ABSTRACT

Title of dissertation: VISUAL DATA REPRESENTATION USING CONTEXT-AWARE SAMPLES

Aravind Kalaiah, Doctor of Philosophy, 2005

Dissertation directed by: Associate Professor Amitabh Varshney
Department of Computer Science

The rapid growth in the complexity of geometry models has necessisated revision of several conventional techniques in computer graphics. At the heart of this trend is the representation of geometry with locally constant approximations using independent sample primitives. This generally leads to a higher sampling rate and thus a high cost of representation, transmission, and rendering. We advocate an alternate approach involving context-aware samples that capture the local variation of the geometry. We detail two approaches; one, based on differential geometry and the other based on statistics. Our differential-geometry-based approach captures the context of the local geometry using an estimation of the local Taylor's series expansion. We render such samples using programmable Graphics Processing Unit (GPU) by fast approximation of the geometry in the screen space. The benefits of this representation can also be seen in other applications such as simulation of light transport. In our statistics-based approach we capture the context of the local geometry using Principal Component Analysis (PCA). This allows us to achieve hierarchical detail by modeling the geometry in a non-deterministic fashion as a hierarchical probability distribution. We approximate the geometry and its attributes using quasi-random sampling. Our results show a significant rendering speedup and savings in the geometric bandwidth when compared to current approaches.

VISUAL DATA REPRESENTATION USING CONTEXT-AWARE SAMPLES

by

Aravind Kalaiah

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

Advisory Commmittee:

Associate Professor Amitabh Varshney, Chair/Advisor
Professor Larry Davis
Professor Bill Goldman
Assistant Professor David Luebke
Professor David Mount

Dedicate to my parents,
Rathna Kalaiah and Dr. Puttiah Kalaiah

ACKNOWLEDGMENTS

I owe my gratitude to several people who have supported me and guided me through various stages of my education. Above all, I would like to thank my advisor, Amitabh Varshney, for his unmatched advice, support, and encouragement. I have sought his advice on all kinds of issues, both technical and non-technical, and he has been very patient with me and has been a big source of strength for me throughout my graduate studies. I have been fortunate to have several discussions with the committee members. I thank them for their advice.

Many thanks to my high school teachers who got me hooked to mathematics and science. In particular I would like to thank Ms. Prema, Mr. Prabhakar, and Ms. Gayathri for inspiring me to be self-motivated. I would like to thank Ms. Menon for encouraging me to pursue higher studies. Special thanks to my school buddies Ajay, Arvind, Manamohan, and Swaroop for the fond memories.

I am grateful to the faculty of IIT, Bombay for the education I received in basic computer science and mathematics. In particular, I would like to thank my undergraduate advisors Vikram Gadre, Sharat Chandran, and S. Biswas for getting me interested in computer graphics and in graduate studies. I also thank Neelima Talwar for helping me to develop an appreciation for humanities. I would like to thank all my hostel mates at Hostel 3 for the good times at IIT.

I am grateful to Holly Rushmeier and Fausto Bernardini for introducing to some very special topics in graphics and for encouraging me in my research during my intern-

ship at IBM Research.

I am grateful to the faculty and staff at Maryland and Stony Brook for their fruitful discussions and encouragement related to my research. In particular I would like to thank Yiannis Aloimonos, Rama Chellappa, Larry Davis, Ramani Duraiswami, Leila De Floriani, Arie Kaufman, David Mount, and Hanan Samet for inspiring me in ways they are probably not even aware of. Thanks to Bill Goldman for teaching me the basics of Differential Geometry and to David Luebke for the many discussions and advice on point-based rendering. I am grateful to my labmates Xuejun Hao, Thomas Baby, Chang-Ha Lee, and Youngmin Kim for the animated discussions on problems in computer graphics. Many thanks to Pankaj, Kanta, Sulabh, Vinod, Jackie, Harish, Ramani, Karthik, Dami, Merrick, and Nilani for making my stay Maryland a memorable one.

