The Delta Tree:
An Object-Centered Approach to Image-Based Rendering

William J. Dally[*][†], Leonard McMillan[†], Gary Bishop[†], and Henry Fuchs[†]

[*]Artificial Intelligence Laboratory, Massachusetts Institute of Technology
[†]Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

This paper introduces the delta tree, a data structure that represents an object using a set of reference images. It also describes an algorithm for generating arbitrary re-projections of an object by traversing its delta tree. Delta trees are an efficient representation in terms of both storage and rendering performance. Each node of a delta tree stores an image taken from a point on a *sampling sphere* that encloses the object. Each image is compressed by discarding pixels that can be reconstructed by warping its ancestor's images to the node's viewpoint. The partial image stored at each node is divided into blocks and represented in the frequency domain. The rendering process generates an image at an arbitrary viewpoint by traversing the delta tree from a root node to one or more of its leaves. A subdivision algorithm selects only the required blocks from the nodes along the path. For each block, only the frequency components necessary to reconstruct the final image at an appropriate sampling density are used. This frequency selection mechanism handles both antialiasing and level-of-detail within a single framework. A complex scene is initially rendered by compositing images generated by traversing the delta trees of its components. Once the reference views of a scene are rendered once in this manner, the entire scene can be reprojected to an arbitrary viewpoint by traversing its own delta tree. Our approach is limited to generating views of an object from outside the object's convex hull. In practice we work around this problem by subdividing objects to render views from within the convex hull.

1. Introduction

The use of images as an underlying model for scene representation is an exciting new paradigm in computer graphics. Generating a scene by re-projecting and compositing images of objects rather than rendering geometric object representations has advantages in performance, level-of-detail management, and model creation. In image-based approaches the computation required to render a scene is proportional to the number of pixels in the images that compose the scene model, independent of the scene's geometric complexity. Since the process of re-projecting an image to a new viewpoint, often called an *image warp* [Wolberg90], uses fast incremental calculations, it can usually be accomplished faster than rendering a geometric model. Another advantage of the image-based rendering is that both level-of-detail and antialiasing can be handled within a single consistent framework by using a frequency-domain representation of the image and warping only those frequency components that are necessary for the desired re-projection. This frequency-based approach automatically allows for differences in the sampling densities between the reference image used to model the scene components and the desired re-projections. This further reduces the number of pixels warped. The reference images used in an image-based rendering system can be acquired from a multitude of sources including rendered polygonal models, photographs, and volume data sets. This provides significant modeling flexibility because the representation of each scene element can be determined according to both its complexity and its overall importance to the scene.

Efficient data representation is a key problem in building any image-based rendering system. Typically several hundred images are required to represent each object and rendering an image of an object at a new viewpoint may involve reprojecting pixels from several of these images. An ideal space-efficient representation stores exactly one sample for every potentially visible component of an object down to some minimal solid angle of resolvabilty. Likewise, a representation that requires reading only one sample for each pixel rendered to the screen would provide for optimal rendering performance.

In this paper we introduce the delta-tree, an efficient data structure for representing an object with images. We also present an algorithm for generating arbitrary views of an object using its delta tree. The delta tree stores an object's images in a multi-rooted tree where each node corresponds to a region of a *sampling sphere* centered on the object and it stores an image of the object from a viewpoint within that region. The view stored at each node is compressed by discarding pixels that can be reconstructed by warping its ancestor's views to the node's viewpoint thus providing a space-efficient representation. A view of the object from an arbitrary viewpoint is generated by traversing a portion of the object's delta-tree and reprojecting portions of the partial views at each node. The set of nodes to traverse and the portion of each view to reproject are selected by a simple recursive subdivision algorithm to give a bandwidth-efficient playback.

2. Background and Related Work

Image-based representations were first introduced to computer graphics in the form of texture mapping [Catmull74] [Blinn76] [Blinn78] [Heckbert86]. The advantage of textures is that they increase the apparent visual complexity of a scene while adding only a small cost to the rasterization process. Notice that there are significant parallels between the incremental computation of texture indices and the image warping process. Similarly, techniques such as MIP mapping [Williams83] and summed-area tables [Crow84] bear a strong resemblance to frequency-domain reconstruction methods. The limitation of texture maps is that the image elements are constrained to follow the underlying geometry of the primitive upon which the texture is mapped.

The recently introduced methods of view interpolation [Chen93], Quicktime VR [Chen95], fundamental-matrix-based re-projection [Laveau94], and plenoptic modeling [McMillan95b]demonstrate the potential of the image-based rendering approach. All of these methods take advantage of the incremental evaluation in their image warps and the amount of computation required is proportional to either the number of pixels in the reference images (view interpolation and plenoptic modeling) or the output image (fundamental-matrix-based re-projection and Quicktime VR). One notable disadvantage of all of these image-based rendering approaches is that they model the entire scene as a single image element, rather than a composition of individual scene elements. Put another way, in most previous image-based rendering approaches have taken a viewpoint-centered approach to modeling a scene rather than an object-centered approach. However, in many applications it is desirable to synthesize scenes from the bottom up by combining various precomputed image-based scene elements. This object-centered view characterizes the image-based rendering system described in this paper. Other researchers have also built image-based systems with an object-centered orientation. The digital compositing techniques [Porter84] [Duff85] have been used to construct complicated scenes by combining prerendered images that share a common viewpoint. Other object-centered approaches include the object-movie component of the QuicktimeVR system [Chen95] and the tree generation system described in [Max95].

