View Interpolation for Image Synthesis

Shenchang Eric Chen, Lance Williams

Apple Computer, Inc.

ABSTRACT

Image-space simplifications have been used to accelerate the calculation of computer graphic images since the dawn of visual simulation. Texture mapping has been used to provide a means by which images may themselves be used as display primitives. The work reported by this paper endeavors to carry this concept to its logical extreme by using interpolated images to portray three-dimensional scenes. The special-effects technique of morphing, which combines interpolation of texture maps and their shape, is applied to computing arbitrary intermediate frames from an array of prestored images. If the images are a structured set of views of a 3D object or scene, intermediate frames derived by morphing can be used to approximate intermediate 3D transformations of the object or scene. Using the view interpolation approach to synthesize 3D scenes has two main advantages. First, the 3D representation of the scene may be replaced with images. Second, the image synthesis time is independent of the scene complexity. The correspondence between images, required for the morphing method, can be predetermined automatically using the range data associated with the images. The method is further accelerated by a quadtree decomposition and a view-independent visible priority. Our experiments have shown that the morphing can be performed at interactive rates on today's high-end personal computers. Potential applications of the method include virtual holograms, a walkthrough in a virtual environment, image-based primitives and incremental rendering. The method also can be used to greatly accelerate the computation of motion blur and soft shadows cast by area light sources.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Additional Keywords: image morphing, interpolation, virtual reality, motion blur, shadow, incremental rendering, real-time display, virtual holography, motion compensation.

1 INTRODUCTION

Generating a large number of images of an environment from closely spaced viewpoints is a very useful capability. A traditional application is a flight in the cabin of an aircraft simulator, whereas the contemporary model is perhaps a walk through a virtual environment; in both cases the same scene is displayed from the view of a virtual camera controlled by the user. The computation of global illumination effects, such as shadows, diffuse and specular inter-reflections, also requires a large number of visibility calculations. A typical approach to this problem is to rely on the computer to repetitively render the scene from different viewpoints. This approach has two major drawbacks. First, real-time rendering of complex scenes is computationally expensive and usually requires specialized graphics hardware. Second, the rendering time is usually not constant and is dependent on the scene complexity. This problem is particularly critical in simulation and virtual reality applications because of the demand for real-time feedback. Since scene complexity is potentially unbounded, the second problem will always exist regardless of the processing power of the computer.

A number of approaches have been proposed to address this problem. Most of these approaches use a preprocess to compute a subset of the scene visible from a specified viewing region[AIRE91, TELL92]. Only the potentially visible objects are processed in the walkthrough time. This approach does not completely solve the problem because there may be viewing regions from which all objects are visible. Greene and Kass[GREE93] developed a method to approximate the visibility at a location from adjacent environment maps. The environment maps are Z-buffered images rendered from a set of discrete viewpoints in 3D space. Each environment map shows a complete view of the scene from a point. An environment map can take the form of a cubic map, computed by rendering a cube of 90˚ views radiating from that point [GREE86]. The environment maps are pre-computed and stored with viewpoints arranged in a structured way, such as a 3D lattice. An image from a new viewpoint can be generated by re-sampling the environment maps stored in adjacent locations. The re-sampling process involves rendering the pixels in the environment maps as 3D polygons from the new viewpoint. The advantage of this approach is that the rendering time is proportional to the environment map resolutions and is independent of the scene complexity. However, this method requires Z-buffer hardware to render a relatively large number of polygons interactively, a feature still not available on most low-end computers.

This paper presents a fast method for generating intermediate images from images stored at nearby viewpoints. The method has advantages similar to those of Greene and Kass' method. The generation of a new image is independent of the scene complexity. However, instead of drawing every pixel as a 3D polygon, our method uses techniques similar to those used in image morphing[BEIE92]. Adjacent images are "morphed" to create a new image for an in-between viewpoint. The morphing makes use of pre-computed correspondence maps and, therefore, is very efficient. Our experiments with the new method have shown that it can be performed at interactive rates on inexpen-

sive personal computers without specialized hardware.

The new method is based on the observation that a sequence of images from closely spaced viewpoints is highly coherent. Most of the adjacent images in the sequence depict the same objects from slightly different viewpoints. Our method uses the camera's position and orientation and the range data of the images to determine a pixel-by-pixel correspondence between images automatically. The pairwise correspondence between two successive images can be pre-computed and stored as a pair of morph maps. Using these maps, corresponding pixels are interpolated interactively under the user's control to create in-between images.

Pixel correspondence can be established if range data and the camera transformation are available. For synthetic images, range data and the camera transformation are easily obtainable. For natural images, range data can be acquired from a ranging camera [BESL88], computed by photogrammetry [WOLF83], or modeled by a human artist [WILL90]. The camera transformation can be found if the relative positions and orientations of the camera are known.