Finally, I would like to thank my extended family and friends for their invaluable support. I owe my biggest gratitude to my parents, to my brother Avinash, and to Anke, for keeping me smiling through the most stressful periods of my work.

Table of Contents

List of Tables

List of Figures

Chapter 1

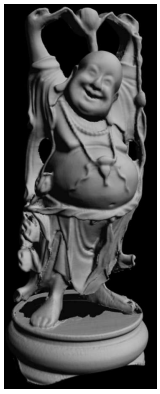
Introduction

In our everyday task of understanding the world around us, we rely significantly on the input gathered by our visual system. Computer graphics primarily deals with methods to augment this system through the creation, capture, processing, and presentation of three-dimensional structures. Over the years, computer graphics has gained tremendous importance in several application domains. For example, it is used in the manufacturing industry for designing mechanical parts and to understand the flow of fluids in a manufacturing process. It is employed in medicine for visualizing internal organ malfunctions and for visualizing molecular docking in rational drug design. It has also become a mainstay in the entertainment industry for cinema and digital games.

Computer graphics deals with several kinds of data including surface geometry, volumetric data, lighting data, and time-varying versions of such data. A large proportion of these datasets are modeled by artists using commercial software such as Maya and 3D Studio Max. Another good fraction of these datasets are generated using simulations such as fluid simulations and global lighting computations. Recently, large datasets are also being captured from real environments using laser range scanners. We categorize all these data under one term as *visual data*.

Visual datasets vary widely in their nature and complexity. Surfaces can be categorized as manifold or non-manifold and can have significant complexity in their genus and curvature. Similarly, the volumetric datasets have 3D complexity. Time-varying data add the dimension of time. Other attributes such as surface reflectance, lighting, and light fields can add further dimensions and complexity to visual data. Working with datasets of such complexity requires high computation, bandwidth, and storage resources. Consequently, the issue of representation of visual datasets is amongst the core challenges in modern computer graphics.

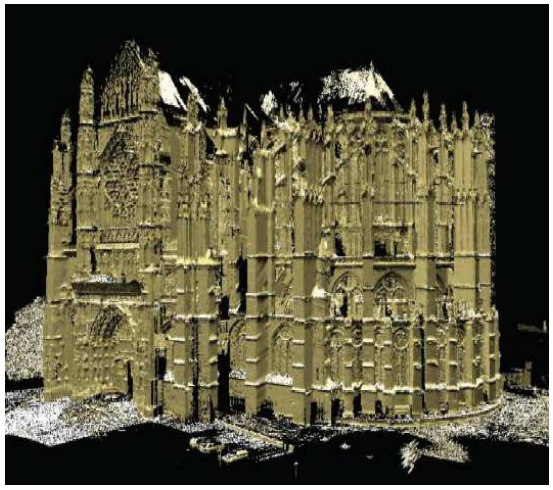
(a) Bunny



(b) Happy Buddha

(c) Pieta

(d) St. Matthew



(e) Ste. Pierre Cathedral

Model	Bunny	Happy Buddha	Pieta	St. Matthew	Ste. Pierre Cathedral
Year	1994	1996	1998	2000	2002
# Points	34,947	543,642	7.2M	127M	220M
Size	200 KB	3 MB	43 MB	762 MB	1.9GB

Table 1.1: *The size of scanned geometry has witnessed a dramatic growth.*

1.1 Recent Trends in Visual Data

The last three decades of research have seen several methods to represent the geometry. These include representations such as triangle meshes, parametric surfaces, implicit surfaces, constructive solid geometry, and procedural surfaces. These diverse representations have coexisted because each representation offers an unique advantage such as speed of rendering, ease of modeling, flexibility of editing, and brevity of representation. However, the last few years have seen another sustained trend that is beginning to take

critical shape: the exploding complexity of visual datasets. This is illustrated in Table 1.1 for the case of scanned data. Several factors have contributed to the increasing complexity of visual datasets. These include developments in a wide variety of fields such as 3D acquisition, 3D modeling, large-scale simulation of physics, and computational biology, which have begun to produce massive datasets for visualization. The large-scale commercial interest in such datasets is further expected to expedite this trend. This has inspired us to seek a revision of the traditional data representational schemes to incorporate the new challenges posed by such large datasets.