3. Definitions

A reference image or *view*, V_p^x, is a hybrid photometric/geometric representation of object *x*, with range data, taken from viewpoint *p* with a specified field-of-view. A view is represented as a dense two-dimensional array of samples with a given width and height. An *image sample* or pixel stores the range information, transparency, surface attributes, and surface normal of a point on the surface of an object.[1] A *partial view* ${}^aV_p^x$ includes those pixels of V_p^x that sample surfaces of *x* that are not adequately sampled by the set of views, *a*. We represent a partial view as a sparse array of *patches*, $n \times n$ arrays of pixels. A *subview* of a view is a rectangular subset of the view with possibly reduced width or height.

An *object* is a scene element represented by a set of views. An object may be primitive or compound. A *primitive object* is represented only by its reference images, which may have been acquired by rendering a polygonal model, taking photographs of a real object, or some other means external to our system. A *compound object* is represented by both its reference images and a set of object-transform pairs. The images of a compound object are generated by transforming and reprojecting the images of its constituent objects. If a compound object is to be viewed only a few times it is easier to use its constituent representation, reprojecting each object to generate a view. On the other

---

[1] Our prototype system does not yet handle transparency.

hand, if a compound object is to be viewed many times, it is more efficient to use its image representation, by generating the reference images of the compound object once and reprojecting them to generate a view. A *scene* is just a compound object.

We represent an object, primitive or compound, with a set of views taken from viewpoints on a *sampling sphere*, a sphere with a given radius, r_s, centered on the object. The center of the sphere is considered to be the origin of the object coordinate system. We will refer to points on the sampling sphere with just their angular coordinates, (θ, ϕ), taking the radius to be implied. Within our system we represent these angles by their coordinates on the unit sphere.

4. Representing an Object with Views

As shown in Figure 1, reprojecting a view, V_p, of an object from viewpoint p on a sampling sphere to a view V_q from viewpoint q may result in holes, if a surface visible from q is occluded from p, or blurring, if a surface is at an oblique angle to p and thus undersampled in V_p. Plate 1 shows this effect on an actual image. When a view of the teapot (left) is warped to a viewpoint 30 degrees to the right, the unsampled surfaces behind the spout and to the rear of the pot leave holes. Constructing an adequately sampled V_q may require reprojecting pixels from several views. The view of the teapot in Plate 1 (right), for example was reconstructed by warping portions of images from five viewpoints. However, it requires reprojecting only the number of pixels visible in the final image, $|V_q|$. The challenge of object representation is to index the views so the proper set of pixels from the proper set of views can be quickly identified and accessed.

Figure 1: Reprojecting a view from p to q.

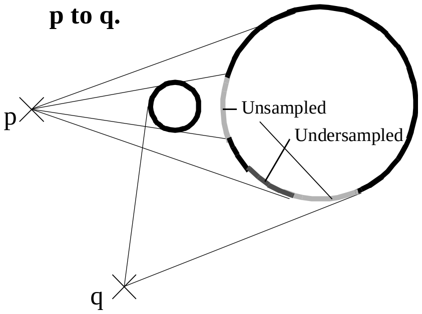
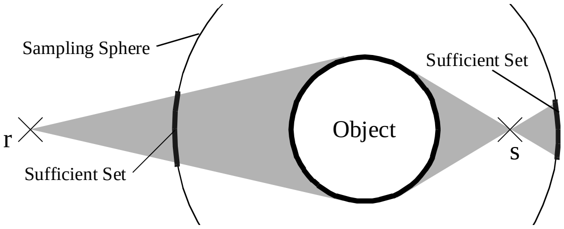


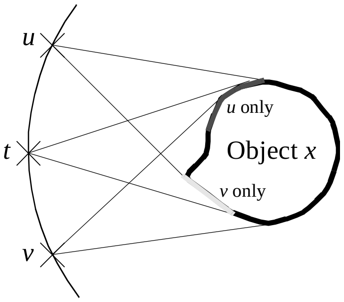
Figure 2: A sufficient set of views is found by intersecting the viewing rays with the sampling sphere.



A view of an object from any point outside its convex hull can be rendered by reprojecting a set of views taken from a sampling sphere. A sufficient set of views can be found by intersecting the sampling sphere with the set of pixel rays that pass through the viewpoint and a point on the object. This sufficient set is illustrated in Figure 2 for a viewpoint outside the sphere, r, and inside the sphere, s. There are pathological surfaces for which this sufficient set of views is also necessary. However, most objects can be rendered from an easily identifiable set of pixels from a small set of views.

