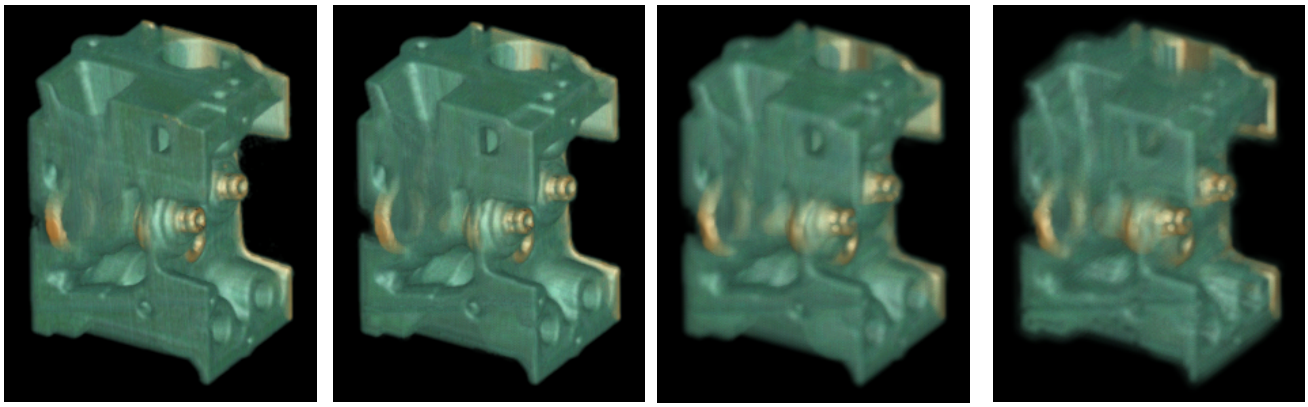
A Frequency-Sensitive Point Hierarchy for Images and Volumes

Tomihisa Welsh Klaus Mueller
Center for Visual Computing, Computer Science, Stony Brook University



(a) Orig. 2,608,070 pts, 9 sec (b) 951,291 pts, 3 sec (c) 195,854 pts, 0.5 sec (d) 79,007 pts, 0.2 sec

Figure 1: The engine dataset (256^3), rendered semitransparent without occlusion culling at various error thresholds with our algorithm.

Abstract

This paper introduces a method for converting an image or volume sampled on a regular grid into a space-efficient irregular point hierarchy. The conversion process retains the original frequency characteristics of the dataset by matching the spatial distribution of sample points with the required frequency. To achieve good blending, the spherical points commonly used in volume rendering are generalized to ellipsoidal point primitives. A family of multiresolution, oriented Gabor wavelets provide the frequency-space analysis of the dataset. The outcome of this frequency analysis is the reduced set of points, in which the sampling rate is decreased in originally oversampled areas. During rendering, the traversal of the hierarchy can be controlled by any suitable error metric or quality criteria. The local level of refinement is also sensitive to the transfer function. Areas with density ranges mapped to high transfer function variability are rendered at higher point resolution than others. Our decomposition is flexible and can be used for iso-surface rendering, alpha compositing and X-ray rendering of volumes. We demonstrate our hierarchy with an interactive splatting volume renderer, in which the traversal of the point hierarchy for rendering is modulated by a user-specified frame rate.

Keywords: volume rendering, point-based rendering, splatting.

1 Introduction

Hierarchies of rendering primitives are desirable, since they allow the renderer to tune the size and number, and therefore the rendering effort, of the rendered primitives to the resolution of the screen (view-sensitive) or the local detail of the object (feature-sensitive) or both. Successful approaches that fulfill both metrics have been developed for polygonal surface-based objects a number of years ago (see Hoppe [1997] and others). One motivation behind the recent trend to point-based surface rendering is that for objects of high geometric complexity a point primitive offers greater simplicity for projection and better delineation of fine object detail than a small triangle [Zwicker et al. 2001a], although hybrid approaches have been proposed [Cohen et al. 2001; Chen and Nguyen 2001]. Similar advantages also exist for point-based volume rendering, often called splatting [Westover 1990], when compared to image-order rendering methods, such as raycasting [Meissner et al. 2000]. However, volume rendering is inherently different from surface rendering (although iso-value volume rendering can also produce the image of a shaded surface). While surface rendering only maintains a collection of points on the surface, volume rendering covers the entire 3D function, supported by a full 3D grid of points, where many points can project and contribute to a single pixel on the screen. On the other hand, the rendering primitives used are similar, both renderers most often represent points as a smooth basis function of radial extent, such as a Gaussian, to provide good blending of the primitives. In surface rendering it is a collection of 2D Gaussians covering the modeled object surface, while in volume rendering it is a collection of 3D Gaussians that blend together to form the 3D function embodied by the grid.

A number of hierarchical point-based surface renderers have been proposed. Most of the earlier ones, such as [Chen and Nguyen 2001; Botsch et al. 2002, Pfister et al. 2000; Rusinkiewicz and Levoy 2000] construct their hierarchies without much consideration of local detail and are mainly view-sensitive, but more recent approaches do perform a feature-sensitive construction of the hierarchy [Dey and Hudson 2002].

While the research in hierarchical point-based surface rendering is numerous, there has not been much work on advancing the concept of hierarchies in the field of point-based volume rendering other than the early work by Laur and Hanrahan [1991]. Most of the recent effort has been spent on using hierarchical representations for the purpose of opacity-based occlusion culling [Lee and Ihm 2000; Mora et al. 2002], while the rendering primitive is still a regular arrangement of points at the original grid resolution.

As in point-based surface rendering, minimizing the number of rendered points while maintaining a moderate amount of overlap for blending will also minimize the time required for rendering. We have already mentioned opacity-based culling as one important

{tfwelsh, mueller}@cs.sunysb.edu, http://www.cs.sunysb.edu/~tfwelsh/Points

means to eliminate the *occluded* points from the rendering pipeline. A feature-sensitive point hierarchy will also reduce the number of *non-occluded* points that need to be rendered. The research reported in this paper is targeted at the latter, by constructing a feature-sensitive hierarchical decomposition of the volume, with capabilities to control the local rendering error on the fly. This enables time-critical rendering as well as view-dependent level of detail, both of which are not possible with a flat volume decomposition. Another important issue, unique to volume rendering, is the existence of transfer functions that map raw volume densities to colors and opacities. We can further reduce the number of rendered points by rendering volume regions that fall into uniform portions of the transfer function at lower resolution.