1.2 Challenges of Large Visual Data

In order to understand the challenges posed by current datasets we have to keep historical trends in perspective. Historically, computer graphics has approached visual data representation from two different perspectives: (1) accuracy of representation, and (2) efficiency of rendering. The former approach has origins in the editing and high precision requirements of the CAD/CAM industry and includes representations such as parametric surfaces, solid modeling, and implicit surfaces. The latter approach is motivated by interactive visualization and includes representations such as triangle meshes. However, the rapid increase in the size and the complexity of visual datasets has imposed an enormous load on both the representational and computational aspects. This has motivated us to devise a compact representational framework that also supports efficient rendering. Addressing this issue is the primary focus of this dissertation. Our approach involves active consideration of the following orthogonal components:

- **Exploiting Coherence:** Visual datasets tend to have high local coherence. Local coherence can vary in shape, scale, and in its distribution in the object space. Data representation needs to be simple, robust, and adaptive for capturing such local coherencies.

- **The Memory Wall Challenge:** Recent trends in computer architecture suggest that the speed of computation is far outstripping the speed of memory access [127]. This can be a bottleneck for traditional approaches that rely on a memory-intensive

representation. We believe that addressing the memory wall issue would require a computation-based representation that shifts a sizeable load of representation from memory to computation.

- **Geometry Bandwidth:** Transmission can be a bottleneck during interactive visualization since the network and the system-bus are generally not fast enough to keep pace with the Graphics Processing Unit (GPU) [53]. A novel representational framework which is compact both on disk and in memory and supports direct rendering from a compressed format would address this challenge.

1.3 Dissertation Hypothesis

The hypothesis of this dissertation is as follows:

> *Visual data can be effectively represented using sample primitives with embedded contextual information. Such a representation is more efficient for the storage, transmission, and rendering of large visual data when compared to sampled representations with little or no embedded information per sample.*

The motivation for our work comes from the observation that visual data exhibits tremendous amount of local coherence since they are sampled nearly uniformly. In this dissertation we propose two approaches that exploit this coherence by modeling the local context of the samples. In particular, we propose the representation of visual data using samples which have embedded local context information. We call such samples as *Context-Aware Samples* (CAS). This approach allows us to substantially reduce the number of samples and helps us address the issues noted in §1.2. In this dissertation we show how our approach leads to efficient storage, transmission, and rendering of large visual data.

We distinguish between two kinds of local context information that are embedded in the CAS primitives: surface-based and space-based. Our surface-based CAS, *Differential Points* [58, 59], uses the differential geometric attributes of a surface to encode the local context, while our space-based CAS, *Statistical Points* [60, 61], use the statistical distribution of object samples in the local vicinity to represent the context.

1.4 Overview

In the remainder of this dissertation we discuss the details of representing visual data using context-aware samples. In Chapter 2, we detail two kinds of context-aware samples and compare their relative merits. We also compare our approach to previous work in this chapter. In Chapter 3, we discuss how large visual data can be modeled using CAS as basic building blocks. We discuss compact encoding of data modeled using CAS in Chapter 4. In Chapter 5, we discuss efficient transmission and rendering of CAS using modern Graphics Processing Units (GPU). We discuss the results and applications of our work in Chapter 6. In Chapter 7 we show how statistical points can be used for shadow computation. We conclude this dissertation in Chapter 8.

1.5 Contributions

Previous work on representing visual data has focused on fitting implicit or parametric functions to the samples or linearly interpolating between the samples (eg. triangle meshes). The main contribution of our work is that we propose a new approach of embedding local contextual intelligence into individual samples and using them as the building blocks to represent the overall data. We believe that this new approach is likely to offer several benefits in computer graphics. Some applications of our approach are already being developed by others [117, 118] and we expect our approach to become even more popular in the future.