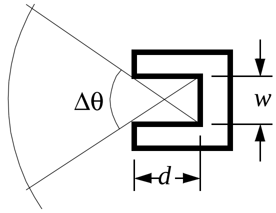
We represent an object with a set of reference views taken at discrete points on the sampling sphere. To reconstruct a view at an unsampled point, *t*, we reproject the nearest set of reference views that surround *t* — viz. the convex hull of their viewpoints encloses *t* in (θ, ϕ) space. The need for surrounding a viewpoint with sample points is illustrated for two dimensions in Figure 3. The horizon of the object as seen from any sample point, *v*, below viewpoint, *t*, will always be closer than as seen from *t*. Thus the top surface of the object will not be adequately sampled and warping the view from *v* to *t* will give an incorrect top silhouette edge. This edge can only be properly reconstructed from sample points at or above *t*, such as *u*. Similarly, the bottom surface of the object is occluded from *u* and can only be reconstructed from *v*. Even a very smooth object with curved surfaces will not be properly reconstructed unless views are warped from a surrounding set of viewpoints. In three dimensions, a minimum of three viewpoints must be used to reconstruct an object. The delta-tree, as described below, divides (θ, ϕ) space into square regions and always uses the four corners of the region enclosing the viewpoint to reconstruct an image.

Figure 3: A viewpoint must be surrounded by sample points



Theoretically, determining the number of views required to represent an object is a difficult problem and there is a broad literature on determining the *aspect graph* of an object [Plantinga90]. In practice, however, we have found a simple regular grid on the sampling sphere generates a set of views which adequately covers the surface of most objects. We are not concerned with generating a minimal set of views as the delta tree encoding, described below, automatically discards views that are completely redundant.

Figure 4: Sampling density is driven by surface topography



The required density of the sampling grid in (θ, ϕ) space is determined by the aspect ratio of the object's surface topography as illustrated in Figure 4. If our object has troughs of width, w, and depth, d, our samples must be spaced at least every $\Delta\theta = 2\operatorname{atan}(w/d)$ to cover the surface of the trough. We check the coverage of a set of views by reprojecting the views, as described below, to unsampled points on the sampling sphere and checking for holes. This method could be used to perform automatic grid refinement for difficult objects.

A limitation of representing objects with images taken from a sampling sphere is that we cannot in general generate views of an object from inside the convex hull of the object as surfaces may be visible inside the object that are not visible from the sampling sphere. This means, for example, that we cannot move the viewpoint inside a tunnel through an object. In practice we overcome this limitation by splitting the object down the center of the tunnel to make the inside of the tunnel visible from a sampling sphere.

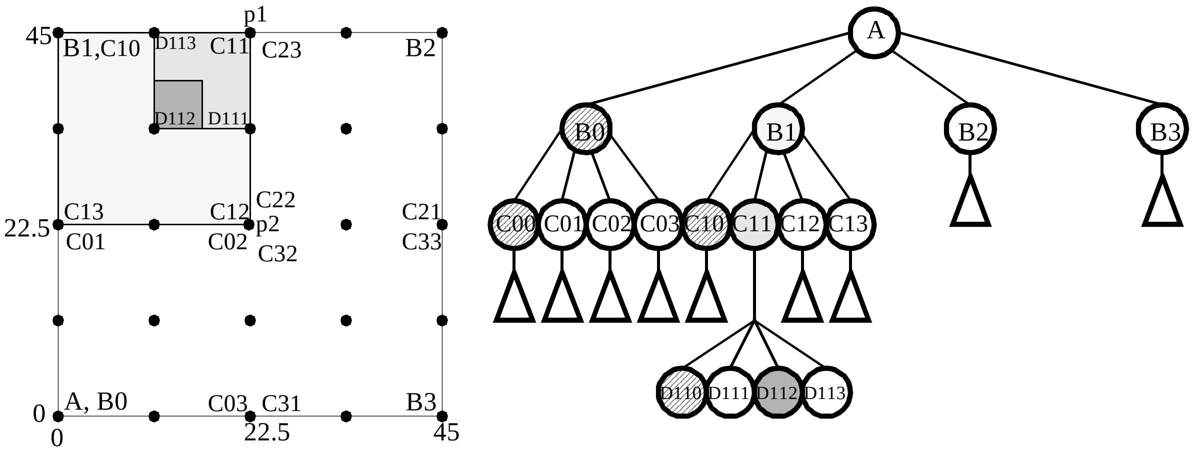
5. The Delta Tree

A delta tree encodes the views representing an object in a quad-tree that divides the angular space of the sampling sphere. Each view is compressed by discarding pixels that sample surfaces adequately sampled by its ancestors. The number of pixels stored to represent an object is equal to the number of pixels needed to adequately sample the surface of the object as seen from the sampling sphere. The views are arranged so an image of the object from an arbitrary viewpoint can be rendered by reprojecting portions of a small set of views identified by a walk of the tree.