The basic idea of our work is motivated by the concept of adaptive, or importance sampling. In adaptive sampling, more sample points are placed in image areas with high detail, while less sample points are placed in homogenous regions (see discussion in Glassner [1995], chapters 7 and 9). Since we would like to synthesize a smooth image from this point distribution, we must fill the empty areas between the sample points by a suitable blending mechanism. For this purpose, we represent each sample point as an elliptical Gaussian basis function, each having an extent corresponding to the local point density. This is illustrated in Fig. 7 (left). There are three stages to this process. First, we need to analyze the image/volume to find proper sample locations for the adaptive sampling. We do this by performing a Gabor wavelet decomposition. Second, from the analysis we obtain the decomposition hierarchy which is synonymous with the point placement in adaptive sampling, and assign an error metric to each component or node in the hierarchy. The third stage is the actual image synthesis where we traverse the hierarchy and pick the most appropriate points given the quality metric and other view-dependent and transfer function-dependent metrics, and splat these chosen points into the image. All of these three stages will be described in the following sections, after a discussion on previous work in this area.

2 Previous Work

As far as we know the only space-covering point hierarchies constructed were in the form of quadtrees and octrees. However, an undesirable property of quadtrees (octrees) is that a subdivision always gives rise to 4 (8) children (as shown in Fig. 2a). A more generalized point hierarchy, as the one shown in Fig. 2b, would allow for a lesser amount of children to be expanded. This reduction of nodes can have important implications on storage and rendering complexity. The hierarchical splatting technique proposed by Laur and Hanrahan [1991] for volume rendering is based on an octree and stores the root mean square error in each node. This measure is then compared with an error threshold during traversal to decide if the node should be expanded or not. The error approximates the cost of rendering a region at the resolution of the hierarchy level and the root mean square error metric is related to the frequency content at that location.

The discrete wavelet transform offers a more versatile and rigorous framework for frequency analysis, and a number of visualization researchers [Guthe et al. 2002; Muraki 1994; Westenberg and Roerdink 2000, Westermann 1994] have used this transform to find a decomposition. In this case, the basis functions are given by the scaled wavelets. A wavelet-based decomposition, with only the significant nodes shown, may look like the one given in Fig. 2c. Note that unlike the quadtree, where only the bottom nodes (the black ones) of the expanded tree have to be used for signal synthesis, in the wavelet transform all nodes with a significant coefficient (the black, blue, and red ones) have to be added, with potentially large overlap among the basis functions. This does not fulfill our minimum overlap criterion. In fact, the large extent of the basis functions encoding the low signal frequencies has prevented the use of wavelet splatting for anything more than X-ray projection, since the spatial order of the basis functions required for compositing does not exist.

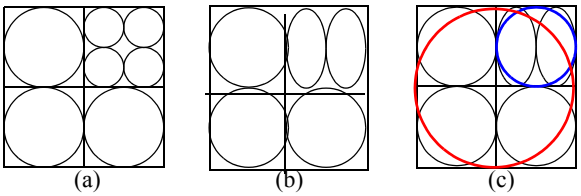


Figure 2: Hierarchical point decompositions: (left) quadree; (middle) generalized; (right) wavelet (overlapping primitives).

A very different strategy to find an image decomposition into radial basis functions is to use numerical optimization. A list of available techniques includes the conjugate gradient method, nonlinear least squares, such as the Levenberg-Marquardt algorithm, or simulated annealing [Press et al. 1988]. The latter was recently used by [Kreylos, B. Hamann 2001] to find a (piecewise linear) polygonal decomposition of an image. Childs et al. [2000] employed a modified version of the Marquardt algorithm to decompose images into a set of elliptical Gaussians. Although their implementation, termed Quickstep, is an order of magnitude faster than the Marquardt algorithm it still takes about one hour to find the decomposition of the Lena image (256^2), which will probably make the algorithm infeasible for volumes. The compression rates are rather impressive, but a drawback of the resulting assembly of Gaussians is that they will only yield a correct image when splatted into the same grid that was used to evaluate the minimization metric. In other words, the continuous function generated by the field of Gaussians is not smooth and can have sharp positive and negative spikes at off-grid positions, which would be disastrous for splat-based volume rendering or image magnification.

3 Overview

For a given dataset, the hierarchy consists of a number of fixed elements similar to a quadtree for images, or an octree for volumes. Thus, the basic structure is a tree with M levels, and each node has a fixed degree 2^D where D is the dimensionality of the dataset. Nodes higher in the tree represent coarser features while leaf nodes (level 0) represent the original grid samples of the dataset. The main difference between our representation and that of a quadtree or octree is that nodes above the leaf nodes are not constrained to represent regular data points but can contain irregular samples as well. In fact, each node contains a different set of primitives (not necessarily 2^D) chosen from a fixed set of possibilities much like a basis set. Fig. 4 shows the point primitives for the 2D case. Here, we see that a given node will maximally use four point primitives. However, by using horizontal or vertical splats we can cut this number in half. In the case of volumes, a node will maximally use up to eight points but 1/4 as many for the horizontal or vertical representation. Thus, by combining the structure of an octree with irregular sampling we are able to increase compression while retaining the traversal speed afforded by the octree-based space decomposition.

There are two main steps in building the point hierarchy. First, the analysis stage involves decomposing the dataset into individual frequency components (scale and orientation) using the Gabor wavelet family. We use this information to build the fundamental types of point primitives (elliptical Gaussians) and their location for a given level in the hierarchy (scale level). More specifically, we locate scale-space regions where features are oriented in a single direction. For these regions we direct sampling in the optimal direction and thereby reduce the number of samples required to capture that feature. This potentially can eliminate a large number of unnecessary samples at each level in the hierarchy while the

structure of the tree remains fixed (each node has 2^D children).

Second, we assign an error metric to each component (node) in the hierarchy. This error value is used later during rendering to determine how far in the hierarchy tree we must traverse from the coarsest, top level towards the finer, bottom level while still capturing the important features of the data. This stage allows a further reduction of points because we are able to render fewer, larger points in regions when there is a low error associated with nodes that are higher in the hierarchy.

The rendering stage involves traversing the hierarchy and splatting the nodes which fall below a threshold criterion. For images, the elliptical Gaussian splats for each node are accumulated in an image buffer along with the weights of these splats. The final image is then normalized by dividing the image buffer with the weight buffer. For volumes, we have implemented rendering using a pre-shaded splatting technique using texture mapping which does not normalize the image buffer [Mueller et al. 1999].

4 Frequency Analysis

In this section we describe the analysis of the dataset's frequency characteristics using Gabor wavelets. The goal is to gather information in both the space and frequency domains to determine the proper sampling rates for different portions of the dataset. The end result is a decomposition of the 2D image or 3D volume into a hierarchy of multiresolution scales and orientations.

4.1 Gabor Wavelet Decomposition

Gabor functions are well suited for the task of extracting oriented, multiresolution features from images and volumes. In particular, Gabor wavelets are attractive for our goal of combining spatial and frequency information. A well known property of these functions is that they provide the best theoretical trade-off between space and frequency resolution [Daugman, 1988].

