Eurographics Symposium on Geometry Processing (2003)
L. Kobbelt, P. Schröder, H. Hoppe (Editors)

Statistical Point Geometry

Aravind Kalaiah and Amitabh Varshney

Graphics and Visual Informatics Laboratory
Department of Computer Science and UMIACS
University of Maryland
College Park, MD - 20742, USA
{ark | varshney}@cs.umd.edu

Abstract

We propose a scheme for modeling point sample geometry with statistical analysis. In our scheme we depart from the current schemes that deterministically represent the attributes of each point sample. We show how the statistical analysis of a densely sampled point model can be used to improve the geometry bandwidth bottleneck and to do randomized rendering without sacrificing visual realism. We first carry out a hierarchical principal component analysis (PCA) of the model. This stage partitions the model into compact local geometries by exploiting local coherence. Our scheme handles vertex coordinates, normals, and color. The input model is reconstructed and rendered using a probability distribution derived from the PCA analysis. We demonstrate the benefits of this approach in all stages of the graphics pipeline: (1) orders of magnitude improvement in the storage and transmission complexity of point geometry, (2) direct rendering from compressed data, and (3) view-dependent randomized rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; G.3 [Mathematics of Computing]: Probability and Statistics

1. Introduction

Recent advances in 3D acquisition technologies have resulted in a large and growing body of 3D point datasets. Such datasets are typically dense, unstructured, lack connectivity, and are increasingly becoming very large with upwards of several million to billions of points. Current techniques for handling such large point datasets have evolved from several decades of research in triangle meshes. The triangle meshes with their origins in the high-precision CAD community have strongly favored deterministic representations. This has carried over to representation of point datasets where we encode the attributes of every input point sample explicitly. This is an appropriate representation when the precision needs are high or the geometry is small. However, the rising usage of high detail geometry has gradually increased the cost of a deterministic representation to a level, where we believe it has become attractive to explore other, less deterministic, representations for point datasets.

In this paper we propose a scheme for representing details of a given unstructured point sample geometry using a hierarchical statistical analysis of the original dataset. Hierarchical statistical analysis allows us to trade off accuracy against determinism. Our motivation for this new representation lies in the twin observations: (1) there is a very high coherence in local point neighborhoods, and (2) the accuracy required to generate a visually realistic image from a point cloud model can be achieved using statistical methods on a sparse point representation.

We analyze the input model using Principal Component Analysis (PCA) [10] – a tool that has been used successfully for analysis of empirical data in a variety of fields. Our input consists of a collection of points with attributes such as spatial location, normal, and color. A PCA analysis of this input allows us to compactly represent the model, reconstruct detail where desired, as well as use randomized rendering to efficiently display the data. The principal contributions of our approach are:

Geometry Bandwidth Reduction: A PCA representation

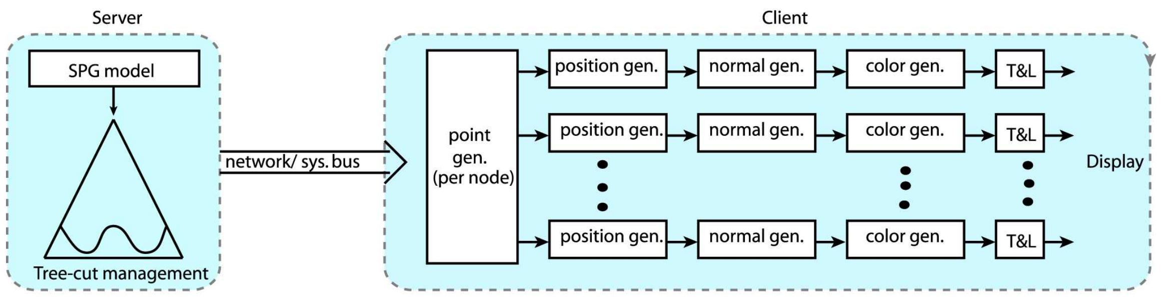


Figure 1: *The rendering pipeline: The server selects the tree cut to be used for rendering in a view dependent manner. The nodes of the cut are then transmitted to the client, which is either the graphics card or a remote machine. The client renders each node by generating points and their attributes from the nodes statistical information. This generation can be speeded by caching and reusing the generated data.*

compactly aggregates the geometric and appearance attributes of a number of points. This reduced representation is more efficient to store and transmit. The graphics processor on receiving the PCA parameters can then use them to regenerate points that share the statistical properties of the input data. A hierarchical PCA representation goes a step further and supports view-dependent transmission and rendering.

Randomized Rendering: Since we are working from a statistical representation of the underlying data, it is highly suitable for stochastic generation and rendering of detail in a tightly-coupled manner. Randomized rendering of point data has the advantage of high-quality rendering that is output-size-sensitive. Our view-dependent randomized rendering generates points based on the number of pixels that are covered by the local region represented by a PCA cluster.

Here is an overview of our approach. The PCA analysis of a group of points results in an estimate of their local orientation frame, the mean, and the variance along the different axes of the coordinate frame. We pre-process the input point dataset using an octree and performing a PCA analysis for each cell of the octree. The PCA parameters (orientation frame, mean, and variances) of the cells tend to be similar for coherent regions. This helps us to classify the node parameters with clustering and quantization. Figure 2 illustrates the entire preprocess pipeline.