5.1 Structure

As shown in Figure 5 each node of the delta tree is associated with a region of the sampling sphere and stores an image of the object as seen from a corner of its region. For example, the region of root node A is the entire square shown in the figure, $(\theta, \phi) = ([0, 45], [0, 45])$, and node A stores a view from (0,0). A complete delta tree contains 4P-such root nodes, one for each quadrant about P primary views. Typically P=6 corresponding to the six cardinal directions. Each child of a node inherits a quarter of its parent's region The regions of B1, C11, and D112, for example, are the shaded regions of the figure. The labeled dots in the figure show the location of the viewpoints associated with the nodes. Each child stores a partial view ${}^aV_p$ at the corner, p, of its region that is also a corner of its parent's region. The partial view includes only those pixels that cannot be adequately reconstructed from the partial views stored by the child's ancestors, a. For leaf nodes, we include the node's older siblings in set a since, as described below, all siblings of a leaf are visited together during reconstruction. If the partial views associated with all of a node's children are empty, the children are deleted and the node becomes a leaf.

Figure 5: Each node of the Delta Tree corresponds to a region of viewing space.



An Example Delta Tree: Plate 2 shows one of 24 quadrants of a three-level delta-tree constructed for a teapot. The partial view stored at each non-empty tree node is plotted in a corner of the node's region in (θ, ϕ) space. This matches the layout of the left side of Figure 5 down to the level of the C nodes. The four columns correspond to viewpoints with $\theta = 0$, 22.5, 22.5, and 45 from left to right and the four rows correspond to viewpoints with $\phi = 0$, 22.5, 22.5, and 45 from bottom to top. Note that two nodes with the same viewpoint, e.g., C03 and C31 at (22.5,0), have very different partial views. The view at C31 has many fewer pixels as it inherits more pixels from its ancestors and siblings than C03. Storing fewer pixels at C31 reduces the number of pixels that must be reprojected to render a view in its region. Storing multiple views from a single viewpoint does not increase the storage overhead as these views share pixels by reference as described below. The compression by partial views is evident in the figure. View A contains 34,000 non-background pixels while the average third-level view contains 3800 non-background pixels.

View Sharing: There are four distinct cases in which nodes of the tree share views. In each case, common pixels are shared by reference and no redundant pixels are stored.

1. The four trees corresponding to the four quadrants about a primary view share views along their common edges. For example, the views stored by A, C01, C13, and B1 are shared with the tree that covers quadrant 2, $\theta = [-45, 0]$. In this case the view at A and partial views along the edge are identical in the two quadrants so the views are shared by reference.
2. One child of each node shares a view with its parent. For example, B0 shares a view with A, and C10 shares a view with B1. In this case, the child, e.g., B0, need not store any pixels of its view since every pixel is adequately sampled by an ancestor, e.g., A. These children with empty partial views are shown cross hatched on the right side of Figure 5.
3. First cousins within a tree may share a view. For example, C11 shares a view V_{p1} with C23, and four cousins share V_{p2}. In this case, the cousins store different partial views. For example, C11 stores ${}^{\{A, B1\}}V_{p1}$ containing only those pixels that are not adequately sampled by A or B1, while C23 stores ${}^{\{A, B2\}}V_{p1}$. Any pixels in common between the partial views are shared by reference as described below. In some cases the set of pixels stored by a node contains the set stored by its cousin. For example, C01 contains all of the pixels of C13 since C13's ancestors {A,B1} includes C01's ancestors {A}.
4. The trees associated with two primary views share views along their common edges. This case is identical to case (3). The nodes have different ancestors and thus store different partial views sharing common pixels by reference.

Redundancy: Even with parent-child compression and view sharing, a number of redundant pixels remain in the delta tree. Two sibling nodes may sample the same surface, and undersampled pixels in an ancestor may be resampled in a descendant. For example, the views corresponding to B1 and B2 in Plate 2 both sample the entire top of the teapot. Redundant samples of the teapot top also appear in C01 and C33. Several views also include a These redundant samples are due to an artifact in the present version of our warper[2]. The delta tree is robust in the presence of such artifacts. If a single pixel in a patch is not properly reconstructed from an ancestor, the entire patch is retained in a node's partial view. This results in a final image free from artifacts at the expense of extra storage and rendering time.

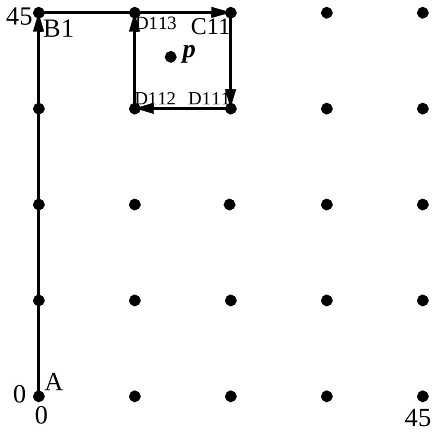
Memory Management: A delta-tree typically has hundreds of nodes of which a small fraction are used at a particular time. For example, the subtree representing the side of an object opposite the viewpoint is not likely to be accessed in the near future. To optimize storage, the nodes of a delta tree can be loaded or synthesized on demand by treating a small set of in-memory nodes as a cache. Pointers to non-resident nodes are flagged with a reserved value. When this value is detected in traversing a tree, one of the in-memory nodes is evicted and the requested node is loaded from backing store or generated on-the-fly, for example by a polygonal renderer. If the demand for a node can be anticipated, for example by predicting viewpoint motion, then the generation or loading of nodes can be done at less than real time rates.