The idea of using images to represent a virtual environment has been presented previously. An earlier approach uses computer controlled videodiscs to perform surrogate travel [LIPP80]. A more recent approach uses digital movie technologies to construct a virtual museum [MILL92]. In both systems, a user navigates a finite set of routes and directions that have been pre-determined. Our method allows greater flexibility in the navigation because the stored frames can be interpolated smoothly to synthesize arbitrary intermediate points of view.

A static subject or environment portrayed by a restricted set of images indexed by the user's point of view supports a form of "desktop virtual reality" termed "virtual integral holography" [VENO90]. In this context also, our method permits smooth interpolation of the images to present a continuous display sequence, rather than quantizing the user's point of view and jumping to the closest prestored image.

The morphing method can be used to interpolate a number of different parameters, such as camera position, viewing angle, direction of view and hierarchical object transformation. The modeling and viewing transformations can be concatenated to compute the correspondence mapping between two images. Generally, the images can be arranged in an arbitrary graph structure. The nodes of the graph are the images. Each arc in the graph represents a correspondence mapping, which is bi-directional, and two maps are associated with each arc. The number of interpolation parameters determines the dimensionality of the graph. For instance, the graph for a virtual camera moving with two degrees of freedom (the latitudes and longitudes of a sphere bounding an object at a central "look-at" point, for example) is a simple polyhedron (rendering of objects rather than environments will be discussed in more detail in Section 4.4, Image-based Primitives.) The camera's location coordinates index a point on a face of the polyhedron, and the desired view is synthesized by interpolating the images and mappings stored with the vertices and edges of the face. Note that if each image is of the form of an environment map, view angle and direction also can be interpolated by re-projecting the environment map to the desired view orientation [MILL93] without increasing the dimensionality of the graph. Similarly, a camera moving in 3D is supported by a graph which takes the form of a 3D space lattice. The barycentric coordinates of the view location can be used to interpolate among the images attached to the vertices of the enclosing tetrahedron in a lattice of tetrahedra.

For the representation of scenes with objects moving or changes other than those consequent to a change in viewpoint, the graph becomes a general polytope. Generally, arbitrary distortions of surfaces are accommodated by the mapping, as are hierarchical motions of linkages or the limbs of animated characters[1]. To index such an elaborate set of mappings by the various parameters can be an arbitrarily complex process, requiring multivariate interpolation of a multidimensional graph.

Without loss of generality, this paper will concentrate on the interpolation of the camera position in 1D and 2D space (accommodating "virtual holograms" of objects as well as restricted navigation in 3D scenes). The scene is assumed to be static, and all the image changes are as a result of camera movement. Although the method can be applied to natural images, only synthetic ones have been attempted in the work described here. Interpolation of images accurately supports only view-independent shading. Reflection mapping or Phong specular reflection could be performed with separate maps for reflection map coordinates or normal components, but only diffuse reflection and texture mapping have been presented here.

Section 2 introduces the basic algorithms of the method as well as its limitations and optimizations. Section 3 gives implementation details and shows some examples. Section 4 shows applications of the method to virtual reality, temporal anti-aliasing, generating shadows from area lights, image-based display primitives and incremental rendering ("progressive refinement"). Conclusions and future directions are discussed in the last section.

2 VISIBILITY MORPHING

Image morphing is the simultaneous interpolation of shape and texture. The technique generally involves two steps. The first step establishes the correspondence between two images and is the most difficult part of most morphing methods. The correspondence is usually established by a human animator. The user might, for example, define a set of corresponding points or line segments within a pair or set of images. An algorithm is then employed to determine the correspondence (mapping) for the remainder of the images[BEIE92]. The second step in the process is to use the mapping to interpolate the shape of each image toward the other, according to the particular intermediate image to be synthesized, and to blend the pixel values of the two warped images by the same respective coefficients, completing the morph.

Our method uses the camera transformation and image range data to automatically determine the correspondence between two or more images. The correspondence is in the form of a "forward mapping." The mapping describes the pixel-by-pixel correspondence from the source to the destination image. The mapping is also bi-directional since each of the two images can act as the source and the destination. In the basic method, the corresponding pixels' 3D screen coordinates are interpolated and the pixels from the source image are moved to their interpolated locations to create an interpolated image. For pixels which map to the same pixel in the interpolated image, their Z-coordinates are compared to resolve visibility. Cross-dissolving the overlapping pixels' colors may be necessary if the image colors are not view-independent. This process is repeated for each of the source images.

This method is made more efficient by the following two properties. First, since neighboring pixels tend to move together in the mapping, a quadtree block compression is employed to exploit this coherence. Adjacent pixels which move in a similar manner are grouped in blocks and moved at the same time. This compression is particularly advantageous since a view-independent visible priority among the pixel blocks can be established. The pixel blocks are sorted once by their Z-co-

[1]Establishing such elaborate mappings is straightforward for synthetic images, a classic vision problem for natural ones.

ordinates, when the maps are created, and subsequently displayed from back to front to eliminate the overhead of a Z-buffer for visibility determination.