At run-time the model is visualized using the Gaussian distribution derived from the PCA analysis. A view-dependent manager determines the cut in the octree and each node of the cut is then visualized with points generated using a trivariate Gaussian random generator. Other attributes of the points such as normal and color are also generated using Gaussian random processes. Figure 1 illustrates the rendering pipeline.

The rest of the paper is organized as follows: we discuss related work in §2 and then discuss how each node is modeled and encoded statistically in §3. The rendering of the encoded input is discussed in §4 and the applications and results are discussed in §5. We conclude the paper with conclusion, discussion, limitations, and future work.

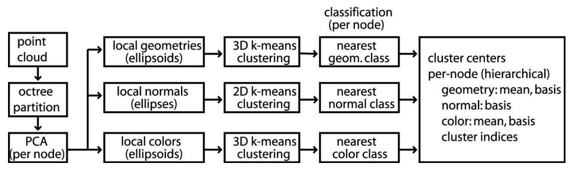


Figure 2: *The preprocess pipeline*

2. Related Work

2.1. Modeling

The traditional approaches to modeling graphics objects include triangle meshes, parametric surfaces, implicit methods, as well as representations that are based on points, images, and volumes. Recent approaches to modeling also include procedural modeling [11]. Here we propose taking the statistical approach to model an object given its point-sampled representation. Our idea is to statistically classify local geometries. The strength of our approach lies in the ability to exploit local coherencies on a global scale. The input to our algorithm could be the points obtained directly from the scanner or after processing for surface reconstruction [3, 4], editing [34], simplification [23] and signal processing [22].

2.2. Representation

A compact representation is essential for achieving bandwidth reduction in the network or bus transmission of a

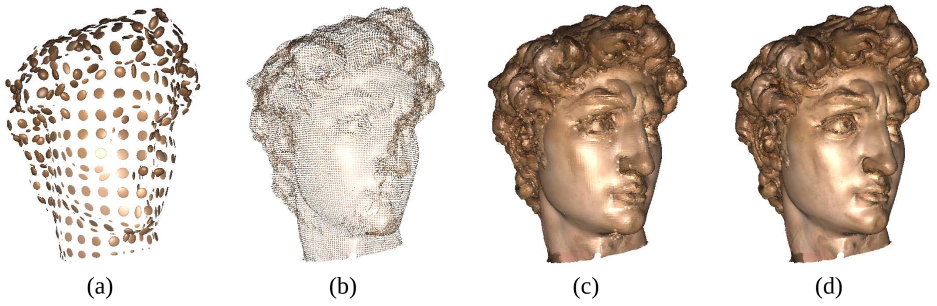


Figure 3: *Figures (a), (b) are two different levels of the PCA Hierarchy shown here by their spatial PCA ellipsoids. The intercepts of these ellipsoids is σ_p. Scaling the intercepts by a factor of γ gives us a hole free representation (figure (c)). The ellipsoids undergo further scaling by a factor of λ to account for the local surface curvature (figure (d)). See figure 12 (see color section) for a picture illustrating the surface curvature.*

given geometry. Traditional methods reconstruct the original input *as is*, with the variable being a possible loss of detail due to quantization [8, 15, 30, 31]. Such methods have been extended for progressive compression and reconstruction [2, 7, 13, 29]. Point-based representations include the bounding balls hierarchy [27], the octree hierarchy [24, 33], and progressive implicit surface representations [12]. Alternatively, higher compression rates can be obtained by using equivalent representations that reconstruct the given input without necessarily trying to reproduce the original samples [18] or with spectral compression [17]. Our statistical approach belongs to this category and achieves better geometric compression since the number of primitives are greatly reduced.

Our representation has several benefits: (1) decoding of each local geometry is done entirely independently, and hence is not memory intensive, is fast, lends itself easily to progressive transmission, and offers, direct real-time rendering from compressed data, (2) it offers a uniform framework for compressing other local attributes of the model such as color, normal, and texture coordinates, and (3) the savings in geometry bandwidth can be used for network transmission of 3D geometric objects as well on a stand-alone computer-system bus connecting CPU, GPU, and memory.

2.3. Rendering

Levoy and Whitted [20] introduced points as geometric rendering primitives. Points have found a variety of applications [14] including efficient rendering of large complex models [26]. Points have evolved from being rendered as a pixel per point to more interesting primitives centered at each sample point. The primitives include sphere [26], tangential disk [24, 35], quadratic surface [16], and higher degree (3 or 4) polynomials [1]. These methods are successful in covering inter-point spaces as long as the local sampling density can provide sufficient detail in the image space. Another useful application of point primitives is the rendering of regions of a triangle mesh with small screen-space projection area [6, 9].

We have developed a statistical approach to model the local geometry in the object space. In our approach we independently render each local geometry by random point generation. Our work has some common elements to procedural rendering [11, 25] and the randomized z-buffer algorithm for triangle meshes [32]. The difference is that our approach uses statistical properties to generate geometry along with other local attributes such as normal and color to achieve a fully randomized rendering. Variance analysis has been widely used for anti-aliasing. Schilling [28] uses it for anti-aliasing normals in bump mapped environment mapping.