5.2 Reconstructing a View

Views on the sampling sphere: A view of the object from a viewpoint, *p*, on the sampling sphere is generated by walking the tree from the root to the leaf whose region contains *p* and including the siblings of this leaf to surround *p* with the nearest possible views. For example, if *p* is in the small shaded region in Figure 5, the path {A, B1, C11, D111, D112, D113} is traversed. As shown in Figure 6 the layout of the delta tree causes the traversal follows a spiral pattern in (θ, ϕ) space. This ensures that, except in a few degenerate cases, the first four nodes visited will surround the viewpoint. The nearest child node at each level is selected by dotting the normal corresponding to *p* with the normal corresponding to the center of the child's region. At each node, each pixel of the required portion of the stored partial view is reprojected (warped) to viewpoint *p* and composited in a z-buffer with the warped views of its ancestors. The z-buffer is required to eliminate hidden surfaces visible along the path.

Generating a view involves reprojecting slightly more than the number of pixels that appear in the final view. The delta-tree's construction ensures that no surface of the object is sampled more than once. However, some surfaces that are not visible from p will be rendered and some surfaces that are viewed obliquely from p will be slightly oversampled.

Figure 6: The Delta Tree is traversed in a spiral pattern.



Views off the sampling sphere: As illustrated in Figure 2, generating a view from a point off the sampling sphere may require reprojecting pixels from different points on the sampling sphere and hence from different leaf nodes of the delta tree. To handle this situation we recursively subdivide the view we are generating as we traverse the delta tree so that at each level of the delta tree all of the pixels in a subview are rendered from the same region.

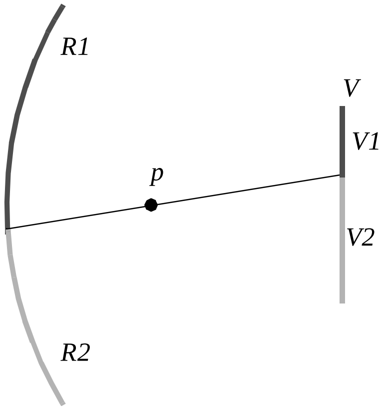
---

[2] We expect to have this eliminated before preparing the final draft of the paper.

Given a delta tree of an object we generate a view of the object from an arbitrary viewpoint using the following pseudocode algorithm. I

```
render(node, viewpoint, view)
  select portion of node's partial view that projects to view
  reproject this portion through viewpoint and composite into view
  if node has children
    divide view into subviews along boundaries between children
    for each subview
      select child of delta tree corresponding to this subview
      render(child, viewpoint, subview)
```

Figure 7: A view is subdivided by projecting regions.



The subdivision is performed by projecting the region boundary on the sampling sphere onto the view. A cross section of this projection is illustrated in Figure 7. The boundary between regions *R1* and *R2* on the sampling sphere is projected through the viewpoint, *p*, onto the view plane, *V*. The projected boundary divides the pixels into two subviews: V1 to be rendered using the delta tree node corresponding to R1, and V2 to be rendered using the node corresponding to R2. Depending on whether a subview straddles zero, one, or two boundary edges, it may be left undivided, or divided into two or four subviews.

To select the portion of a node's partial view that projects to a given subview we project the corners of the subview to the node's viewpoint. Depth information is required to perform this projection, so we do it twice using the minimum and maximum depth of the object. The bounding rectangle of the resulting eight points is used to select the pixels of the partial view to reproject. As a fringe benefit this step also prevents us from reprojecting most pixels that will wind up being clipped by the warper when an object straddles the edge of a view.

Level of Detail: A change in viewing distance or field-of-view may require magnifying or minifying an image during reprojection. To handle magnification, our warper scan converts the quadrilateral corresponding to four adjacent, connected pixels interpolating color and depth values between the vertices. As one zooms into an object, fine detail appears blurred as one cannot see more detail than was present in the original image as with any sampled image approach. With magnification, the number of pixels reprojected to construct a view of an object is often less than the number of pixels that appear on the screen, with the scan converter making up the difference.

Minification is handled by low-pass filtering and decimating images before reprojection. This could be implemented using MIP maps; however, our current implementation filters by storing the images in the frequency domain as described below. In either case, the filtered image has the required linear size rounded up to the next largest power of two. The worst case, where the required size is just greater than one power of two, requires reprojecting four times as many pixels as appear on the screen. In practice this is not a serious overhead as the number of pixels subtended by a minified object tends to be small. This approach allows us to efficiently render an object from a single representation with level-of-detail ranging from the entire screen to a single pixel. No level-of-detail hierarchy with multiple representations is required.