Gabor functions are created by modulating a Gaussian function with complex sinusoids. In its general form, the 2D Gabor function is written [Daugman, 1988; Manjunath and Ma 1996] as:

$$g(x, y, u_0, v_0) = \exp\left[-\left[\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right] + 2\pi i[u_0 x + v_0 y]\right] \quad (1)$$

where σ_x and σ_y are the extents of the Gaussian in the spatial domain and (u_0,v_0) is the frequency of the complex sinusoid. In the frequency domain, the Gabor function is simply a Gaussian function scaled differently in the x and y directions:

$$G(u, v) = e^{-2\pi^2(\lambda^2(u'-u_0')^2 + (v'-v_0')^2)}$$
$$u' = u\cos\theta + v\sin\theta \qquad v' = -u\sin\theta + v\cos\theta \quad (2)$$

where λ is the aspect ratio σ_y/σ_x. $(u_o', v_o') = (\omega_c\cos\theta, \omega_c\sin\theta)$, ω_c is the central frequency value and θ is the central direction. Thus, to create the full family of oriented, multiscale wavelets we just rotate and dilate this mother wavelet in the frequency domain.

We set N=4 possible orientations and set the number of levels M in the hierarchy to depend on the dataset. More specifically:

$$\theta = \frac{n\pi}{N},\ 1 \le n \le N \qquad \Delta w = 2^{-m-2},\ 1 \le m \le M$$
$$\omega_c = 2^{-m} - \Delta w \qquad \lambda = \frac{2\Delta w(k-1)}{\pi\omega_c} \quad (3)$$

Here, n and m are the direction and scale values for a given wavelet function. These parameters create the decomposition shown in Fig. 3a. Thus, we are able to capture horizontal, vertical and diagonal orientations. We should note that by using a linear combination of these values it would be possible to further localize

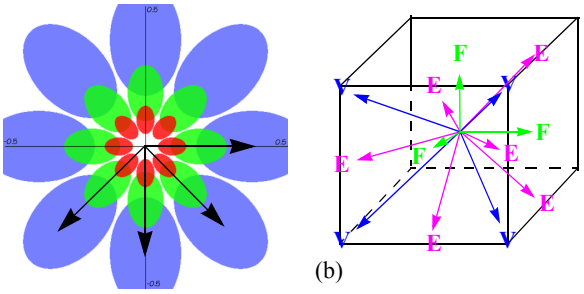


Figure 3: The Gabor wavelet family coverage in the frequency domain (a) for N=4 (orientations) and M=3 (scales) in 2D and (b) for a single level (N=13, M=1) in 3D. The 13 quantized directions in 3D are represented as faces (F), edges (E) and vertices (V) on a cube. Arrowheads also indicate the direction in which the point primitives are flattened.

features between these major directions. However, as this would have adverse effects on storage, we only use quantized directions.

For volumes, we quantize directions to the 13 edges, faces and vertices of a cube (see Fig. 3b). In spherical coordinates these correspond to the directions $(\phi, \theta) = \{(-90,0), (-45,0), (-45,-45), (-45,-90), (-45,-135), (0,0), (0,-45), (0,-90), (0,-135), (45,0), (45,-45), (45,-90), (45,-135)$ in the frequency domain. The 3D version of equation (2) is:

$$G(u, v, w) = e^{-2\pi^2(\lambda^2(u'-u_0')^2 + (v'-v_0')^2 + (w'-w_0')^2)}$$
$$u' = u\cos\theta\cos\phi - v\sin\phi + w\sin\theta\cos\phi$$
$$v' = -u\cos\theta\sin\phi + v\cos\phi + w\sin\theta\sin\phi \quad (4)$$
$$w' = -u\sin\theta + w\cos\theta$$

The Gabor wavelet transform can be performed in the spatial domain as a set of separable 1D convolutions [Loy 2002], however for simplicity, we extract our wavelet decomposition using multiplication of the Gabor function with the image/volume in the frequency domain. We use the Fast Fourier Transform (FFT) to transform the data into the frequency domain, perform the multiplications with the Gabor functions, and then transform each result back into the spatial domain using the inverse FFT operation

4.2 Point Primitives

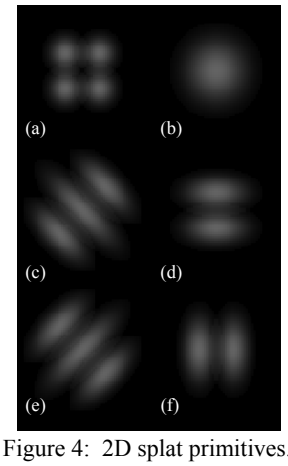


Figure 4: 2D splat primitives.

For images, the point primitives are 2D elliptical Gaussians oriented in four quantized directions and scaled to any size. Fig. 4 shows the six different splat types that can occur in 2D: two isotropic and four anisotropic. The four primitives on the top left corner of the figure represent the original samples of image. The single large splat on the top right is used when no significant features exist for a given node. The anisotropic primitives are used when features are directed along a single direction. The orientation of the flattening corresponds to the arrow directions in Fig. 3a. The splat primitives in Fig. 4c-f correspond to the arrow directions in Fig. 3a, counterclockwise. In both images and volumes, the number and orientation of the points is designed to fully cover each node in the hierarchy spatially while limiting the sampling rate along the direction orthogonal to the feature. For example, if only a strong horizontal

feature exists for a given node in an image then the feature can be represented with only two thin splats (Fig. 4f). For diagonal features, the diagonal length $\sqrt{2^2+2^2} \approx 2.83$ of a node conservatively requires 3 directed splats as shown (Fig. 4c,e). The size of the splats within a node are adjusted in order to spatially cover the node while minimizing overlap with neighboring nodes. It should be noted that if one normalizes the splatted image by the sum of weights, such that overlap between neighboring splats is corrected for, then the precise size and position of the diagonal splats is not important.

For volumes, there are 15 splat primitive sets: two isotropic splat sets and 13 anisotropic splat sets directed along quantized directions of a cube depicted in Fig. 3b. A cube has 6 faces, 8 vertices and 12 edges but by symmetry there are only 13 relevant directions. The 3 face directions can be covered with two large, pancake-shaped splats (analogous to the 2D case of Fig. 4d,f), while the 6 edge and 4 vertex directions can be covered with three splats flattened perpendicular to the direction vector, placed equidistant from each other (analogous to the 2D case of Fig. 4c,e). In addition, there is the single large-splat configuration and the standard 8-small-splats configuration. In practice, we have found that it is better to use 4 elongated cigar-shaped splats for the edge directions when normalization is not used in the volume rendering.