3. Statistical Neighborhood Modeling

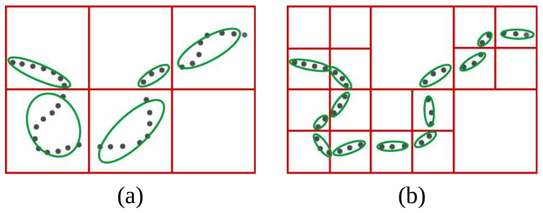


Figure 4: *A PCA hierarchy illustrated in 2D: An octree subdivision of the model is followed up by a PCA analysis of the points in each node. The PCA analysis of the spatial positions of the points give approximating ellipsoids. A separate PCA analysis is done for other local attributes, such as normal and color.*

The PCA analysis of a collection of N points in a d-dimensional space gives us the mean μ, an orthogonal basis f, and the variance σ of the data [10]. The terms μ and σ are d-dimensional vectors and we refer to their i-th scalar value as μ^i and σ^i respectively, with $\sigma^i \geq \sigma^j$ if $i > j$. The basis f consists of (atmost) d vectors with the i-th vector referred to as f^i. Our input is a set of N points with three attributes: spatial position p, normal n, and color c. It should also be possible to handle other local attributes such as texture coordinates using our approach. We do an independent PCA analysis for each of these attributes since they each have special requirements. For instance, normals are unit length and colors have to lie in the range $[0,1]$. We refer to the mean, variance, and the basis of each of these attributes by their subscripts p, n, and c corresponding to the position, normal and color respectively (eg. μ_p, f_n, and σ_c). We determine the values μ_p, f_p, and σ_p from the PCA analysis of the (x,y,z) values of the points. This gives us an ellipsoid centered at μ_p, aligned in the directions f_p^1, f_p^2, and f_p^3, with the intercepts of the ellipsoid being σ_p^1, σ_p^2 and σ_p^3 respectively.

The PCA computation is in general quite robust except for the degeneracies associated with processing data with zero variances. In practice, we set a minimum value for σ_p^i (of the order 10^{-15}) and any variance is set to the maximum of itself and this value. This allows us to deal uniformly with all boundary cases. Thus at render time we only need to consider ellipsoidal (Gaussian) distributions (even if they are vanishingly thin along some dimensions) to generate the points. When there are two zero variances, we retain the principal direction derived from eigen-analysis and modify the other two directions so that the z-direction of the ellipsoid points along the average normal. For three zero-variances, we set the z-axis of the ellipsoid to point along the normal while the other two directions are any two orthogonal vectors in the tangent plane. The values μ_c, f_c, and σ_c are determined from the PCA analysis of the (r,g,b) values of the points. However, we do not find any necessity to set a minimum value for σ_c^i.

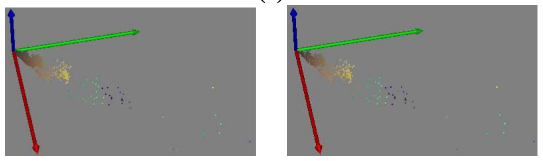
The PCA for normals has to be done on the unit sphere. We first orient a unit sphere such that its z-axis is along the average of the N normals. We then transform all the normals to this basis and determine their respective elevation (θ) (measured from the z-axis) and azimuth (ϕ) angles. The normals are now points in this sphere and they are unwrapped onto a tangent plane at the north pole using the transformation: $(u,v) = (\theta\sin(\phi), \theta\cos(\phi))$. This parameterization preserves the arc-lengths along the longitudes. A PCA in this parametric space gives us an ellipse. The x- and the y-axes of the sphere are then made parallel to the axes of the ellipse. The PCA analysis of the normals thus gives us a 2D variance vector σ_n and a 3D frame f_n (basis of the sphere), which represents both its mean and principal directions.

To represent the data at different levels of detail we spatially partition the input using an octree, and do an independent PCA analysis of the points in each node of the octree. This is illustrated in 2D in figure 4. Figures 3(a) and 3(b) show the PCA nodes at two different resolutions. The octree subdivision is done top-down with the recursive subdivision terminating at nodes which have less than a user-specified number of points. This is done mainly to reduce the number of nodes since there is generally tremendous coherence at the leaf nodes. Each spatial ellipsoid represents a Gaussian distribution with variance σ_p. We limit the extent of the distribution to $\gamma \times \sigma_p$ (γ is generally set to 3.5 since it corresponds to a confidence interval (CI) greater than 99.7%). This gives us a hole-free representation of the surface as shown in figure 3(c). We further factor in the inherent assumption that the points are actually from a surface and note that the value $\frac{\sigma_n^1}{\|\sigma_p\|}$ is a measure of the maximum principal curvature of the local surface area (figure 3(d)). We scale the variance σ_p by a factor $\lambda = c\sqrt{\frac{\sigma_n^1}{\|\sigma_p\|}}$ to get a curvature-sensitive analysis of point locations (figure 3(e)).