Chapter 2

Context-Aware Samples

Computer graphics has a rich collection of geometric modeling approaches such as procedural, parametric, implicit, and sampled representations. The procedural and implicit representations enjoy unique advantages in compression, modeling natural phenomenon, and editing. However, by far the most widely used geometric representations are the sampled representations. They are easy to acquire, very flexible, and can represent arbitrarily complex data.

Sampled Representation

The underlying principle of sampled representations is to sample the original data and then approximate or reconstruct it by interpolating between these samples. Popular sampled representations include triangle meshes, parametric surfaces, textures, and 3D volumetric datasets. Triangle meshes and tetrahedral meshes are sampled representations in which the samples are connected by edges and the visual data is approximated by linearly interpolating between the samples (the edges are used to identify the neighbors). NURBS are a higher-order sampled representation which offers non-linear interpolation between the samples (control points). However, in this case, the samples need not necessarily be on the original surface geometry. Textures and 3D volumes are regularly sampled representations where the interpolation is done implicitly using linear or non-linear techniques. Sampled representations have been used extensively for representing geometry as well as other kinds of data such as surface reflectance properties [120], bi-directional texture fields [25], illumination [55], and light fields [71].

Context-Aware Samples

A sampled representation that is growing in popularity is point clouds [73]. Although such a representation cannot give a continuous representation of the data it has been found to be useful for rendering purposes. Since this representation just has points with-

out any per-point context information, we call such samples as *Context-Blind Samples*. Some recent work in this area assign a spherical volume to each sample and renders such points as squares, circles, or ellipses [99]. Alternately, a tangent disk can be associated with each point and the points can be splatted or blended on the screen for a high-quality rendering [129]. We categorize such primitives under *Context-Aware Samples* since each sample has some information describing the local vicinity. In this dissertation we formalize the notion of context-aware samples and introduce two classes of context-aware samples: one based on the Taylor's series expansion (Differential Points) and the other based on Statistical Analysis (Statistical Points). Before we present the details of our context-aware samples, we briefly discuss prior art in the next section.

2.1 Previous Work

Our work on context-aware samples is motivated by historical trends in modeling, representation, and rendering of the geometry. In this section we briefly summarize some of the previous works that have influenced our work. In this dissertation we use the terms 'samples' and 'points' interchangeably.

Artists use several modeling approaches for 3D content creation. These include parametric, implicit, and procedural [35] methods for modeling the geometry. Recent advances in 3D acquisition have made it possible to digitize real-world geometries [72]. This has lead to a dramatic increase in the size of the geometry modeled with intricate details. We address this issue by embedding contextual information in the samples which reduces the cardinality and the overall size of the geometry. Our approach can handle any sampled visual data whether created by artists, generated by simulations, or acquired from the real world.

2.1.1 Geometry Acquisition and Processing

Point samples of real-world environments are acquired using several acquisition techniques [7, 37, 72, 93, 98] with the choice depending on the environment being sampled. This information is processed by surface reconstruction algorithms [6, 9] and subse-

quently denoised [49]. The sampled points can also be processed directly using spectral processing techniques [86]. Alternately, the coarse triangle mesh can be fitted with parametric surfaces [36, 69] for denoising and to aid other higher-level applications.

Our work on differential points uses results from classical differential geometry which gives us a mathematical model for understanding the surface variation at a point. There is a collection of excellent literature on this subject and in this dissertation we follow the terminology of do Carmo [29]. Curvature computation on parametric surfaces has a robust mathematical model. Various techniques have been designed to estimate curvature from discrete sampled representations [52, 108]. Taubin [109] estimates curvature at a mesh vertex by using the eigenvalues of an approximation matrix constructed using the incident edges. Desbrun *et al.* [79] define discrete operators (normal, curvature, etc.) of differential geometry using Voronoi cells and finite-element or finite-volume methods. Their discrete operators respect the intrinsic properties of the continuous versions and can be used at the vertices of a triangle mesh.