5.3 An Example of Image Generation

Plate 3 illustrates the generation a view of the teapot from the delta tree of Plate 2. The view is reconstructed at a viewpoint of (30, 15, $0.75 r_s$) giving a slight magnification. The top row shows the partial views stored in the five nodes of the delta tree, corresponding to {A, B3, C32, C31, C33} in Figure 5. The middle row shows the result of warping, with magnification, each of the partial views by itself to the final viewpoint. Each frame of the final row shows the result of compositing the frame above with the frame at the left. The view from A is missing the top and back of the pot and leaves a hole behind the spout. Adding the view from B3 fills in the back and the hole behind the spout. The remaining views fill in small details about the rim with C33 completing the top of the teapot.

6. Implementation Notes

This section describes some of the details of our delta tree implementation and discusses design alternatives.

6.1 Partial View Representation

The partial view stored in each node of the delta tree is a sparse matrix of pixels. A straightforward representation as pixel vector and index vector has a number of limitations. The overhead of storing a 24-bit index,12-bits each for x and y, for each pixel would be substantial. With an unstructured pixel vector, the warper would be unable to use an incremental transformation as it steps across scan lines. Instead each pixel would have to be independently transformed. The lack of structure also complicates inter-pixel compression methods.

To overcome these limitations, our present implementation stores a partial view as a list of $n \times n$, typically 8×8, arrays or *patches* of pixels. The selection of the portion of a view to include in a partial view and the portion of a partial view to reproject are made in terms of patches rather than pixels. This increases the number of pixels stored and projected as partially filled patches are filled out; however, it has a number of advantages. The overhead of indexing, both storage and index manipulation, is amortized across n^2, typically 64, pixels as is the decision logic for deciding whether or not to include a pixel or patch in a selection. The regular structure of each patch also facilitates incremental warping, DCT-based inter-pixel compression, and the first level of image filtering.

We combine image compression with reconstruction filtering by storing each patch of a partial view in the frequency domain as a set of DCT coefficients. To minify a view by a factor of i/n, where $i \in [1, n-1]$, we zero all but the first i frequency components in each dimension, set the size of each dimension to i rounded up to the nearest power of 2, and perform the IDCT. The resulting filtered view is then reprojected. To avoid distorting the shape of the object when filtering z-values at object boundaries, background z must be set near the object depth. A large z step near the object edge badly distorts nearby pixels when filtered. While we have not yet implemented compression we plan to compress each patch by quantizing and variable-length coding the DCT coefficients as is done in JPEG.

There are several alternative representations that we considered and rejected. One could store a partial view as a run-length coded vector of pixels. Incrementally computing the index, however, makes it difficult to reproject portions of a partial view without traversing the entire index vector. Also run-length coding complicates block-based image compression. Another alternative is to simply zero out the unused pixels of a partial view and depend on image compression to exploit the regularity. This method has the disadvantage that it still requires storage of an index vector to rapidly identify the non-zero pixels.

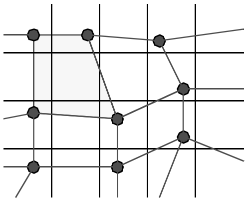
6.2 Reprojecting Views

Our approach to reprojecting views is based on that of [McMillan95b]. Forward mapping is used to reproject a pixel at a point $X_s = (x_s, y_s, z_s)$ in the source image, a node of the delta tree, to a point in the target image $X_t = (x_t, y_t, z_t)$, the view being generated. Because we are performing a 3-D warp, backward mapping, as is commonly used to texture-map polygons, is not a viable alternative. As we do not know z_t a priori, a backward warp would require a costly search to find the nearest (there may be more than one) point in the source image that maps to (x_t, y_t).

A perspective transformation is used to map each point $X_s(i)$ to the corresponding $X_t(i)$. The warper incrementally evaluates the transformation within a patch exploiting the correlation of adjacent pixels. For example, for each step in the x direction, the warper performs the computation shown at right. T_{ab} is an element of the transformation matrix. Every n-th step, the warper uses a slightly different set of equations to take a step in y using values remembered from the start of the last row. Each step of the incremental calculation requires 1 reciprocal, 5 multiplies, and 6 adds as compared to 1, 12, and 9 for a general perspective transformation.

$$\begin{aligned}
x_t'(i) &= x_t'(i-1) + T_{xx} \\
y_t'(i) &= y_t'(i-1) + T_{yx} \\
z_t'(i) &= z_t'(i-1) + T_{zx} \\
z_t(i) &= z_t'(i) + z_s(i)T_{zz} \\
d &= 1/z_t \\
x_t(i) &= d(x_t'(i) + z_s(i)T_{xz}) \\
y_t(i) &= d(y_t'(i) + z_s(i)T_{yz})
\end{aligned}$$

Figure 8: Scan Conversion



In general each on-grid point, X_s, in the source image transforms to an off-grid point, X_t, in the destination image. A scan converter is used to interpolate the color and z values at the on-grid points as illustrated in Figure 8. The figure illustrates how nine points of the source image might map irregularly onto the target image. For each quadrilateral formed by four points of the source image, the enclosed pixels are rendered by interpolating the values of the vertices. For example, the three pixels of the target image contained in the shaded quadrilateral are rendered using the values of the four surrounding pixels of the source image. Our scan converter exploits the incremental nature of reprojecting a patch exploiting the fact that the right edge of one quadrilateral is the left edge of the next quadrilateral.