3.1. Classification and Quantization

$\mu_p(32)$	$f_p(24)$	$\sigma_p(12)$	$f_n(24)$	$\sigma_n(12)$	$\mu_c(16)$	$f_c(10)$	$\sigma_c(6)$

(a)



(b) (c)

Figure 5: *(a): A node is quantized into 13 bytes for the spatial and normal information. Four extra bytes are used for the optional color information. The breakdown is shown shown in bits. (b): About 600K PCA values of σ_p for the David's Head, and (c): their 512 k-means cluster centers.*

The refinement of the geometry into nodes with independent PCA analysis leads to significant savings (see §5.1). Further, the PCA parameters are highly coherent across nodes. This is reflected in the similarity of the variances. We run a K-means clustering algorithm on the variances (σ_p, σ_n, and σ_c) to derive a small number of representative variances (between 64 to 4K for each model). Figure 5 shows the original variance values and the cluster centers for σ_p. With each node we then store the index of the best matching variance using 12 bits each for σ_p and σ_n and 6 bits for σ_c. We use quantization to further reduce the rest of the storage. The values of μ_p are encoded in 32 bits using a $10-11-11$ quantization depending on the direction of minimum width. The value of μ_c is encoded in 16 bits using a $5-6-5$ quantization of its red, green, and blue values. To encode frame f_p, we observe that it is a rotation of the unit basis. This corresponds to first rotating the basis in the plane of its z-axis and f_p^3, so that the z-axis is now f_p^3. It is then rotated again so that its x- and y-axes are equal to f_p^1 and f_p^2. The encoding of f_p is done by using 8 bits for the the second rotation and 16 bits for the f_p^3 axis. The encoding of f_n is done similarly with 24 bits, while f_c is encoded in 10 bits with three bits for the angle and seven bits for f_c^3. This lets us compress the PCA representation for one cluster to 13 bytes without color and 17 bytes with color.

4. Randomized Rendering

The PCA subdivision of the geometry gives us local geometries compactly approximated by ellipsoids. We render these geometries by sampling them for points according to their PCA distribution parameters.

The PCA analysis gives us a Gaussian distribution for each of its attributes (geometry, normal, color, etc). For an ellipsoid centered at the origin with its frame being the coordinate axes, the spatial attribute of points is generated using the following trivariate Gaussian random number generator (a 3D version of the Box-Muller transform [5]):

$$\tau_p = \sqrt{-2\ln(r_{p0})}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \sigma_p^1 & 0 & 0 \\ 0 & \sigma_p^2 & 0 \\ 0 & 0 & \sigma_p^3 \end{bmatrix} \begin{bmatrix} \tau\sqrt{1-r_{p2}^2}\,\cos(2\pi r_{p1}) \\ \tau\sqrt{1-r_{p2}^2}\,\sin(2\pi r_{p1}) \\ \tau\, r_{p2} \end{bmatrix}$$

where r_{p0}, r_{p1}, and r_{p2} are uniformly distributed random numbers in $(0,1]$, $[0,1]$, and $[-1,1]$ respectively. Here τ_p ensures that the distances of the points from the mean are spread in a Gaussian manner, while r_{p1} and r_{p2} approximate a uniform parameterization of the sphere with a $(\cos(\theta),\phi)$ parameterization. The generation of the point colors is done similarly. The (θ,ϕ) values of the normals are generated using a Gaussian sampling of its ellipse using the Box-Muller transform as follows:

$$\begin{aligned} \tau_n &= \sqrt{-2\ln(r_{n0})} \\ \alpha &= \sigma_n^1\tau_n\cos(2\pi r_{n1}) \\ \beta &= \sigma_n^2\tau_n\sin(2\pi r_{n1}) \\ \theta &= \sqrt{\alpha^2+\beta^2} \\ \phi &= \tan^{-1}\left(\frac{\beta}{\alpha}\right) \end{aligned}$$

where r_{n0} and r_{n1} are uniform random numbers in $(0,1]$ and $[0,1]$ respectively. Here τ_n ensures a Gaussian distribution of the samples in the tangent plane to the sphere at (0,0,1), with the α and β being intermediate terms in the inverse projection to the (θ,ϕ) space.

An important constraint on the sampling process is that there should be sufficient number of them to fully cover the screen space area (a_s) of its ellipsoid. The projection of an ellipsoid to a plane is an ellipse which can be determined with eigen analysis. When the ellipsoid is far away then this can be approximated by computing the screen-space projection area of its best-enclosing sphere. If the distribution of points within the ellipsoid is uniform then the required number of points is given by the term $a_s\ln(a_s)+ca_s$, where the value c decreases dramatically with the value of a_s [32]. Generating this number of points using a Gaussian distribution does not necessarily cover all the pixels in a_s. However, since the Gaussian distribution of neighboring nodes overlap, we have that all screen pixels are covered. A value of $c = 0.2$ was sufficient for all our test cases.

```
Render( )
1.  For every node, n, in the octree cut
2.      Decode( n )
3.      If ( cull( n ) )
4.          merge( n )
5.          continue
6.      $p \leftarrow$ RequiredNumberOfPoints( n )
7.      If ( DoSplit( n, $p$ ) )
8.          split( n )
9.          continue
10.     RenderPoints( n, $p$ )
11.     If ( DoMerge( n, $p$ ) )
12.         merge( n )
```