Note that the average occupancy of all of these configurations is less than 4 splats, which is considerably smaller than the standard 8-splat case. In conjunction with the fact that these lower-occupancy configurations can be tuned reasonably well to the local features, we expect a high utility during rendering and therefore considerable savings.

4.3 Selecting Node Types

For images, each node is assigned one of the 6 types shown in Fig. 4. For volumes, each node is assigned one of 15 possible types (single splat, 8 small splats or one of 13 possible directed splats). Based on the construction of the Gabor wavelet family, each node will have 4 Gabor coefficient values for images and 13 coefficients for volumes to examine. We use a simple algorithm to determine whether any directed feature exists for that node:

```
if (foreach direction i, coeff_i < t_1)
then  nodetype="single large splat"
else if
     foreach direction i
          foreach direction j not neighboring i
             coeff_i > coeff_j (i is the strongest direction)
             and coeff_j < t_2 (j is not too large)
             and coeff_i/coeff_j > t_3 (i is comparably larger than j)
then   nodetype="directed towards i"
else   nodetype="maximum number of small splats"
```

Thus, if no large Gabor coefficients exist, one large splat is sufficient. If, however, there is a large, dominant coefficient and no other large *non-neighboring* coefficients then we select a directed splat. By only considering non-neighboring directions we take care of the possibility that there could be a feature oriented between two quantized directions, which would give rise to large coefficients at both of these orientations. In our experiments we have found that thresholds around t_1=2,t_2=10 and t_3=0.05 worked reasonably well.

4.4 Assigning Error Values to the Hierarchy

The final pre-processing step involves assigning an error value to each node in the hierarchy. It is this error which is used during rendering to determine whether a node should be rendered (because it is below a user-defined threshold) or whether the hierarchy needs to be traversed more (the error associated with splatting at a course resolution is too large). We examined a number of error metrics including: (i) the maximum Gabor coefficient at a given node, (ii) the root mean square error used by Laur and Hanrahan which assumes the rendering primitive is a spherical Gaussian, (iii) a direct measure of the difference between a node splatted using its assigned point primitives and the original volume.

The first method involves selecting the largest coefficient from the frequency decomposition generated by the Gabor wavelets for each node in the hierarchy. Clearly, we expect large coefficients to be associated with important features. On the other hand, since we traverse the hierarchy in a top-down fashion during rendering, we do not want to skip important nodes low in the hierarchy by selecting nodes above them before examining the entire hierarchy. Thus, we store for each node both the maximum error value (i.e., the coefficient) from the Gabor wavelet analysis and the value of the highest coefficient below it in the tree. We render nodes with a high coefficient value. The traversal algorithm is:

```
TraverseNode(node){
  if ( (node.coeff >= threshold and node.max_coeff_below <= threshold)
       or node.isLeafNode)
  then   render current node    else   traverse child nodes   }
```

This traversal method makes sense when synthesizing a single image or a "cut" across the hierarchy [Leven et al. 2002]. It is a conservative approach in that it will never cull nodes below a given node if the children nodes contain important high frequencies.

Although the traversal method using the Gabor coefficients directly produces acceptable results, there are at least two reasons for considering other error metrics. First, the Gabor wavelets were built in the frequency domain so there is always some amount of trade-off of spatial resolution for frequency resolution. In other words, the coefficients are somewhat susceptible to ringing artifacts and not always spatially precise. Also, it should be observed that we cannot expect gradual changes in the point reduction when the threshold is changed. For instance, when we raise the threshold, there can be a sudden drop in the number of rendered points when many coefficients in the lower hierarchy levels fall below the threshold. This lack of level of detail control leads to unexpected, irregular changes in the rendered image as the threshold changes.

A second error metric that we implemented is the root mean square error used by Laur and Hanrahan:

$$e_j^l = \sqrt{\sum s_i^2 + \left(\sum s_i\right)^2} \tag{5}$$

where e_j^l is the error associated with node j at level l, s_i is the value of the voxel sampled on a volume pyramid and each of the summation terms is over the entire node region. This metric effectively measures the variation for a node at the resolution of its pyramid level. Although this metric can be computed relatively quickly, it assumes a constant function throughout the node.

Because our point hierarchy uses irregular samples within a node, we found it more suitable to use a metric which is sensitive to the specific manner in which the point hierarchy was built. Ideally, we should use a metric which is sensitive to errors associated with the final rendered image. However, this is infeasible since the final choice of splats will be dynamic and unknown before rendering. Our simple approach is to splat each level of the hierarchy into a 2D or 3D image buffer and to assign an error to each node based on the difference between the original image and the splatted image at a given level. Although this approach is computationally slow, it produces the best results. It has an advantage over using the Gabor coefficients directly when an interactive traversal mechanism is required because the threshold is directly proportional to image quality. The traversal algorithm then becomes:

```
TraverseNode(node){
  if ( (node.error < threshold   or  node.isLeafNode)
  then   render current node    else   traverse child nodes   }
```

Note that we still need to use the Gabor coefficients to determine splat orientations. Had we used a slow, brute force technique and simply tested the error for each of the 15 possible configura-

tions for each node, the error values would only be able to reflect properties local to the node and not take into consideration the global frequency characteristics due to overlapping nodesRendering 2D images using elliptical splats is straightforward and can be implemented efficiently in software [Heckbert 1989]. The only major difference is that a final normalization step is required due to the irregular point arrangement. Normalization is performed by dividing the value buffer by a weight buffer, which is obtained by splatting the points with a value of 1.0.

5 Rendering

Rendering in 3D is a much more difficult task due to computational complexity. A further complication is the issue of how to composite semi-transparent samples before the normalization step can be applied. One solution would be to use a slice-based splatting approach [Mueller 1999] where each sheet buffer is normalized before shading and compositing. However, for better speed we chose to implement the simpler composite-only pre-shaded splatting algorithm by Westover [1989] which renders each splat as a whole and not in sections. The lack of interpolated volume sheets, however, makes normalization infeasible, which, to our surprise, has not compromised image quality significantly beyond the usual blurring associated with this type of algorithm (see Results).

5.1 Creating 3D Elliptical Gaussians

The shape of a 3D elliptical Gaussian can be modeled via the implicit equation of an ellipsoid (quadric surface):

$$f(x,y,z) = ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2gx + 2hy + 2jz + k = 0$$

Since we assume that the ellipsoid is centered about the origin for a given splat and of unit size (k=1), we let g=h=j=0 and drop the last four terms. We can represent this ellipsoid using matrix notation [6], giving rise to a 3x3 quadric matrix Q. With this representation we can create arbitrarily oriented and scaled ellipsoids by applying the desired 3D affine transformation contained in matrix M with the formula:

$$Q = \begin{bmatrix} a & d & f \\ d & b & e \\ f & e & c \end{bmatrix} = (M^{-1})^T \cdot I \cdot M^{-1} \qquad (6)$$

where I is the identity matrix. In our application, a non-uniform scale followed by a rotation will generate each of the 13 configurations (for each scale). We can pre-multiply M by the viewing matrix V before application of (6) to obtain the screen space ellipsoid E. To obtain the 2D screen projection of E we can simply drop its last row and column to obtain E's 2D screen-space ellipse E'.