Interpolation across a quadrilateral is only performed if the four vertices are determined to belong to the same surface as determined by a comparison of z-values. If the four vertices belong to more than one surface, each pixel is rendered by extrapolating the values of the surface corresponding to the nearest pixel. If one or more of the vertices belongs to the background, the pixels closest to the background vertices are set to background.

6.3 Shading

Our system shades the final image of a scene after composition using the surface normal at each pixel in the image and a lighting model. At present our system supports only point light sources and ambient light sources. To facilitate shading, the images used to model an objects are acquired using only ambient lighting. As this gives them a rather flat appearance, several of the figures in this paper were generated using shaded images to improve clarity. Our actual delta trees, however, are all built with unshaded images. We conjecture that the regularity of these unshaded images will result in better image compression.

Our present system computes the surface normals for each view of an object using the difference in depth between adjacent pixels. This gives some objects, such as the teapot and bishop, a ribbed appearance as while the polygonal renderer we used to generate their images interpolated normals across each polygon, the depth reflects the original flat polygon.

6.4 Extensions

Symmetric Objects: The delta tree handles symmetric objects using view sharing. If an object that is symmetric about a plane, the partial views of nodes on either side of the plane store identical views by reference. If an object is symmetric about a line, only nodes with $\phi = 0$ are stored. A single view suffices to describe a point-symmetric objects.

Effects: Our current system does not implement effects such as shadows or reflections. However, it is straightforward to implement these using methods similar to those employed by polygonal rendering systems. Shadows are handled by projecting the scene to the viewpoint of each light source and retaining the z-values of the resulting images. During the final shading process, each pixel in the image is warped to the viewpoint of each light source and compared to the stored z-value to determine if it is illuminated. In a similar manner, a reflecting surface is rendered by computing the scene as seen from the mirror image of the actual viewpoint clipped to the boundary of the reflecting surface.

6.5 Alternatives

During the development of the delta tree we considered and rejected several alternative representations. One of the more promising alternatives involved storing at each leaf the minimum set of partial views required to reconstruct a view within the leaf's region. To render a view, the nearest leaf is selected and the set of partial views at that leaf are reprojected. This approach results in better rendering performance than the delta tree described above as no redundant pixels are reprojected. We rejected it, however, because the overhead of storing complete sets of partial views at each leaf, even with view sharing, was prohibitive.

We decided not to store multiple samples at each pixel of a view, as is done in [Max95]. Storing multiple z-values per pixel significantly complicates the warper and scan converter. In particular, tests are required to determine which samples at adjacent pixels are on the same surface. Storing multiple samples per pixel also slows rendering as the redundant pixels are reprojected.

At an early stage we considered representing the object by its projection onto a spherical viewing surface with the center of projection within the object. This approach requires multiple samples per point to handle occluding surfaces and fails to sample surfaces with tangents passing through the center of projection. It is also surprisingly difficult to determine the minimum set of samples from such a projection required to reproject an object to an arbitrary viewpoint.

7. Evaluation

Table 1 shows the storage efficiency of the delta tree. The first row corresponds to one primary view of the three-level teapot delta tree one quarter of which is shown in Plate 2. Each unencoded view of the teapot contains 400×400 pixels or 2500 patches. The tree has 81 nodes representing 25 unique views. It stores 6748 patches of which 2087 are shared by more than one node. This corresponds to the number of patches required to hold 2.7 full views giving a compression ratio, before DCT coding, of 9.3:1. A chess piece, the bishop, represented with a three-level tree also gives a compression ration of about 9:1. The bottom row shows the effect of deepening the tree. Expanding the teapot to a four-level tree roughly doubles its storage requirements. While the leaves of the four-level tree take very little storage, about 30 patches per node, the level 3 nodes take more storage than the smaller tree, 87 vs 61 patches per node, as they are not compressed against one another.

Table 1: Storage efficiency of some delta trees

Object	Size	Primary Views	Levels	Nodes	Views	Patches	Shared	Equivalent Views	Ratio
Teapot	400 x 400 (2500)	1	3	81	25	6,748	2,087	2.7	9.3:1
Bishop	256 x 256 (1024)	6	3	486	122	14,473	6,767	14.1	8.7:1
Teapot	400 x 400 (2500)	1	4	337	81	15,124	3,302	6.1	13.3:1

Table 2 shows the number of pixels reprojected to synthesize images of the teapot at various magnifications. At unit magnification about twice as many pixels are reprojected as appear in the final image. This overhead is due to overlap of undersampled surfaces in parent nodes with surfaces at child nodes. With a modest magnification, fewer pixels are reprojected than appear on the screen. The scan converter interpolates between the reprojected pixels to make up the balance. As the object is minified, the number of reprojected pixels remains constant until a power of two breakpoint is reached. At each breakpoint the ratio of reprojected to visible pixels remains about 2.0.