Figure 6: *The view dependent rendering algorithm.*

Our view-dependent rendering algorithm is similar to the ones generally used for triangle meshes [21]. We maintain an active list of nodes representing the octree cut using a doubly linked list. The overall rendering algorithm is described in figure 6. We do the rendering by iterating over the nodes of the octree cut. A node is rendered only if its normal-cone based culling test is negative [19]. To be on the conservative side, the width of the cone is chosen to be the maximum of the normal ellipse intercepts. If the node could be culled then it can be merged to its parent only if all its siblings are willing to be merged as well. To do this efficiently, at each node, we keep a count of how many children wished to be merged at that rendering cycle. We also maintain a time stamp at each node identifying the last time this count was updated. So, when a parent node gets a merge request from a child, the count is reset to one if its time stamp is outdated, else it is incremented. The actual merge happens when this count is equal to its number of children. The children are then replaced by the parent in the octree cut. This is done by the *merge*( ) function of the algorithm. If the node is not culled then we compute the number of points to render. If this number is above a maximum threshold (set to 40 in all our tests) then the node is split. The children of the split node are inserted into the octree cut in the position of the parent (done by the function, *split*( ), in the algorithm) and the iteration continues with the first of the inserted children. The node is rendered by generating the points with their attributes. The points could be generated on the fly or they could be selected from a cache. In a network environment, the server can send the client just the encoded information of the node and the number of points to render. After rendering, the node puts in a merge request with its parent if its screen-space size is less than a threshold (8 in all our test cases).

While an on-the-fly point generation is suitable for hardware implementation and saves memory space, caching is

Octree Level	Compression (in %)	Max. error (in %)	Avg. error (in %)
Level 9	213.79	0.4161	0.0751
Level 8	829.23	0.546	0.1466
Level 7	3392.11	1.186	0.280
Level 6	13316.08	1.952	0.641
Level 5	45433.10	3.768	1.504
Level 4	104477.13	8.086	3.458

Table 1: *The relationship between the compression and the degree of approximation for the David's Head model. The first column represents the hierarchy level of the octree cut. The error columns measure the Hausdorff distance between the generated points and the original point samples.*

generally faster on current systems. For caching, we maintain a global bank of blocks of points of constant size (16 in all our tests). A node in the octree cut can request these blocks as necessary and will fill it up with generated points right after assignment. When a node is split, its parents cache is freed, while the act of merging a node with its parent frees up its children's cache. The overall size of the global bank is directly related to the window resolution and a one time allocation of 40 million points sufficed for all our tests on a 1024×1024 window.

5. Applications and Results

We did all our tests on a Pentium4 based PC with 2GB RAM and a NVIDIA Quadro4 graphics card. We tested our work on the Stanford's David's Head model, David (full) model, Lucy model, and the St. Matthews face model. We added colors to these models by solid texturing. These models took no more than two hours of preprocessing with the classification and quantization phase taking up most of the time. We used a naive partition-based clustering scheme. More advanced clustering schemes should improve this number dramatically [10].

5.1. Storage and Network Transmission

The hierarchical PCA subdivision of the geometry is very effective at compactly representing the 3D point-based models. Given the original collection of points, a typical uncompressed representation would require 8 bytes for each point (two bytes for each of the *x*, *y*, and *z* components and two bytes for the normal). On the other hand, our PCA representation can encode a collection of points with just 12 bytes, which means that one starts saving with a PCA representation as soon as the number of points in a cluster exceeds two. The processing of the David's Head data yielded us a tree with about 1.34 million nodes of which about one million nodes are at the leaf level. We used 1500 spatial ellipsoids, 800 normal ellipses, and 128 color ellipsoids to classify the respective intercepts of the PCA nodes. The original 2 million points of the David's Head model requires about 46MB of data while our total representation with the hierarchy and the classification requires about 21MB. Figures 9(a), 9(b), and 9(c) (see color section) show the PCA hierarchy for the Lucy model. Figure 10 (see color section) shows that the approximation with classification and quantization does not add any noticeable distortion to the model.

The compression however comes at the cost of an approximation error. Table 1 summarizes this relationship for the David's head model. The approximation error is measured using the Hausdorff error metric. We first compute the Hausdorff distance between the points generated at a given node and the original points present at the node (the number of generated points is equal to the number of original points at that node). We then convert this number to percentage by comparing it with the bounds of the best fitting axis-aligned cube. The third column in table 1 represents the maximum of all such (per-node) distances at any given level. For example, the maximum distance between any generated point at level 9 and its nearest point on the surface is about 0.4%. We average the Hausdorff distances over all the nodes at a level and these values are shown in column four of table 1. These results show that the error can be very small at higher resolutions while still giving significant compression. Pauly et. al. discuss a more sophisticated measure of geometry error [23].