We will describe our method in terms of the morphing between two images first. Generalization of the method to more images is straightforward and will be discussed later.

2.1 Establishing Pixel Correspondence

As a camera moves, objects in its field of view move in the opposite direction. The speed of each object's apparent movement is dependent on the object's location relative to the camera. Since each pixel's screen coordinates (x, y and z) and the camera's relative location are known, a 4x4 matrix transformation establishes a correspondence between the pixels in each pair of images. The transformations can be pre-computed and reduced to a 3D spatial offset vector for each of the pixels. The offset vector indicates the amount each of the pixels moves in its screen space as a result of the camera's movement. The offset vectors are stored in a "morph map," which represents the forward mapping from one image to another. This map is similar in concept to a disparity map computed from a stereo pair[GOSH89], the field of offset vectors computed for "optical flow" analysis[NAGE86], or motion compensation in video compression and format conversion[MPEG90]. For a computed image or range image, an exact pixel-by-pixel map can be created. The mapping is many-to-one because many pixels from the first image may move to the same pixel in the second image. Therefore, the morph map is directional and two morph maps are needed for a pair of images.

The use of a pre-computed spatial look-up table for image warping has been presented in [WOLB89]. Wolberg used the look-up table to implement arbitrary forward mapping functions for image warping. Wolberg's maps contained absolute coordinates rather than offset vectors.

In a typical image morph, as described in the beginning of this section, a sparse correspondence provided by a human operator is used to perform strictly two-dimensional shape interpolation. Such a morph can also be used to interpolate stored images in order to represent 3D scenes or objects, as suggested in [POGG91]. The advantages of our method are that the correspondence is dense (every pixel has an explicitly computed map coordinate), the correspondence is automatic (rather than relying on human effort), and the explicit prestored maps permit the image deformations to be generated very quickly.

2.2 Interpolating Correspondences

To generate an in-between view of a pair of images, the offset vectors are interpolated linearly and the pixels in the source image are moved by the interpolated vector to their destinations. Figure 1 shows the offset vectors, sampled at twenty-pixel intervals, for the camera motion sequence in Figure 3.

The interpolation is an approximation to the transformation of the pixel coordinates by a perspective viewing matrix. A method which approximates the perspective changes with local frame shifting and scaling is presented in [HOFM88]. Perspective transformation requires multiplication of the pixel coordinates by a 4x4 matrix and division by the homogeneous coordinates, a rather computationally taxing process, although bounded by image resolution rather than scene complexity. Linear interpolation of pixel coordinates using the morph maps, on the other hand, is very efficient and can be performed incrementally using forward differencing.

If the viewpoint offset is small, the interpolation is very close to the exact solution. Moreover, quadratic or cubic interpolation, though slightly more expensive to perform, can be used to improve the accuracy of the approximation. When the viewpoint moves parallel to the viewing plane, the linear interpolation produces an exact solution. This case is demonstrated in Figure 2a, which traces the paths of mapped pixels in the interpolated image as the viewpoint traverses the four corners of a square parallel to the viewing plane. The squares in the figure are the extents of the pixel movement. Because the squares are parallel to the viewing plane, the linear interpolation of the square corners produces the same result as perspective transformation. Another special case is when the viewpoint moves perpendicular to the viewing plane along a square parallel to the ground(Figure 2b). The resulting pixel locations form trapezoids, which are the projections of squares parallel to the ground. The trapezoids can be interpolated linearly in the horizontal direction. The vertical direction requires perspective divisions. The divisions can be avoided if a look-up table indexed by the vertical offset is pre-computed for each possible integer height of the trapezoids. The second case can be generalized to include the case when the squares are perpendicular to both the ground and the viewing plane. If the viewpoints are aligned with a 3D lattice, the result will always fall into one of the above two cases, which allows us to use linear interpolation to generate an exact solution.

2.3 Compositing Images

The key problem with forward mapping is that overlaps and holes may occur in the interpolated image.

2.3.1 Overlaps

One reason overlaps occur is due to local image contraction. Local image contraction occurs when several samples in a local neighborhood of the source image move to the same pixel in the interpolated image. A typical example of this case is when our view of a plane moves from perpendicular to oblique. Perspective projection causes the image to contract as the plane moves away from the point of view. In the mapping, the samples on the far side of the plane contract while the samples on the near side expand. Contraction causes the samples to overlap in the target pixels.

Multiple layers of pixel depths also will cause the samples to overlap, as in the case of the foreground sculpture in Figure 3. Resolving this case is really a hidden surface problem. One way of solving this problem is to use the Z-buffer algorithm to determine the frontmost pixel. A more efficient way of determining the nearest pixel is presented in the Optimization Section.

