, row 3 – positional Weighted Blending of Node 1 and Node 2, row 4 – ray traced frames.

We have compared the results of the different blending methods presented, using a squared distance error measurement for a reference sequence depicted in the bottom row of Figure 21. The error grows quadratically from 0, for identical images, to 1, for entirely different images. Note that the measurements were performed in the RGB space, which is not linear perceptually.

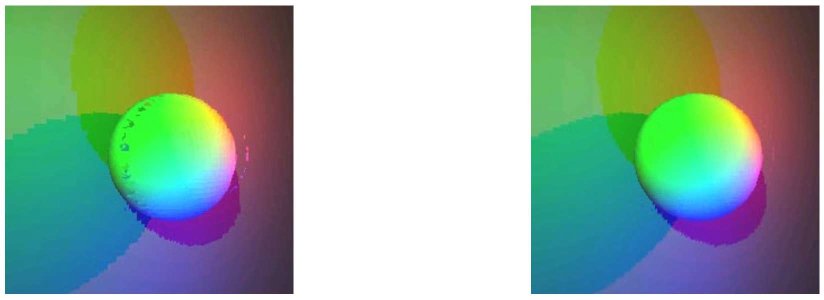


Fig. 22. Mesh layering (a) versus Positional weighted blending (b).

Each reference frame was rendered using ray tracing from the 3D models, and compared to the image-based rendering technique using the different blending methods. A comparison chart indicating the error for each different view is

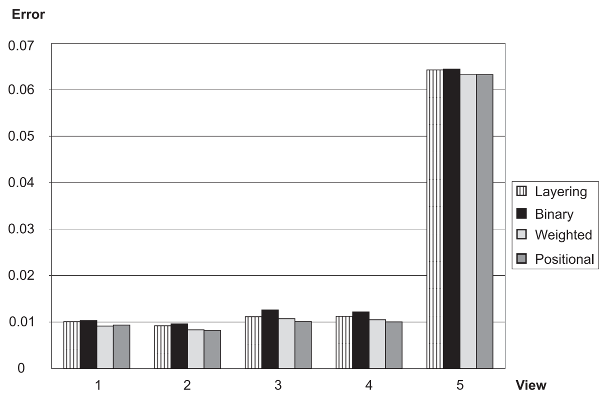


Fig. 23. Blending techniques comparison for each different view

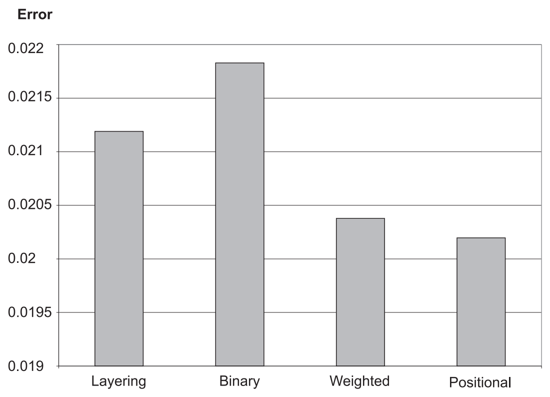


Fig. 24. Average error

shown in Figure 23. The average error for each blending technique is shown in Figure 24. The results indicate that, as expected, the error increases as the observer moves away from the node centers. Also, the positional weighted blending produces consistently the best results.

Other results are shown in Figures 18, 19, 20, and 21. 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 22 in the spurious triangles near the left side of the sphere in the mesh layering approach.

Figure 21 shows frames of an animation obtained by blending two nodes using the positional 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 near the center of the first node, translates to the left, then rotates to the right and to the bottom.

8 Conclusion

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.

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 frustum to be restricted to the bounding box of the visibility gaps. A similar approach has been shown to work well [29].

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.

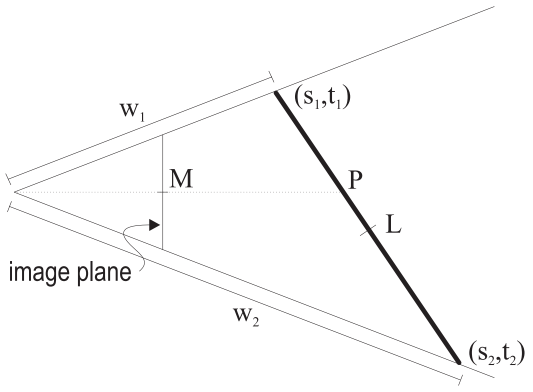


Fig. A.1. Linear and perspective corrected interpolations.