The nodes of the PCA hierarchy represent their local region of interest independent of each other (no connectivity). Hence, they are suitable for order-independent network rendering in a client-server model, and also potentially in a single system rendering with GPU-level support for vertex generation and random number computation. The key issue in such an environment is network/bus bandwidth reduction. To illustrate the reduction in the network bandwidth we setup an experiment where the camera eye is placed at various distances relative to the object center. The object is then rotated

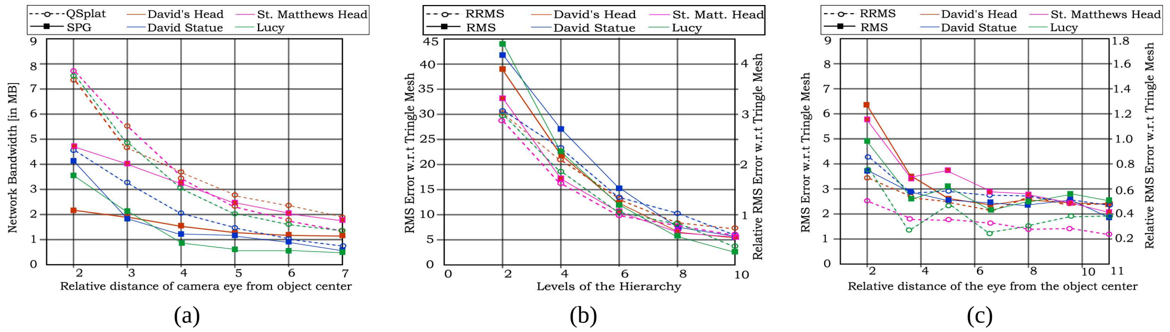


Figure 7: *(a): Comparison of the network bandwidth for view dependent transmission. The x-axis measures the distance of the eye w.r.t. the center of the (normalized) bounding box of the model. (b): Screen space error (measured w.r.t. a plain triangle mesh based rendering) is inversely proportional to the level of the hierarchy cut used for rendering. (c): Chart illustrating the (screen space) error during view dependent rendering. The error is measured w.r.t. a plain triangle mesh based rendering. The x-axis refers to the distance of the eye from the center on the object (same distance as in case (a)).*

around an axis aligned with the y-axis of the camera and the network bandwidth required to transmit the PCA information in the octree cut is averaged over a 360 degree rotation. The results are shown in figure 7(a). We compare the results of our approach of Statistical Point Geometry (SPG) with QSplat. QSplat is actually designed for network streaming. However, by the strength of its broad approach, it doubles up as the state-of-the-art in point-based network graphics. We did all our tests on a 1024×1024 test window. The results show that we obtained an orders-of-magnitude reduction in network bandwidth. We note that this improvement is at the cost of approximately regenerating the original data (not rendering the actual input points like QSplat does).

5.2. Randomized Rendering

The hierarchical nature of the PCA analysis means that the error of approximation decreases dramatically with the level of subdivision. Apart from the object space error analysis shown in table 1, we also compare the rendering quality with a plain triangle mesh rendering (figure 7(b)). Results show that the approximation can get very close to a triangle mesh representation when higher levels of the hierarchy are rendered. Figure 7(c) compares the screen space image quality of our view-dependent rendering, with a plain triangle-mesh-based rendering. Figure 11 (see color section) illustrates view dependent rendering of the David's Head model. The rendering speed was roughly half the speed of QSplat, with about 3-4 frames per second for a 2× relative distance from the camera eye. At resolutions where the number of points sent over the bus are similar, the rendering speeds are similar as well. However, at closer views, since the size of our points is always a single pixel, we render more points leading to a slower rendering speed.

6. Discussion and Conclusions

We have presented a novel way to represent geometry using statistical analysis. The method works by exploiting local coherence with a PCA analysis and is fairly general in that it can handle other local attributes such as normal and color within this framework. The representation is quite compact and efficiently approximates the original geometry with hierarchy. The application of this work can be realized in reducing network bandwidth and in high-quality interactive rendering. This representation also holds promise for a reducing the bus bandwidth in future graphics card architectures.

There is, however, scope for much improvement. The rendering of the PCA nodes with randomly sampled points fits well with the current generation of graphics hardware. We believe that an EWA-style [35] anti-aliased rendering of the PCA nodes would give us high quality rendering and also get rid of the temporal aliasing. The treatment of the PCA subdivision is done with no connectivity information and this can sometimes cause two non-adjacent surface areas to be merged into one PCA node. Assuming a "good" sampling rate, this can be solved with a local clustering analysis which separates the data into spatial- and normal-coherent parts. We have not investigated the relationship between the point size, rendering quality and number of points required for rendering a PCA node. A bigger point size reduces the bandwidth but can cause significant aliasing artifacts. We leave this for future work.

Acknowledgements

We thank the Stanford Graphics Laboratory for providing us with the David's Head model, David (full) model, Lucy model, and the St. Matthews face model. We would also like to thank Thomas Baby for his help with coding. This work was supported by the NSF funding grants IIS00-81847 and ACR-98-12572.