2.3.2 Holes

Holes between samples in the interpolated image may arise from local image expansion when mapping the source image to the destination image. This case is shown in Figure 3 where a source image is viewed from viewpoints rotated to the right. The cyan regions indicate holes. Generally, a square pixel in the source image will map to a quadrilateral in the destination image. If we interpolate the four corners of the square instead of the pixel's center, the holes can be eliminated by filling and filtering the pixels in the destination quadrilateral.

A more efficient, though less accurate, method to fill the holes is to interpolate the adjacent pixels' colors or offset vectors. The holes are identified by filling the interpolated image with a reserved "background" color first. For those pixels which still retain the background color after the source to target mapping, new colors are computed by interpolating the colors of adjacent non-background pixels. Alternatively, we can interpolate the offset vectors of the adjacent pixels. The interpolated offset is used to index back to the source image to obtain the new sample color. Note that using a distinguished background color may not identify all the holes. Some of the holes may be created by a foreground object and are filled by a background object behind it (e.g., the holes in the sculpture in the rightmost image in Figure 3). This problem is alleviated,

though not completely eliminated, when more source images are added as described below (e.g. Figure 5d).

Holes may also arise from sample locations invisible in each of the source images but visible in the interpolated image. The hole region, as shown in Figure 4, is the intersection of the umbra regions cast by viewpoints A and B and the visible region from point M. The small circle in the hole region is completely missed by the two source images from points A and B. One way of solving this problem is to use multiple source images to minimize the umbra region. Figure 5a shows the holes (cyan pixels) created by rotating one source image. Figure 5b shows that the number of holes is significantly less when two sources images are used. The number of holes can be reduced further if we place the two source viewpoints closer (Figure 5c). The remaining holes can be filled by interpolating the adjacent pixels(Figure 5d). If the images are computer-generated, a ray-tracing type of rendering can be used to render only those missing pixels.

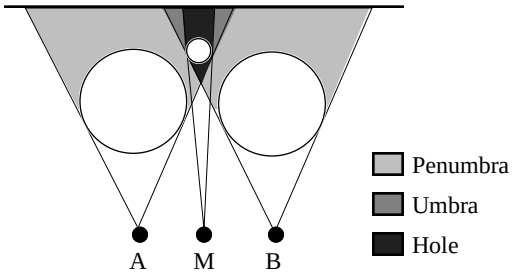

Fig. 4 Penumbra, umbra and hole regions

2.4 Optimization

The basic method is made more efficient by the following two steps.

2.4.1 Block Compression

Since adjacent pixels tend to move together in the mapping, a block compression scheme such as a quadtree can be applied to compress the morph map. The compression serves two purposes. First, it reduces the size of the morph map. Second, it allows us to interpolate offsets for entire blocks instead of pixel-by-pixel. The second aspect greatly accelerates the interpolation process as the main cost in the process is the interpolation of the offset vectors.

The compression ratio is related to the image depth complexity and the viewpoint movement. For images with high depth complexity, the compression ratio is usually low. The ratio is also lower if the viewpoint's movement results in greater pixel depth change. Figure 6 shows the quadtree decomposition of the morph map for the image sequence in Figure 3. The maximal offset threshold within a block is one pixel in Figure 6a and two pixels in Figure 6c, which means the offset vector coordinates within a block do not differ more than one or two pixel units. The compression ratio in Figure 6a is 15 to 1 and in Figure 6b is 29 to 1 (i.e., the number of blocks vs. the number of pixels).

The threshold provides a smooth quality degradation path for increased performance. Large threshold factors result in fewer quadtree blocks and, therefore, reduce the interpolation time. The performance gain is at the expense of increasing blockiness in the interpolated image. The interpolation times in Figure 6b and 6d are accelerated by a factor of 6 and 7 respectively. Note that the speedup factor does not grow linearly with the compression ratio because the same number of pixels still need to be moved.

2.4.2 View-Independent Visible Priority

In the basic method, the Z-buffer algorithm is used to resolve visibility. However, as shown in Figure 7, the A-closer-than-B priority established in View1 is still valid in View2, since Point A and Point B do not overlap in View2. The priority is incorrect in View3 when A and B overlap. As long as the angle θ in the figure is less than 90 degrees, the A-B priority does not need to be changed when the viewpoint is moved. This observation allows us to establish a view-independent visible priority for every source pixel for a viewing range. The pixels are ordered from back to front based on their original Z-coordinates when the morph maps are created, and are subsequently drawn in a back-to-front order in the interpolation process. This ordering of the samples, or sample blocks, eliminates the need for interpolating the Z-coordinates of every pixel and updating a Z-buffer in the interpolation process.