For orthographic rendering we can pre-compute E' for all cases and scales before the frame is rendered (the hierarchy only stores the ID for the case and scale per splat). For perspective rendering, V changes with distance from the screen and therefore we need to compute E' per splat [Zwicker et al. 2001b].

5.2 Texture Mapping the Ellipse

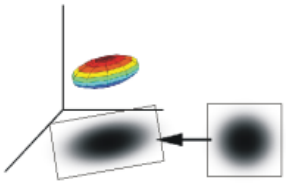


Figure 5: Mapping a unit 2D Gaussian function to ellipsoid E's screen-space elllipse E'.

Next we need to determine the function that maps the unit Gaussian function stored in the graphics board's texture map to the ellipse represented by E' (see Fig. 5). We can perform this affine mapping by simply scaling a unit circle in two orthogonal directions by factors s_x and s_y and then rotating the stretched circle by an angle θ to form the appropriate ellipse. A procedure that achieves this is described in Westover [1990], but we noticed that the derivation contains two crucial (possibly typographic) errors and thus we outline our own derivation in closer detail here.

Starting from the identity matrix we solve for the three unknown parameters s_x, s_y, θ, using equation (6) with the parameters of E' and scale and rotation matrices S and R:

$$E' = ((S \cdot R)^{-1})^T \cdot I \cdot (S \cdot R)^{-1} = R \cdot (S^{-1})^2 \cdot R^T \qquad (7)$$

Plugging in the unknown parameters s_x, s_y, θ we get:

$$E' = \begin{bmatrix} a & d \\ d & b \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} 1/s_x^2 & 0 \\ 0 & 1/s_y^2 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$$a = \frac{\cos^2\theta}{s_x^2} + \frac{\sin^2\theta}{s_y^2}, \qquad b = \frac{\sin^2\theta}{s_x^2} + \frac{\cos^2\theta}{s_y^2} \qquad (8)$$

$$d = \frac{\sin\theta\cos\theta}{s_x^2} + \frac{\sin\theta\cos\theta}{s_y^2}$$

These are the terms that are incorrectly reported by Westover [1990]. After some algebraic manipulation we get:

$$\frac{a-b}{d} = \frac{\cos\theta}{\sin\theta} - \frac{\sin\theta}{\cos\theta} \qquad (9)$$

When d=0, the mapping is determined by scaling factors $s_x = \sqrt{1/a}$, $s_y = \sqrt{1/b}$ and θ=0. When d is not 0, let:

$$G = \frac{(a-b)}{d}, w = \frac{\cos\theta}{\sin\theta}$$

$$G = \left(w - \frac{1}{w}\right), \text{ or equivalently } (w^2 - Gw - 1) = 0 \qquad (10)$$

$$w = \frac{G \pm \sqrt{G^2+4}}{2}, \theta = arctan\left(\frac{1}{w}\right)$$

We can now use equation (8) to compute s_x, s_y. This gives us all parameters of S and R to transform the four vertices of the polygon that maps the unit Gaussian texture to the screen.

5.3 Hierarchy Traversal and Image Synthesis

We now describe the final traversal algorithm for our interactive volume renderer. In this section we assume that method 3 (difference metric) is used to obtain the node error.

Sensitivity to uniform transfer function regions. By making our traversal algorithm sensitive to uniform regions in the transfer function we can further reduce the number of rendered points (provided such uniform areas exist). When the corresponding density values are spatially localized then we can represent the same function with fewer, larger splats using our hierarchy. We are able to make our traversal algorithm sensitive to the transfer function by ways of a simple procedure. First, we build a 2D table TF[0-255][0-255] storing the variability of the transfer function for all of the quantized range intervals. For instance, we store the difference between the minimum and maximum values for ranges: {0-5},{0-10}, ... {0-255}, {5-10},{5-15}, ...{200-255}. Second, as a preprocessing step, we store at each node in the hierarchy the range of values of the volume v_{min}, v_{max} for the spatial region covered by that node. Thus, during traversal we have an additional error metric to consider, that is, TF_{var}=TF[v_{min}][v_{max}], which indicates the variability of the transfer function for that node. If TF_{var} is zero or below a threshold for a given node then there is no need to traverse the hierarchy further.

Smooth transients for speed and quality. Because our goal is to create an interactive system in which traversal depths between frames are expected to vary, we need to consider mechanisms to

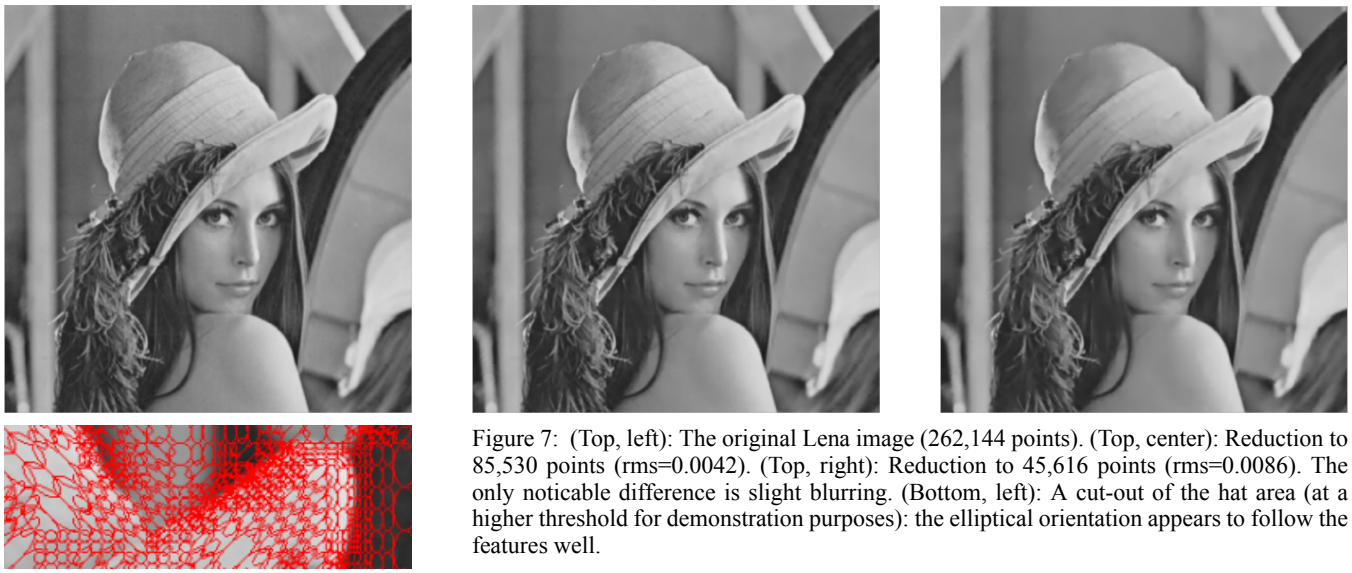


Figure 7: (Top, left): The original Lena image (262,144 points). (Top, center): Reduction to 85,530 points (rms=0.0042). (Top, right): Reduction to 45,616 points (rms=0.0086). The only noticable difference is slight blurring. (Bottom, left): A cut-out of the hat area (at a higher threshold for demonstration purposes): the elliptical orientation appears to follow the features well.