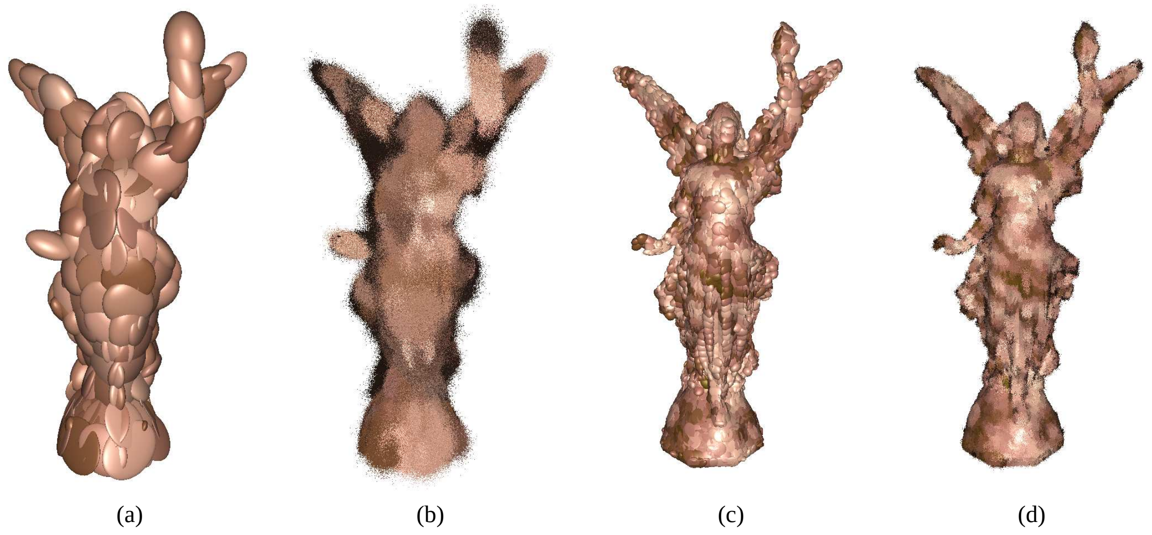


Figure 8: *(a), (c): Lower resolutions of the Lucy model. (b), (d): (Zoomed-in) Randomized renderings at these resolutions. In practise, we adjust the resolution of rendering in a view dependent fashion. See figure 13 (see color section) for an example of view dependent rendering.*

References

1. M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, C. Silva, and D. Levin. Point set surfaces. In *IEEE Visualization 2001*, pages 21–28, October 2001.
2. P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of SIGGRAPH 2001*, pages 195–202, August 2001.
3. N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of SIGGRAPH 98*, pages 415–422, 1998.
4. F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, October 1999.
5. G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *Ann. Math. Stat.*, 28:610–611, 1958.
6. B. Chen and M. X. Nguyen. POP: A polygon hybrid point and polygon rendering system for large data. In *IEEE Visualization'01*, pages 45–52, October 2001.
7. D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *IEEE Visualization '99*, pages 67–72, 1999.
8. M. F. Deering. Geometry compression. In *Proceedings of SIGGRAPH'95*, pages 13–20, August 1995.
9. T. K. Dey and J. Hudson. Pmr: Point to mesh rendering, a feature-based approach. In *IEEE Visualization'02*, pages 155–162, October 2002.
10. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2 edition, 2001.
11. D. Ebert, F. Musgrave, P. Peachey, K. Perlin, and S. Worley. *Texturing & Modeling: A Procedural Approach*. AP Professional, San Diego, 3rd edition, 2002.
12. S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva. Progressive point set surfaces. *ACM Transactions on Graphics*, (to appear) 2003.
13. P.-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics*, 21:372–379, 2002. Proceedings of SIGGRAPH'02.
14. J. P. Grossman and William J. Dally. Point sample rendering. In *Rendering Techniques '98*, Eurographics, pages 181–192. Springer-Verlag Wien New York, 1998.
15. M. Isenburg and J. Snoeylink. Face fixer: Compressing polygon meshes with properties. In *Proceedings Siggraph'00*, pages 263–270, 2000.
16. A. Kalaiah and A. Varshney. Modeling and rendering points with local geometry. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):30–42, January 2003.
17. Z. Karni and C. Gotsman. Spectral compression of

mesh geometry. In *Proceedings of Siggraph'00*, pages 279–286, 2000.

18. A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proceedings of Siggraph'00*, pages 271–278, 2000.

19. S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical back-face computation. In *Rendering Techniques '96*, Eurographics, pages 231–240. Springer-Verlag Wien New York, 1996.

20. M. Levoy and T. Whitted. The use of points as a display primitive. In *Technical Report 85-022, Computer Science Department, UNC, Chapel Hill*, January 1985.

21. D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufman, 2002.

22. M. Pauly and M. Gross. Spectral processing of point-sampled geometry. In *Proceedings of SIGGRAPH'01*, pages 379–386, August 2001.

23. M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization 2002*, pages 163–170, October 2002.

24. H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of SIGGRAPH 2000*, pages 335–342, July 2000.

25. W. T. Reeves. Particle systems — A technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, July 1983.

26. S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH 2000*, pages 343–352, July 2000.

27. S. Rusinkiewicz and M. Levoy. Streaming QSplat: A viewer for networked visualization of large, dense models. In *ACM Symposium on Interactive 3D Graphics*, pages 63–68, March 2001.

28. A. Schilling. Antialiasing of environment maps. *Computer Graphics Forum*, 20(1):5–11, 2001.

29. G. Taubin, A. Gueziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *Proceedings of SIGGRAPH 98*, pages 123–132, July 1998.

30. G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

31. C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface*, pages 26–34, June 1998.

32. M. Wand, M. Fischer, I. Peter, F. M. Heide, and W. Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proceedings of SIGGRAPH'01*, pages 361–370, August 2001.