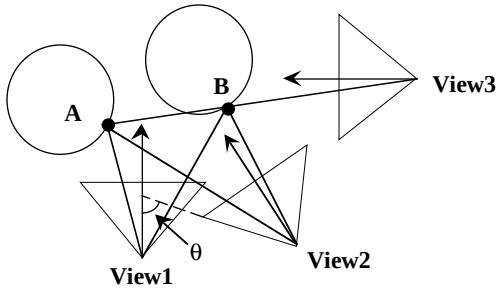

Fig. 7 View-independent visible priority

Note that the priority established here is for image pixels rather than for the underlying objects, unlike list-priority algorithms for hidden-surface removal[SCHU69].

This method applies to multiple source images as well. The source images' pixel Z-coordinates are transformed to a single coordinate system for establishing the Z-priority. All the pixels in the source images are sorted into the same priority list.

The priority can be assigned to every quadtree pixel block. With static objects and a moving camera, pixel offsets are directly related to Z-coordinates. Since the pixels within a block have similar offsets, they also have similar Z-coordinates. The Z-coordinates within a block are filtered to determine a Z value for the priority sort. The result is a sorted list of pixel blocks valid for the entire range between views.

3 IMPLEMENTATIONS

The method presented above can be summarized as follows.

3.1 Preprocessing

The preprocessing stage establishes the correspondence between each pair of source and destination images. As mentioned in Section 1, the source images are connected to form a graph structure. Each node of the graph contains a source image, its range data and camera parameters (i.e., camera's position, orientation). For each set of adjacent nodes in the graph, a sorted list of quadtree blocks is created (e.g., a block list is created for every triangle in a 2D lattice structure). Each block in the list contains a pointer to a pixel block in a source image, the size, the screen coordinates and the offset vectors of the block. The block list is created in the following steps:

Step 1. Get input data: a source node (image, range data and camera parameters), a destination node (only the camera parameters are needed) and a threshold factor for the quadtree decomposition.

Step 2. Create a morph map from the source to the destination (Section 2.1).

Step 3. Decompose the morph map into quadtree blocks and add the blocks to a block list (Section 2.4.1).

Step 4. Repeat Step 1 to 3 for each directional arc connecting the set of nodes.

5. Sort the block list from back to front by the blocks' Z-coordinates.

3.2 Interactive Interpolation

In the interactive interpolation stage, the block list corresponding to a new viewing location is retrieved. The parametric coordinates of the location with respect to the adjacent nodes are used as interpolation parameters. An interpolated image for the new location is generated in the following steps:

Step 1. Get input data: interpolation parameters and a sorted block list.

Step 2. Fill the interpolated image with a distinguished background color.

Step 3. For every block in the list in back-to-front order, compute its new location from the offset vectors and the interpolation parameters. Copy the pixel block from the source image to its new location in the interpolated image (Section 2.2).

Step 4. For every pixel in the interpolated image that still retains the background color, compute its color by filtering the colors of the adjacent non-background pixels (Section 2.3.2).

3.3 Examples

Figure 8 shows a sequence of images generated by moving the viewpoint to the right. The images were rendered at 256x256 resolution using progressive radiosity [COHE88] from a model created for the Virtual Museum project[MILL92].

Figure 9 shows two intermediate images created by morphing the leftmost and rightmost images. Each image took 0.17 second to generate (excluding the preprocessing time) on a Macintosh Quadra 950.

Note that for the interpolation to work properly, the source image cannot be anti-aliased. Anti-aliasing is view-dependent. It blends silhouette pixel colors from a particular viewpoint. Since the Z-buffer cannot be anti-aliased in the same way, the anti-aliased silhouette pixels may attach to either the foreground or the background objects depending on the quantization of the Z-buffer. This problem can be solved by morphing high-resolution unfiltered source images and then filtering the interpolated image.

The method can be applied to interpolating more than two source images. Figure 10 shows a sequence of images interpolated from the four source images in the corners. The viewpoints of the source images form a square parallel to the viewing plane. Therefore, as discussed before, linear interpolation is an exact solution to the perspective transformation. New images are computed from the nearest three corner images. The barycentric coordinates of the new viewpoint are used to interpolate the three images. Dividing the lattice into simplices minimizes the cost of interpolation.

4 APPLICATIONS

The morphing method can be used in a wide variety of applications which require fast visibility computations of a predefined static scene. Simulation and virtual reality applications typically require a scene to be displayed interactively from different viewpoints. Temporal anti-aliasing, or motion blur, can be accelerated by using morph maps to integrate image samples over time. The image samples are interpolated from key images using the morphing method. We also present an application of morph mapping to compute shadows from area lights using the shadow buffer method [WILL78]. The morphing method makes it possible to define a new class of graphic display primitives based on images. This approach is also useful in incremental rendering as it provides a way to reuse the pixels computed for previous images.

4.1 Virtual Reality

Instead of representing a virtual environment as a list of 3D geometric entities, the morphing method uses images (environment maps). To perform a walkthrough, the images adjacent to the viewpoint are interpolated to create the desired view.