We use a simplification process to prune an initial set of differential points to obtain a sparse point representation. Turk [114] uses a point placement approach where the point density is controlled by the local curvature properties of the surface. Witkin and Heckbert [122] use physical properties of a particle system to place points on an implicit surface. Simplification methods have been studied extensively for triangle meshes. They can be broadly classified into local and global approaches. Local approaches work by pruning vertices, edges, or triangles using various metrics. Global approaches work by replacing subsets of the mesh with simplified versions or by using morphological operators of erosion and dilation. Cignoni *et al.* [21] and Cohen *et al.* [22] document various simplification techniques. More recently, Lindstrom [74] uses error quadrics in a vertex clustering scheme to simplify complex datasets that are too large to fit into main memory. We refer the readers to the book by Luebke *et al.* [76] for a thorough treatment of simplification and level-of-detail techniques.

Image-assisted organization of points [45, 75, 103] are efficient at three-dimensional transformations as they use the implicit relationship among pixels to achieve fast incremental computations. They are also attractive because of their efficiency at represent-

ing complex real-world environments. The multiresolution organizations [17, 89, 99] are designed with the rendering efficiency in mind. They use the hierarchical structure to achieve block culling, to control depth traversals to meet the image-quality or frame-rate constraints, and for efficient streaming of large datasets across the network [100]. Recent advances on surface parameterization have allowed the representation of the entire surface in the parametric space as a set of images [46, 90]. We refer the readers to the book by Samet [101] for a thorough treatment of the advantages of spatial data structures such as images, quadtrees, and octrees.

The input to our algorithm could be the points obtained directly from the scanner or after processing for surface reconstruction [4], editing [88], simplification [87], and signal processing [86].

2.1.2 Geometry Representation

Historic developments in data modeling and hardware rasterization have led to the adoption of the triangle mesh as the preferred representation of the geometry. Various methods for lossy and lossless compression of the triangle mesh have been proposed [27, 54, 111, 113]. Such methods have been extended for progressive compression and reconstruction [3, 23, 42, 110]. Alternatively, higher compression rates can be obtained by using representations that approximate the given input without necessarily trying to reproduce the original samples [66], by using spectral compression [63], or by mapping the geometry to images [90]. Our approach belongs in this category and achieves better geometric compression since the number of primitives is greatly reduced.

Early sample-based representations modeled the geometry simply as points [45, 73]. This includes images based representations with per-pixel depth [17, 26, 75, 77, 83, 103]. Recent research has grown in the direction of assigning a local region of influence to each point. The local region of influence can be surface-based or volume-based. Surface-based point representations model the surface around the point using a tangential disk [10, 48, 89, 95, 129], tangential ellipse [126], higher degree (3 or 4) polynomials [2, 39], or wavelet basis [119]. They approximate scanned datasets well at high resolutions and

are usually sensitive to noise. On the other hand volume-based representations such as spheres [99], image-based trees [17], and octree cells [11, 84, 89, 125] are topology blind and easy to organize hierarchically. These multiresolution organizations are designed with the rendering efficiency in mind. They use the hierarchical structure to achieve block culling, to control depth traversals to meet the image-quality or frame-rate constraints, and for efficient streaming of large datasets across the network [100]. For a complete reference on the data structures used for level-of-detail techniques we refer the readers to a survey by De Floriani *et al.* [40].

Current point-based representations are isotropic and therefore do not approximate the underlying data distributions compactly. Our surface-based context aware samples extend and generalize points with local surface context information. Our space-based context-aware samples extend the current volumetric point representations by use of anisotropy and statistical distribution of the vicinity. Uncertainties and noise in the data [57] are handled very well by our approach. Our statistical approach has some common elements to procedural rendering [35, 94] and the randomized z-buffer algorithm for triangle meshes [116]. 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 [102] uses it for anti-aliasing normals in bump-mapped environment mapping.

Geometry representation using context-aware samples has several benefits: (1) the local geometries of the context-aware samples can be handled entirely independent of each other and hence are well suited for modern Single Program Multiple Data (SPMD) GPUs and for network transmission, (2) they are procedural in nature and hence are not memory intensive, are fast, and offer direct rendering from compressed data, and (3) they offer a uniform framework for compressing other local attributes of the model such as color, normal, and texture coordinates.