enforce smooth continuities between frames. Due to the manner in which the hierarchy is built, it is possible for scale size or resolution between a parent node and its child to differ by up to two levels (a parent node could be rendered as one large splat at size level i while its child might consist of 8 small splats at level i-1). Also, image quality tends to make large jumps when the threshold changes between certain critical regions. Fig. 6 (pink line) demonstrates this effect. The plot shows the number of points being rendered as a function of increasing threshold. The sharp drops in the line indicate these critical regions where we suspect the traversal mechanism introduces discontinuities in rendering quality.

In order to lessen these effects, we introduce another layer to our node hierarchy for single large splats. This is actually just the large splat primitive type that we use to build our hierarchy, but we add an additional error term for rendering this type at each node. Thus we simply add another error test during traversal and do not need to store any additional information. Fig. 6 (blue line) shows how this simple modification reduces the large discontinuities in the number of points being rendered. Note, the blue line is lower than the pink line because less, larger splats are being utilized. However, image quality falls at a greater rate.

Final traversal algorithm. Our final traversal algorithm is:

```
TraverseNode(node){
  if (node.error < thresh or TF_var < tf_thresh or node.isLeafNode)
  then   render current node
  else if (node.error2 < threshold (error for rendering large splat))
  then   render single large splat
  else   traverse children nodes  }
```

The final splat value is obtained by sampling a Gaussian pyramid. Also, because we render using compositing, we must order our splats from front to back. This is achieved using a standard bucket sorting algorithm which is performed before a view is rendered. Once the splats are properly ordered we send them to the video card where they are composited using OpenGL texture mapping and lighting.

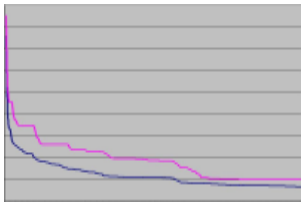


Figure 6: Number of rendered points as a function of threshold value. The pink line shows rapid changes at critical points. The blue line is the result after modifying the traversal algorithm as described in the text.

6 Results

6.1 Images

We implemented a 2D software splatter to demonstrate the results of applying our algorithm to images. Fig. 7 shows a point reduction for the Lena image of 1/3 and 1/5 of the original number of points. At 1/5 of the original number of points, only very slight smoothing can be perceived. Looking closely at the shape of the points (bottom left image), we observe that the frequency analysis seems to be quite successful in placing oriented splats close to the appropriate features. Note that for this image the threshold was set very high, to bring the number of points down to a visibly differentiable level. We should mention that we do not compare our results to standard image compression algorithms such as JPEG, since the main goal of our system is to create a point hierarchy system and not image compression.

6.2 Volumes

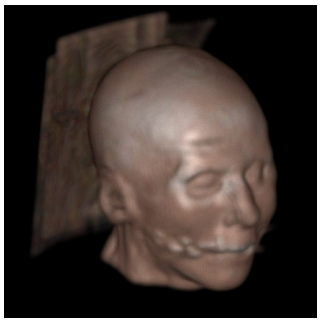
In this section we show the results of applying our algorithm to volumes. Results were generated on a Pentium 4 (2.0 GHz) system with an nVidia Geforce4 4400 Ti graphics board and 1 GB of main memory. All voxels (including the original reference volume) were thresholded above values of 5 to eliminate “air” voxels. Also, no occlusion culling was used for all our experiments to show the pure speedup facilitated by our flexible point hierarchy.

Fig. 1 presents the 256^3 engine volume, rendered semi-transparently as a point cloud with no opacity-based gradient modulation. This example demonstrates the advantage of using a minimized point hierarchy when the goal is interactive rendering speeds. The original dataset uses over 2.6 million points and takes 9 seconds to render. However, a very high quality image (b) can be obtained with less than 1 million points and can be rendered in 3 seconds. Also, at near interactive frame rates, the volume only looks blurry but still retains all the important features (c, d).

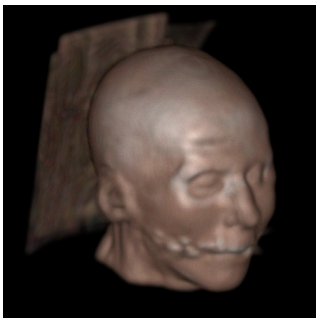
Fig. 8 compares the results of using the three different error metrics that we have described in Section 4.4. The original image is shown on the left. Image (b) uses the default error metric (method 3) which is just the 3D difference between the original volume and the reduced point representation for the node. Image (c) uses the root mean square error used by Laur and Hanrahan. This error is faster to compute and good in quality but there is some image quality loss. For example, observe the loss of detail above the eye brow. Next, (d) shows the image generated using the Gabor coefficients directly. Although the quality is good, there are

many more points being used and there is also a lack of range in the compression level. Finally, (e) shows similar loss of detail around the brow when the full hierarchy is rendered with 8 spherical splats per node but using method 3. This indicates that by using oriented primitives we are able to render higher quality images even when the error metric and all other factors remain the same. Note that the octree decomposition of Laur and Hanrahan is a subset of our decomposition. To fit their approach, each of our feature-aligned elliptical splats would have to be substituted by a set of spherical splats to achieve the same error and therefore more primitives would have to be rendered.

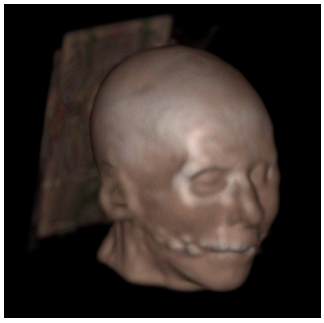
Fig. 9 (left column) shows another example of a volume rendering, a semi-transparent foot. Using our decomposition we are able to get interactive framerates with minimal loss of image quality. Finally, Fig. 9 (right column) shows the results of our implementation for X-ray rendering. Here we are using a software implementation since the limited framebuffer precision does not allow the accumulation of X-ray images in hardware. We are however, able to perform normalization by a separately accumulated weight image, since X-ray performs order-independent summations. We achieve interactive frame rates at the expense of image blurring. Image (a) is the original dataset. Image (b) uses our technique with 1/4 compression, and we observe only slight quality degradations. Image (c) shows that at 1/10 compression overall structure is still present, but at a considerable amount of blurring.