In addition to supporting walkthroughs in virtual environments, the method can be used to create virtual holograms, where the display on the screen will change with respect to the user's viewpoint to provide 3D motion parallax. One existing approach uses 3D rendering to display the scene from the viewpoint obtained by a head location sensor[DEER92]. Another approach uses a finite set of pre-rendered frames, each corresponding to a particular viewing location[VENO90]. With the morphing method, only a few key images are required. The interpolation can generate the in-between frames. Figure 10 shows a sequence of images with vertical and horizontal motion parallax.

The image-based morphing method is inexpensive computationally and provides a smooth quality-speed tradeoff. Although the total storage requirement may be large, the amount of data needed to compute a frame is relatively small and can be read from secondary storage as needed. This approach is very appropriate for CD-ROM based devices because of their large storage capability. As the complexity of geometrical models increases, the advantage of image-based approaches will be more significant because of their bounded overhead.

Another advantage of using the image-based approach is that a real environment can be digitized by photographic means. Using a camera to capture the environment usually is much easier than modeling it geometrically. Although our method relies on range data to establish the correspondence between images, range data should be easier to obtain than the complete 3D geometry of the environment.

4.2 Motion Blur

If an image in a motion sequence is a sample at an instant of time instead of over a time interval, the motion will appear to be jerky and the image is said to be aliased in the temporal domain. One way to perform temporal anti-aliasing is super-sampling. The motion is sampled at a higher rate in the temporal domain and then the samples are filtered to the displayed rate. Super-sampling requires the computation of many more samples. For images which are expensive to render, this technique is very inefficient.

The morphing method allows additional temporal samples to be created by interpolation. The interpolation time is constant regardless of the rendering time for each frame. The sampling rate is determined by the largest offset vector from the morph map in order to perform proper anti-aliasing. Figure 11a is a motion blurred image computed from 32 source images for the camera motion in Figure 8. The images were first rendered at 512x512 resolution and then filtered down to 256x256 resolution before temporal anti-aliasing was performed. The temporal samples were anti-aliased with a box filter. Each image took around 5 seconds to render on a high-end workstation with 3D graphics hardware support. Figure 11b was computed from the same number of images interpolated from three of the source images. Each interpolated image took 0.6 second to compute on a Macintosh Quadra950. The only minor visible difference between the two images is the top of the inside loop of the foreground sculpture, due to the holes created from the interpolation as discussed previously.

The super-sampling approach requires the sampling rate to be determined based on the worst case. For images with fast

moving objects and slowly moving backgrounds, this method is not very efficient. One way to solve this problem is to segment the images based on object movement and use different sampling rates for each segment. For instance, the foreground sculpture in this figure needs to be sampled at the highest rate while the wall behind it needs only a few samples. In the case of motion caused by viewpoint changes as in this figure, the segments can be sorted in order of depth as discussed in Section 2.4.2. Each segment is filtered independently and a temporal coverage value for each pixel is kept to indicate the ratio of background samples vs. all samples. The multiple segment layers are then composited in front-to-back order with each segment's pixel colors attenuated by the coverage value from the previous segment.

4.3 Shadows

A very general and efficient way of rendering shadows is the shadow buffer algorithm [WILL78]. The algorithm computes a Z-buffer (i.e., shadow map) from the point of view of the light source. To compute shadows, a surface point's coordinates are transformed to the light source's space and its Z-coordinate is compared to the corresponding Z-coordinate in the shadow map. If the point is further away then it is in shadow.

The algorithm only works for point light sources. To approximate a linear or an area source, many point lights may be needed [SHAP84]. The cost of computing the shadows is proportional to the number of point sources used.

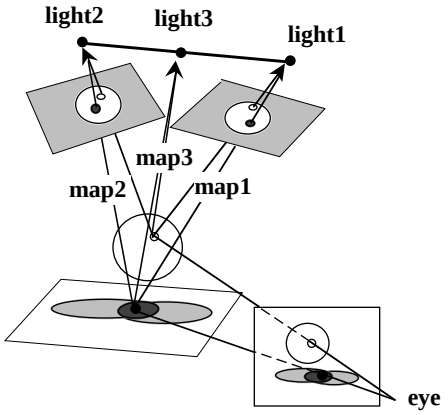

Fig. 12 Shadow buffer interpolation for a linear light source

The morphing method can be used to significantly reduce the cost of computing the shadow map for each of the point sources. Figure 12 illustrates the process of using the method to compute shadows from a linear light source. A shadow map is computed first for each of the two end points of the source (i.e., light1 and light2) using the conventional rendering method. A morph map from the viewpoint to each of the two end points is also computed to transform the screen coordinates to each point source's coordinate space (i.e., map1 and map2). The shadow map for an in-between point (e.g., light3) on the linear source is interpolated from the corner shadow maps using the morphing method. The same interpolation factor is used to interpolate the two morph maps (map1 and map2) to create a morph map from the viewpoint to the in-between light source point (map3). The standard shadow buffer algorithm is then used to compute shadows for the in-between point source. The process is repeated for all the in-between points at a desired interval. The resulting shadow images are composited to create the soft shadow of the linear source. This method can be generalized to any area or volume light source.