2.1.3 Real-Time Rendering

Linear-interpolation based representations such as triangle mesh, tetrahedral mesh, and volumetric grids have traditionally been the preferred representation for rendering since they are simple to rasterize. However, the recent growth of datasets has called this into question. This is because the resolution of the data far outstrips the screen resolution and hence the average screen-space size of a triangle could be much smaller than a pixel. For such datasets it has been shown that it is more efficient to render the points (vertices) simply as square rectangles [99]. The quality of such rendering can be further enhanced by using splatting or blending on the screen [80, 89, 129] or by sampling points on their local polynomial (if any) [2]. Points can be rendered without any CPU involvement by storing the point geometry directly on the graphics card [10, 24, 48]. Temporal coherence can be exploited by keeping track of the visible Surfels in the frame buffer of successive frames [47]. Point primitives can also co-exist with triangles by leaving the representation of the smoother parts of the surface to the triangle mesh [18, 28]. In this dissertation we show how our surface-based context-aware samples make use of the procedural computation capabilities of modern GPUs for efficient rendering. We render space-based context-aware samples by generating new points according to their embedded statistical information. Our approach is inspired by prior work on procedural rendering [35, 94] and randomized z-buffer [116].

2.2 Differential Points

Our surface-based context-aware samples are based on the Taylor's series expansion. The Taylor's series expansion of a differentiable function $f(.)$ at a point x is given by:

$$f(x) = f(a) + \sum_{n=1}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n, \tag{2.1}$$

where $f(x)^{(n)}$ is the n-th derivative of $f(.)$ at x. In simple terms, the Taylor's series expansion says that if there exists a differentiable function $f(.)$ such that all of its derivatives are known at a point a, then the value of the function can be determined everywhere in

the domain. So one possible approach to representing visual data is to determine all of its derivatives at a point and then use it to determine its value everywhere else. However, in practice, it is hard to determine all the derivatives of the function at a point, or the function may not be infinitely differentiable. Alternately, the function can be represented by determining a finite set of derivatives, $f(a)^{(1)}$,..., $f(a)^{(p)}$, at a a finite set of points $a_0, \ldots, a_k$. Then the function can be represented by partitioning the domain such that each point x has an associated sample a_i with the function at that point being approximated by:

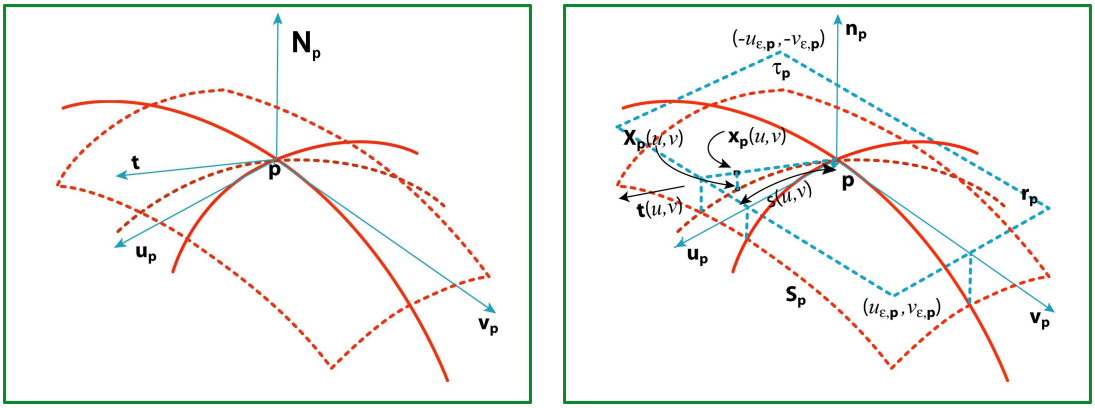
$$f(x) \approx f(a_i) + \sum_{n=1}^{p} \frac{f^{(n)}(a_i)}{n!}(x - a_i)^n. \tag{2.2}$$

We translate this approximation to the domain of the surfaces using the techniques of Differential Geometry [29]. Classical differential geometry is a study of the local properties of curves and surfaces. We limit ourselves to only the first two derivative in this approximation since higher derivatives may be hard to compute or may not exist. Moreover, using only two derivatives allows us to make use of the well understood mathematics of surface curvatures.