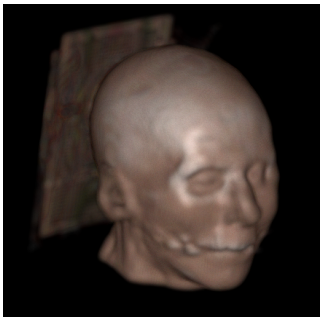
(a) (Original) 1,779,610 pts



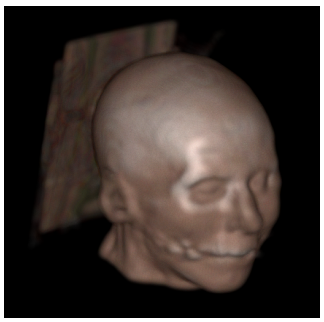
(b) Method 3 (877,310 pts)



(c) Method 2 (851,690 pts)



(d) Method 1 (1,110,680)



(e) Full Hierarchy (882,292 pts)

Figure 8: The CT head dataset (128^3) rendered with the different error metrics for hierarchy traversal (a-d). (e) This image was generated using the method 3 error metric but is rendered with 8 spherical points per node. This directly demonstrates the advantage of using splat primitives which are chosen based on frequency characteristics of the volume.

Preprocessing time for the Gabor wavelet decomposition is currently the most lengthy step largely because it was implemented using the FFT rather than using a recently reported technique that employs a linear filter in the spatial domain [Loy 2002]. A 256^3 volume was processed in 195 min. using MATLAB. The preprocessing time for building the node hierarchy using method 3 was 7 min. for a 128^3 volume and 156 min. for a 256^3 volume. However, building the hierarchy using method 1 and method 2 was only a few seconds for a 128^3 volume.

The hierarchy is implemented without pointers so each node stores only the type of primitive (1 byte) and two floating point error values (8 bytes). This allows the volume values (1 byte per point) and gradient (quantized to 2 bytes per point [Rusinkiewicz and Levoy 2000]) to be stored in a separate full data pyramid. The location of each point is determined implicitly from the node location and type at rendering time in order to minimize the storage costs of the hierarchy.

6.3 Time-Critical Volume Rendering

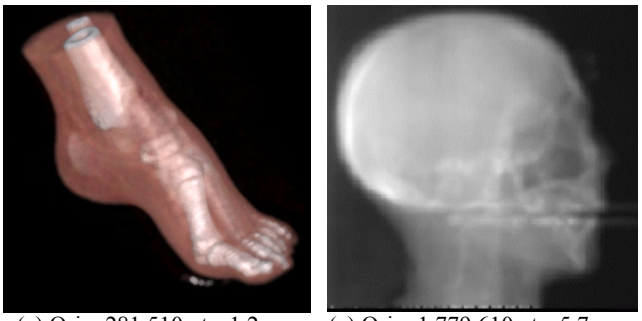
We have implemented a rendering system which permits time-critical volume rendering and progressive refinement of the image. Based on a history of previous frame-rates, the system adjusts the error threshold to achieve a specified frame rate to maintain a user-selected frame rate. When there is no motion, the system progressively refines the image by lowering the error threshold and consequently increasing the number of points in the representation. Please see the conference DVD or project webpage for the video.

7 Conclusions and Future Work

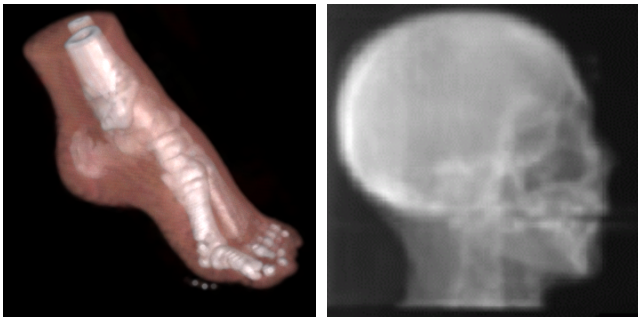
We have introduced a method for converting an image or volume sampled on a regular grid into an irregular point hierarchy. The conversion process retains the original frequency characteristics of the dataset while minimizing the number of points. The basic approach is straightforward. We use the Gabor wavelet to analyze the frequency characteristics of the source dataset at varying scale-spaces. The analysis is used to determine the size, orientation and location of points which can be employed to reconstruct the original signal using an elliptical Gaussian splatting kernel. Our technique produces high-quality reconstructions of the original data for both 2-D images and 3-D volumes, and also allows for time-critical rendering with progressive refinement.

Although our technique works well for finding the initial size and placement of points, we believe that the quality of the image could be further improved by using a less quantized point position and orientation. A potential solution might involve using an error metric to guide point placement and resizing the points with a numerical optimization algorithm. However, in order to avoid the large computation times that global optimization solutions require, we believe a local optimization policy is necessary. Also, such a technique would result in a trade-off of computational speed and space efficiency. We have, however, experimented with an algorithm that flattens larger points based on local gradients and point occupancies in the lower hierarchy levels.

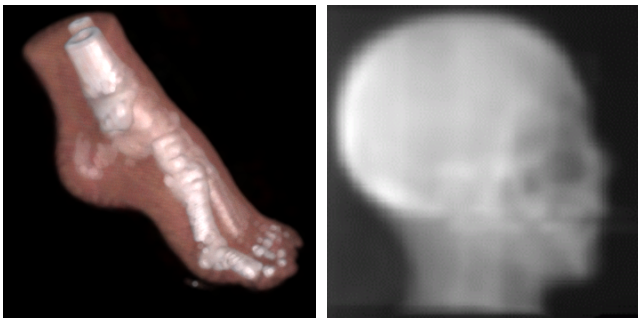
We also hope to increase the image quality by using a slice-based splat rendering algorithm [Mueller 1999]. The current method of rendering via compositing-only does not permit normalization of the weights. Using slice-based splatting, we could normalize each image sheet after accumulating splats values and weights. This would result in fixing some of the artifacts caused by rendering variably-sized overlapping splats. If such a technique were implemented on graphics hardware, there would be little loss in speed. We would also like to test our hierarchy to decompose other, more efficient regular grids, such as the BCC grids [Neophytou and Mueller 2002]. Finally we believe we can save substantial amounts of memory by quantizing the error metric.