Figure 13 shows the result after compositing 100 in-between shadow images generated by randomly distributed points on a rectangular light source above the triangle. Four source shadow maps located at the corners of the rectangle were created for the interpolation. The shadow maps were rendered at 512x512 resolution and the shadow image resolution is 256x256. Percentage closer filtering [REEV87] was used to anti-alias the shadows for each image. Each shadow image took 1.5 seconds to compute. Shading for the illuminated pixels was computed by Lambert's Law weighted by the projected size of the rectangle source over the pixel.

4.4 Image-Based Primitives

A 3D object is perceived on a flat display screen through a series of 2D images. As long as we can generate the images from any viewpoint, it does not matter if a 3D description of the object is available. The morphing method permits any view of an object to be generated by interpolation from some key images. Therefore, a new class of primitives based on images can be defined. These image-based primitives are particularly useful for defining objects of very high complexity since the interpolation time is independent of the object complexity.

Figure 14 shows a sequence of images of a rotating teapot generated by the morphing method. The middle images were generated by interpolating the two key images at the extreme left and right. The key images were rendered with viewpoints rotated 22.5 degrees around the center of the teapot. A larger angular increment of the key images may result in holes and distortions as a result of the linear interpolation. Figure 15 is the same source images extrapolated to show the pixel blocks which compose the teapot.

Rendering an object using the morphing method is really not different from rendering a complete scene as described previously. The image-based object or scene can be treated as a "sprite" that can be composited with images generated by other means.

4.5 Incremental Rendering

Adjacent images in an animation sequence usually are highly coherent. Therefore, it's desirable to perform the rendering incrementally. Ideally, the rendering should be limited to only the pixels which are different from the previous frame. However, searching for the pixels that change is not always trivial. Some incremental rendering approaches which make use of frame-to-frame coherence were presented in [CHEN90], [JEVA92].

The morphing method provides a natural way of making use of frame coherence. For an animation sequence where the motion of every frame is known in advance, the frames can be rendered initially at a coarse temporal sampling rate. The remaining frames can then be computed by the morphing method. The missing samples or view-dependent shading, such as highlights, of the interpolated frames can be computed by additional rendering. If accuracy rather than speed is the main concern, the map-based interpolation or extrapolation of pixel coordinates can be replaced by perspective transformation.

5 CONCLUSIONS AND FUTURE DIRECTIONS

The interactive speed which the image-based display has achieved on modest computing platforms has fulfilled our primary goal in pursuing this research. In addition to this primary objective, we have demonstrated effective application of the view interpolation approach to computing some of the more complex rendering effects. Image-based computer graphics promises to be a productive area of research for some time. A number of intriguing research problems suggest themselves:

An automatic camera has been developed to record an array

of images of an object from viewpoints surrounding it [APPL92]. What are the prospects for automatic camera location selection to minimize the number of holes in the interpolated images? Similarly, what are good algorithmic criteria for dispensing with as many recorded images as possible, or selecting the best subset of images to represent the object?

By modeling the 3D transformation from one image to the next by a field of straight-line offsets, we introduce an approximation analogous to polygonization (except in the restricted cases mentioned in Section 2.2). Higher-dimensional, rather than linear, interpolation might be expected to better approximate the arcs traversed by objects rotating between views. Curved motion blur is another possible benefit of higher-order interpolation.

View-dependent shading such as specular reflection would extend the useful range of morphing as a display technique. One possibility mentioned previously is to define additional maps for specular surfaces, which specify normal components or reflection map coordinates.

Special-purpose image compression might profit greatly from morph-mapping algorithms. The resemblance of the morph maps to motion-compensation vectors commonly used in video sequence compression has been mentioned. These vectors, used in format conversion to address the interlace problem, and in compression to squeeze a little more redundancy out of the signal, also find application in optical flow algorithms for tracking objects in the visual field. The redundancy removed from the video sequence by motion compensation is limited, as it applies only between successive frames. In a morph mapping encoder, objects which appear and disappear repeatedly could be encoded with a small set of maps. The decoder, a hybrid of an image warper and a graphics pipeline, would use them as "sprites" from a catalog of maps.

The representation of objects and surfaces as sets of images and maps, possibly pyramidal maps, suggests the application of morph mapping to more general global illumination models. The approach of determining visibility to an area light source to compute soft shadows can be extended to treating all surfaces as sources of radiosity. For many global illumination problems, a few images and morph maps can serve to represent hundreds or thousands of computed images.