Table 2: Bandwidth required

Mag	Reprojected	Appear	Ratio
0.5	16,560	8,398	2.0
1	66,240	33,622	2.0
1.5	66,240	80,140	0.82

8. Conclusions

We view the delta tree as a critical and enabling technology for the next generation of computer graphics platforms constructed around image-based rendering methods. Delta trees are ideally suited both to the representation of scene components as well as their re-projection and rendering. Every visible portion of an object is represented within a delta tree at multiple scales and with minimal redundancy. The multiple scale property is provided by the frequency domain representation that is used to store the image patches of the object being represented by the tree, and, in the near future, we expect to take further advantage of this frequency-domain representation to provide additional compression of the delta tree.

Future rendering systems based on the delta tree representation will have the advantage of being object-centered rather viewpoint-centered as previous image-based rendering approaches propose. Obviously, the delta tree approach allows objects to be instanced many times within the same scene, or even reused in entirely different scenes. In addition, objects described by delta trees can used as elements of dynamic scenes. But most important, whereas a view-centered image-based rendering approach represents a significant departure from the traditional modeling practices, the delta tree approach maintains much of the same building-block flavor.

While the delta tree data structure has many advantages purely from the modeling perspective, perhaps it greatest utility is the ease with which it allows objects to be rendered. It is straightforward to determine the sufficient subset of a delta tree that is necessary for an arbitrary re-projection of the object that a delta tree encapsulates. Futhermore, a simple traversal of a delta tree along a single path from a root node to one of its leaves provides a complete set of surface patches necessary to render the desired object or portion there of. With an estimate of the projected size of each patch in the output image the delta tree representation allows us to select from the patch only those frequency components that are sufficient to reconstruct the object with appropriate detail and spatial filtering. When an image sample from a delta tree is near the desired view, which is nearly always the case, then the re-projection of the image data involves only small changes in visibility. Thus a delta tree determines the intra-object visibility with a nearly constant depth-complexity close to one. Thus little effort is spent rendering or re-projecting surface elements that are not seen in the final image. This represents a significant advantage over traditional geometry-based rendering. In some sense one could view the delta tree as an abstract form of a volumetric representation in which only those voxel elements that are both occupied and potentially visible are represented, all while maintaining a the spatial coherence necessary for rapid re-projection and rendering. In the near future we hope to explore several enhancements to both the delta tree data structure and its associated traversal algorithms, such as shadow generation and reflectance mapping, in order to enhance its applicability.

As the cost of rendering a scene from a polygonal model continues to increase with the number of primitives in the scene, alternative approaches to computer image generation whose cost is independent of scene complexity become increasingly attractive. Few alternatives show greater promise that the various image-based methods that have been suggested in the last few years. In this paper we have introduced a new variation of this image-based approach, which retains much of the traditional building-block methodology from the modeling perspective while embracing all of the performance and scalability advantages of the new image-based approach.

9. Acknowledgments

Redacted to preserve anonymity of the authors.

10. References

[Blinn76] J. F. Blinn and M. E. Newell, "*Texture and Reflection in Computer Generated Images*," **Communications of the ACM**, vol. 19, no. 10, pp. 542-547, October 1976.

[Blinn78] J. F. Blinn,"*Simulation of Wrinkled Surfaces*," **Proceedings of SIGGRAPH 78,** (Atlanta, GA) August 23-25, 1978, pp. 286-292.

[Catmull74] E. Catmull, **A Subdivision Algorithm for Computer Display of Curved Surfaces** (Ph. D. Thesis), Department of Computer Science, University of Utah, Tech. Report UTEC-CSc-74-133, Dec 1974.

[Chen93] S. E. Chen and L. Williams. "*View Interpolation for Image Synthesis*," **Proceedings of SIGGRAPH 93,** (Anaheim, CA) August 1-6, 1993, pp. 279-288.

[Chen95] S. E. Chen, *"Quicktime VR— An Image-Based Approach to Virtual Environment Navagation,"* **Proceedings of SIGGRAPH 95,** (Los Angeles, CA) August 6-11, 1995, pp. 29-38.

[Crow84] F. C. Crow, "*Summed-Area Tables for Texture Mapping*," **Proceedings of SIGGRAPH 84,** (Minneapolis, Minnesota), July 23-27, 1984, pp. 207-212.

[Duff85] T. Duff, "*Compositing 3-D Rendered Images*," **Proceedings of SIGGRAPH 85,** (San Francisco, Minnesota), July 22-26, 1985, pp. 41-44.

[Heckbert86] P. Heckbert, "*Survey of Texture Mapping*," **IEEE Computer Graphics and Applications**, vol. 6, no. 11, November 1986, pp. 56-67.

[Laveau94] S. Laveau and O. Faugeras, *"3-D Scene Representation as a Collection of Images and Fundamental Matrices,"* **INRIA, Technical Report No. 2205**, February 1994.

[Max95] N. Max and K. Ohsaki, "Rendering Trees from Precomputed Z-Buffer Views," 6th Eurographics Workshop on Rendering, (Dublin, Ireland) June 1995, pp. 45-54.

[McMillan95a] L. McMillan and G. Bishop, *"Head-Tracked Stereo Display Using Image Warping,"* **1995 IS&T/**

SPIE Symposium on Electronic Imaging Science and Techno