(a) Orig. 281,510 pts, 1.2 sec (a) Orig. 1,779,610 pts, 5.7 sec



(b) 107,708 pts, 0.3 sec (b)551,950 pts, 2.4 sec



(c) 76,038 pts, 0.2 sec (c) 76,038 pts, 0.2 sec

Figure 9: (Left column): The foot dataset (128^3) rendered at different error thresholds. (Right column): X-Ray rendering of the CT head at different error thresholds.

While it is true that today's texture mapping hardware is heavily optimized to render large blocks of 2D and 3D textures, the continued rise in the popularity of point-based surface representations will undoubtedly lead to hardware that can also render point primitives at high rates. Once this hardware is available, we believe volume point approaches such as ours will become more competitive for a much broader range of volume datasets and applications.

Acknowledgement

We would like to thank the anonymous reviewers for their helpful ideas and comments. This work was supported by NSF Career grant ACI-0093157.

References

BOTSCH, M., WIRATANAYA, A., KOBBEL, L., "Efficient High Quality Rendering of Point Sampled Geometry", *Proc. of the 13th Eurographics Workshop on Rendering Techniques*, Pisa, Italy, pp. 53-64, 2002.

CHEN, B. AND NGUYEN, M., "POP: A hybrid point and polygon rendering system for large data," *Proc. Visualization'01*, pp. 45-52, 2001.

CHILDS, J., LU, C.-C. AND POTTER, J., "A fast, space-efficient algorithm for the approximation of images by an optimal sum of Gaussians," *Graphics Interface'00*, 2000.

COHEN, J., ALIAGA, D. AND ZHANG, W., "Hybrid simplification: Combining multi-resolution polygon and point rendering," *Proc. Visualization'01*, pp. 37-44, 2001.

DAUGMAN, J., "Complete discrete 2-D gabor transforms by neural network for image analysis and compression," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 36, no. 7, pp. 1169--1179, 1988.

DEY, T. AND HUDSON, J., "PMR: Point to mesh rendering: a feature-based approach," *Proc. IEEE Visualization*, pp. 155-162, 2002.

FOLEY, J.D., VAN DAM, A., FEINER, S.K., AND HUGHES., J.F., *Computer Graphics: Principles and Practice.* Addison-Wesley, 1996.

GLASSNER, A., *Image Synthesis*, Morgan-Kaufman, 1995.

GUTHE, S., WAND, M., GONSER, J., AND STRASSER, W., "Interactive rendering of large volume datasets," *IEEE Visualization 2002*, pp. 53-60.

HOPPE, H., "View-dependent refinement of progressive meshes.", *ACM SIGGRAPH 97*, pp. 189-198, 1997.

HECKBERT, P., *Fundamentals of Texture Mapping and Image Warping*, M.Sc. Thesis, University of California, Berkeley, June, 1989.

LEE, R. AND IHM, I., "On enhancing the speed of splatting using both object- and image space coherence," *Graphical Models and Image Processing*, vol. 62, no. 4, 2000. pp 263-282.

KREYLOS, O., HAMANN, B., "On simulated annealing and the construction of linear spline approximations for scattered data," *IEEE Trans. Visualization and Computer Graphics*, vol. 7, no. 1, pp. 17-31, 2001.

LAUR, D. AND HANRAHAN, P., "Hierarchical splatting: A progressive refinement algorithm for volume rendering," *ACM SIGGRAPH 91*, pp. 285-288, 1991.

LEVEN, J., CORSO, J., KUMAR, S. AND COHEN, J., "Interactive visualization of unstructured grids using hierarchical 3D textures," *Proc. Symposium on Volume Visualization and Graphics 2002*. pp 33-40, 2002.

LOY, G., "Fast Computation of the Gabor Wavelet Transform," *Proc. of Digital Image Computing - Techniques and Applications (DICTA2002)*, Melbourne, January 2002.

MANJUNATH, B.S. AND MA, W.Y., "Texture features for browsing and retrieval of image data", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol.18, no.8, pp.837-42, Aug 1996.

MEISSNER, M., HUANG, J., BARTZ, D., MUELLER, K., CRAWFIS, R.,"A practical comparison of popular volume rendering algorithms," *Symposium on Volume Visualization and Graphics 2000*, pp. 81-90, 2000.

MORA, B., JESSEL, J., CAUBET, R., "A new object-order raycasting algorithm," *Proc. IEEE Visualization 2002*, pp. 203-210, 2002.

MUELLER, K., SHAREEF, N., HUANG, J., CRAWFIS, R., "High-quality splatting on rectilinear grids with efficient culling of occluded voxels," *IEEE Trans. on Vis. and Comp. Graph.*, vol. 5, no. 2, pp. 116-134, 1999.

MURAKI, S., "Multiscale 3D edge representation of volume data by a DOG wavelet," *1994 Symp. on Volume Visualization*, pp. 35-42.

NEOPHYTOU, N. AND MUELLER, K., "Space-time points: 4D Splatting on effcient grids," *Symp. Vol. Vis. and Graphics 2002*, pp. 97-106.

PRESS, W., FLANNERY, B., TEUKOLSKY, S., VETTERLING, W., *Numerical Recipes in C*, Cambridge University Press, 1988.

REDDY, M., "Perceptionally optimized 3D graphics," *IEEE Computer Graphics & Applications*, vol. 21, no. 5, pp. 68-75, 2001.

RUSINKIEWICZ, S., LEVOY, M., "QSplats: A multiresolution point rendering system for large meshes," *ACM SIGGRAPH 2000*, pp. 343-352, 2000.

THEUßL, T., MÖLLER, T., GRÖLLER, M. E., "Optimal Regular Volume Sampling". *In Proceedings of IEEE Visualization 2001*, pp. 91-98, 2001.

WESTENBERG, M. AND ROERDINK, J., "X-Ray Volume Rendering by Hierarchical Wavelet Splatting," *Proc. 15th Internl' Conference on Pattern Recognition,* pp. 163-166, 2000.

WESTERMANN, R., "A multiresolution framework for volume rendering," *Proc. 1994 Symp. on Volume Visualization*, pp. 51-58, 1994.

WESTOVER, L., "Footprint evaluation for volume rendering", *ACM SIGGRAPH 90*, pp. 367-376, 1990.

WESTOVER, L., "Interactive volume rendering," *Proc. 1989 Wo rkshop on Volume Visualization*, pp. 9-16, Chapel Hill, NC, May 1998.

ZWICKER, M., PFISTER, H., VAN BAAR, J. AND GROSS, M., "Surface Splatting," *ACM SIGGRAPH 2001*, pp. 371-378, 2001a.

ZWICKER, M., PFISTER, H., VAN BAAR, J. AND GROSS, M., "EWA Splatting," *Proc. IEEE Visualization 2001*, pp. 29-36, 2001b.