Given any point $\mathbf{p}$ on the surface, differential geometry gives us a tangential orthonormal basis consisting of the *direction of maximum curvature* ($\hat{\mathbf{u}}_\mathbf{p}$) and the *direction of minimum curvature* ($\hat{\mathbf{v}}_\mathbf{p}$). We denote the curvatures along these directions by $\lambda_{\mathbf{u_p}}$ and $\lambda_{\mathbf{v_p}}$ respectively. The relationship between these attributes can be summarized as follows:

$$\begin{aligned}
|\lambda_{\mathbf{u_p}}| &\geq |\lambda_{\mathbf{v_p}}| \\
\langle \hat{\mathbf{u}}_\mathbf{p}, \hat{\mathbf{v}}_\mathbf{p} \rangle &= 0 \\
\hat{\mathbf{u}}_\mathbf{p} \times \hat{\mathbf{v}}_\mathbf{p} &= \hat{\mathbf{n}}_\mathbf{p} \\
\mathbf{dN_p}(\hat{\mathbf{u}}_\mathbf{p}) &= -\lambda_{\mathbf{u_p}} \hat{\mathbf{u}}_\mathbf{p} \\
\mathbf{dN_p}(\hat{\mathbf{v}}_\mathbf{p}) &= -\lambda_{\mathbf{v_p}} \hat{\mathbf{v}}_\mathbf{p}
\end{aligned}$$

where $\langle \cdot, \cdot \rangle$ is the vector dot product, $\times$ is the vector cross product operator, $\hat{\mathbf{n}}_\mathbf{p}$ is the normal at $\mathbf{p}$, and $\mathbf{dN_p}(\hat{\mathbf{t}})$ is the first-order normal variation at the point $\mathbf{p}$ along the



(a) (b)

Figure 2.1: *(a) Neighborhood of a Differential Point. (b) Approximating the local geometry by a quadratic surface.*

direction $\hat{\mathbf{t}}$ (see Figure 2.1). The normal variation (gradient) along any unit tangent, $\hat{\mathbf{t}}$ (= $u\hat{\mathbf{u}}_\mathbf{p} + v\hat{\mathbf{v}}_\mathbf{p}$), at $\mathbf{p}$ can be computed as:

$$\begin{aligned} \mathbf{dN_p}(\hat{\mathbf{t}}) &= \mathbf{dN_p}(u\hat{\mathbf{u}}_\mathbf{p} + v\hat{\mathbf{v}}_\mathbf{p}) \\ &= u\mathbf{dN_p}(\hat{\mathbf{u}}_\mathbf{p}) + v\mathbf{dN_p}(\hat{\mathbf{v}}_\mathbf{p}) \\ &= -(\lambda_{\mathbf{u_p}} u\hat{\mathbf{u}}_\mathbf{p} + \lambda_{\mathbf{v_p}} v\hat{\mathbf{v}}_\mathbf{p}) \end{aligned} \tag{2.3}$$

Similarly, it can be shown that the normal curvature along $\hat{\mathbf{t}}$, $\lambda(\hat{\mathbf{t}})$, is given by [29]:

$$\lambda_\mathbf{p}(\hat{\mathbf{t}}) = \lambda_{\mathbf{u_p}} u^2 + \lambda_{\mathbf{v_p}} v^2 \tag{2.4}$$

The normal variation and the normal curvature terms give us second-order information about the behaviour of the regular surface around the point $\mathbf{p}$.