33. J. C. Woolley, D. Luebke, B. Watson, and A. Dayal. Interruptible rendering. In *ACM Symposium on Interactive 3D Graphics*, pages 143–152, 2003.

34. M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3d: An interactive system for point-based surface editing. In *Proceedings of SIGGRAPH'02*, pages 322–329, July 2002.

35. M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proceedings of SIGGRAPH 2001*, pages 371–378, August 2001.

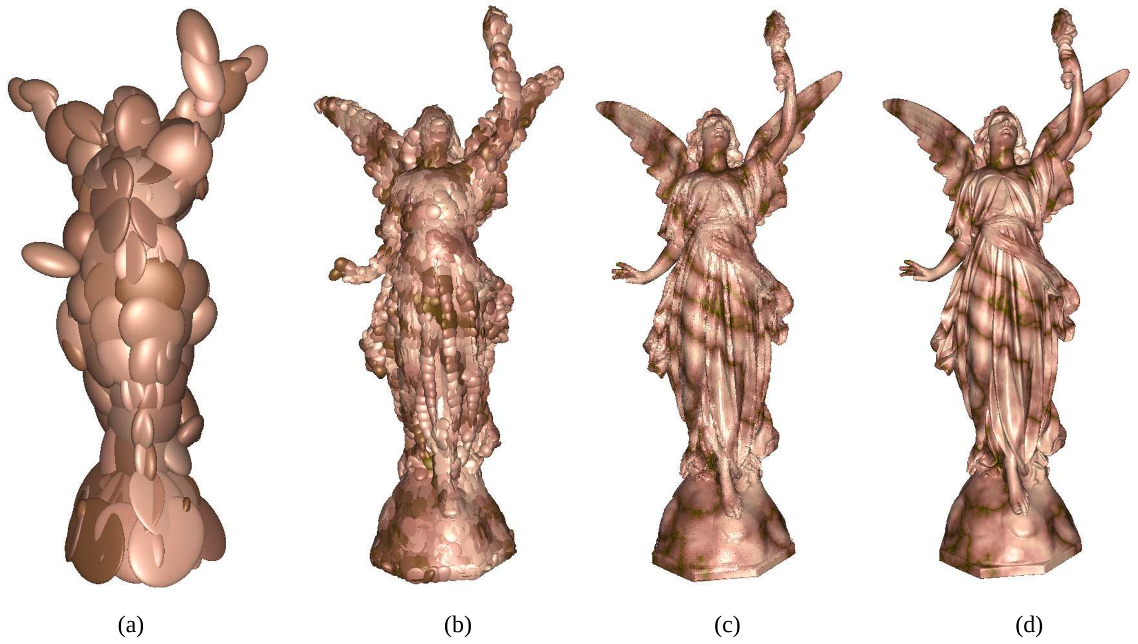


Figure 9: *(a), (b), (c): The PCA subdivision of the Lucy model. The ellipsoids correspond to the spatial subdivision. They are colored by the mean color of the points belonging to that node. (d): Randomized rendering of the lucy model.*

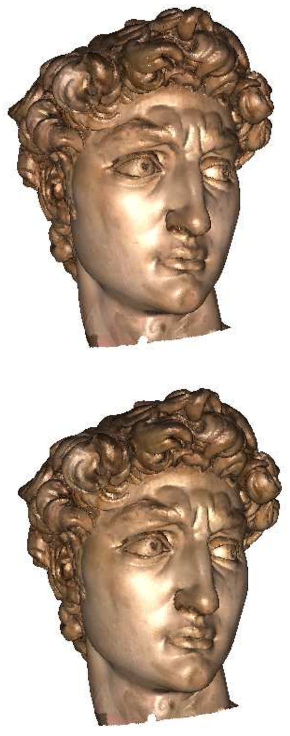


Figure 10: *After quantization and classification of the original input (top) there is no significant change.*

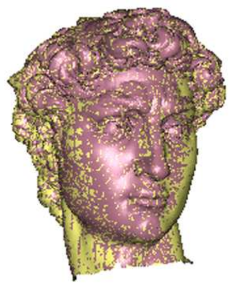


Figure 11: *During view dependent rendering, the generated points are colored according to the level of their node.*

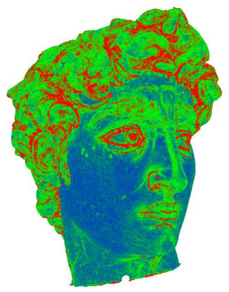


Figure 12: *Estimates of the surface curvature varying from high (red) to medium (green) to low (blue).*

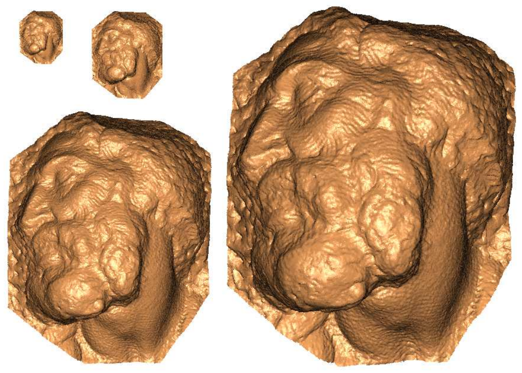


Figure 13: *The view dependent renderings of the St. Matthews Head*