6. ACKNOWLEDGMENTS

Thanks to the Virtual Museum team for the museum model and images. Dan Venolia anticipated the use of range images as display primitives (without interpolation) in his virtual holography work. Ken Turkowski contributed the teapot images. Ned Greene, Nelson Max and members of the Advanced Technology Computer Graphics Group have offered useful ideas and criticism. Frank Crow and Apple Computer's continuous support of this research is highly appreciated.

REFERENCES

[AIRE91] Airey, J., J. Rohlf and F. Brooks. Towards Image Realism with Interactive Update Rates in Complex Building Environments. ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics, 41-50.

[APPL92] Apple Human Interface Group. Object Maker. [exhibit] In Interactive Experience, CHI'92, Monterey CA.

[BESL88] Besl, P.J. Active Optical Range Imaging Sensors. Machine Vision and Applications Vol. 1, 1988, 127-152.

[BEIE92] Beier, T. and S. Neely. Feature-Based Image Metamorphosis. SIGGRAPH'92 Proceedings, 35-42.

[CHEN90] Chen, S. E. Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System. SIGGRAPH'90 Proceedings, 135-144.

[COHE88] Cohen, M. F., S. E. Chen, J. R. Wallace and D. P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. SIGGRAPH'88 Proceedings, 75-84.

[DEER92] Deering, M. High Resolution Virtual Reality. SIGGRAPH'92 Proceedings, 195-202, 1992.

[GOSH89] Goshtasby, A. Stereo Correspondence by Selective Search. Proc. Japan Computer Vision Conf., 1-10, July, 1989.

[GREE86] Greene, N. Environment Mapping and Other Applications of World Projections. IEEE CG&A, Vol. 6, No. 11, November, 1986.

[GREE93] Greene, N. and M. Kass. Approximating Visibility with Environment Maps. Technical Report 41, 1993, Apple Computer, Inc.

[HOFM88] Hofman, G. R. The Calculus of the Non-Exact Perspective Projection. Eurographics'88 Proceedings, 429-442

[JEVA92] Jevans, D. Object Space Temporal Coherence for Ray Tracing. Graphics Interface'92 Proceedings, 176-183, 1992.

[LIPP80] Lippman, A. Movie Maps: An Application of the Optical Videodisc to Computer Graphics. SIGGRAPH'80 Proceedings, 32-43.

[MILL92] Miller, G., E. Hoffert, S. E. Chen, E. Patterson, D. Blacketter, S. Rubin, S. A. Applin, D. Yim and J. Hanan. The Virtual Museum: Interactive 3D Navigation of a Multimedia Database. The Journal of Visualization and Computer Animation, Vol. 3, No. 3, 183-198, 1992.

[MILL93] Miller, G.and S. E. Chen. Real-Time Display of Surroundings Using Environment Maps. Technical Report 42, 1993, Apple Computer, Inc.

[MPEG90] MPEG Video Committee Draft, December, 1990.

[NAGE86] Nagel, H.-H. Image Sequences - Ten (octal) Years from Phenomenology to a Theoretical Foundation. Proc. 8th ICPR, Paris 1986, 1174-1185.

[POGG91] Poggio, T. and R. Brunelli. A Novel Approach to Graphics. MIT A.I. Memo No. 1354, C.B.I.P. Paper No. 71, February, 1992.

[REEV87] Reeves, W. T., D. H. Salesin and R. L. Cook. Rendering Antialiased Shadows with Depth Maps. SIGGRAPH'87 Proceedings, 283-291.

[SCHU69] Schumacker, R., B. Brand, M. Gilliland, and W. Sharp. Study for Applying Computer-Generated Images to Visual Simulation, Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX, September, 1969.

[SHAP84] Shapiro, B. L., N. I. Badler. Generating Soft Shadows with a Depth Buffer Algorithm. IEEE CG&A, Vol. 4, No. 10, 5-38, 1984.

[TELL92] Teller, S and C. Sequin. Visibility Preprocessing for Interactive Walkthroughs. SIGGRAPH'91 Proceedings, pp.61-69, 1991.

[VENO90] Venolia, D. and L. Williams. Virtual Integral Holography. Proc. SPIE-Extracting Meaning from Complex Data: Processing, Display, Interaction (Santa Clara, CA, February, 1990), 99-105.

[WILL78] Williams, L. Casting Curved Shadows on Curved Surfaces. SIGGRAPH'78 Proceedings, 270-274.

[WILL90] Williams, L. 3D Paint. ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics, 225-233.

[WOLB89] Wolberg, G. and T. E. Boult. Separable Image Warping with Spatial Lookup Tables. SIGGRAPH'89 Proceedings, 369-377.

[WOLF83] Wolf, P. R. Elements of Photogrammetry, McGraw-Hill, New York, 1983.