Acknowledgement

We would like to thank CNPq (Brazilian Council of Scientific and Technological Development) for the financial support to the 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 A.1. 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

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

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.

References

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

[2] J. M. Airey. *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, 1990.

[3] D. G. Aliaga. Visualization of complex models using dynamic texture-based simplification. In *IEEE Visualization '96 Proceedings*, pages 101–106. ACM/SIGGRAPH Press, October 1996.

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

[5] Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen Zagier. 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, July 1994.

[6] Jim F. Blinn. Simulation of wrinkled surfaces. In *SIGGRAPH '78*, pages 286–292. ACM, 1978.

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

[8] Ed Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, 1974.

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

[10] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, August 1993.

[11] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. V. Wright. 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, August 1996.

[12] Cyan. *Myst: The Surrealistic Adventure That Will Become Your World.* Broderbund Software, 1994.

[13] Lucia Darsa and Bruno Costa. Multi-resolution representation and reconstruction of adaptively sampled images. In *SIBGRAPI'96 Proceedings*, pages 321–328, October 1996.

[14] Lucia Darsa, Bruno Costa, and Amitabh Varshney. Navigating static environments using image-space simplification and morphing. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 25–34. ACM SIGGRAPH, ACM Press, April 1997.

[15] Luiz Henrique de Figueiredo. Adaptive sampling of parametric curves. In Alan Paeth, editor, *Graphics Gems V*, pages 173–178. AP Professional, San Diego, CA, 1995.

[16] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. 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, August 1996.

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

[18] Jonas Gomes, Bruno Costa, Lucia Darsa, Luiz Velho, George Wolberg, and John Berton. *Warping and Morphing of Graphical Objects.* SIGGRAPH '95 Course Notes #3, 1995.

[19] Steven J. Gortler, Radek Grzeszczuk, Richard Szelinski, and Michael F. Cohen. 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, August 1996.

[20] N. Greene. 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, August 1996.

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

[22] Ned Greene and M. Kass. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993.

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

[24] H. Hoppe. 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, August 1996.

[25] Youichi Horry, Ken ichi Anjyo, and Kiyoshi Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA, August 3–8, 1997)*, Computer Graphics Proceedings, Annual Conference Series, pages 225–232. ACM SIGGRAPH, ACM Press, August 1997.

[26] Marc Levoy and Pat Hanrahan. 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, August 1996.

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

[28] Dani Lischinski. Incremental delaunay triangulation. In Paul S. Heckbert, editor, *Graphics Gems IV*, pages 47–59. AP Professional, San Diego, CA, 1994.

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

[30] P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 Symposium on Interactive 3D Computer Graphics*, pages 95–102, 1995.

[31] Y. Mann and D. Cohen-Or. Selective pixel transmission for navigating in remote virtual environments. In *Proceedings EUROGRAPHICS '97*, pages C–201 – C–206, 1997.

[32] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, ACM Press, April 1997.

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

[34] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1.* Addison-Wesley, 1993.

[35] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

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

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

[38] Mark Segal. Personal communication, 1996.

[39] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):249–252, July 1992.

[40] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 75–82. ACM SIGGRAPH, ACM Press, August 1996.

[41] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, August 1978.

[42] P. Sloan, M. F. Cohen, and S. Gortler. Time critical lumigraph rendering. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 17–23. ACM SIGGRAPH, ACM Press, April 1997.

[43] Richard Szeliski. Video mosaics for virtual environments. *IEEE CG&A*, pages 22–30, March 1996.

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

[45] Jay Torborg and James T. Kajiya. Talisman: Commodity realtime 3d graphics for the pc. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 353–364. ACM SIGGRAPH, ACM Press, August 1996.

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

[47] K. Weiler. Polygon comparison using a graph representation. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, pages 10–18, 1980.

[48] G. Wolberg. *Digital Image Warping.* IEEE Computer Society Press, Los Alamitos, CA, 1990.

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