We use this information to construct a quadratic surface, $\mathbf{S_p}$, centered at the point which corresponds to a second order approximation of the surface in the vicinity of the point. This surface acts as our rendering primitive and we refer to it as a *differential point* (DP). We use upper-case characters or symbols for terms related to $\mathbf{S_p}$ and lower-case characters or symbols for terms related to the tangent plane $\tau_\mathbf{p}$ (a notable exception to this rule is the arc-length $s(u, v)$). The surface $\mathbf{S_p}$ is defined implicitly as follows: given any tangent $\hat{\mathbf{t}}$, the intersection of $\mathbf{S_p}$ with the normal plane of $\mathbf{p}$ that is co-planar with $\hat{\mathbf{t}}$ is

a semi-circle with a radius of $\frac{1}{|\lambda_{\mathbf{p}}(\hat{\mathbf{t}})|}$ with the center of the circle being located at $\mathbf{x_p} + \frac{\hat{\mathbf{n}}_{\mathbf{p}}}{\lambda_{\mathbf{p}}(\hat{\mathbf{t}})}$ and oriented such that it is cut in half by $\mathbf{x_p}$ (if $\lambda_{\mathbf{p}}(\hat{\mathbf{t}})$ is 0, then the intersection is a line along $\hat{\mathbf{t}}$). These terms are illustrated in Figure 2.1(b).

We parameterize the tangent plane $\tau_{\mathbf{p}}$ by the (u, v) coordinates in the vector space of $(\hat{\mathbf{u}}_{\mathbf{p}}, \hat{\mathbf{v}}_{\mathbf{p}})$. A point on $\tau_{\mathbf{p}}$ is denoted by $\mathbf{x_p}(u, v)$ and $\hat{\mathbf{t}}(u, v)$ denotes the tangent at $\mathbf{p}$ in the direction of $\mathbf{x_p}(u, v)$. We parameterize $\mathbf{S_p}$ with the same (u, v) coordinates as $\tau_{\mathbf{p}}$, with $\mathbf{X_p}(u, v)$ denoting a point on $\mathbf{S_p}$. The points $\mathbf{X_p}(u, v)$ and $\mathbf{x_p}(u, v)$ are related by a homeomorphic mapping, $\mathcal{P}_{\mathbf{p}}$, with $\mathbf{x_p}(u, v)$ being the orthographic projection of $\mathbf{X_p}(u, v)$ on $\tau_{\mathbf{p}}$ along $\hat{\mathbf{n}}_{\mathbf{p}}$. The arc length between $\mathbf{X_p}(0, 0)$ and $\mathbf{X_p}(u, v)$ is denoted by $s(u, v)$ and is measured along the semi-circle of $\mathbf{S_p}$ in the direction $\hat{\mathbf{t}}(u, v)$. The (un-normalized) normal vector at $\mathbf{X_p}(u, v)$ is denoted by $\mathbf{N_p}(u, v)$. Note that $\mathbf{x_p} = \mathbf{X_p}(0, 0) = \mathbf{x_p}(0, 0)$ and $\hat{\mathbf{n}}_{\mathbf{p}} = \mathbf{N_p}(0, 0)$.

The surface $\mathbf{S_p}$ is used to describe the spatial distribution around $\mathbf{x_p}$. We derive the normal distribution, $\mathbf{N_p}(u, v)$, around $\mathbf{x_p}$ using $\mathbf{S_p}$ and the curvature properties of the surface. To derive $\mathbf{N_p}(u, v)$ we express it in terms of its orthogonal components as follows:

$$\mathbf{N_p}(u, v) \;=\; \sum_{\hat{\mathbf{e}}=\hat{\mathbf{u}}_{\mathbf{p}},\hat{\mathbf{v}}_{\mathbf{p}},\hat{\mathbf{n}}_{\mathbf{p}}} \langle \mathbf{N_p}(u, v), \hat{\mathbf{e}} \rangle \, \hat{\mathbf{e}} \tag{2.5}$$

Consider the semi-circle of $\mathbf{S_p}$ in the direction $\hat{\mathbf{t}}(u, v)$. As one moves out of $\mathbf{x_p}$ along this curve the normal change per unit arc-length of the curve is given by the normal gradient $\mathbf{dN_p}(\hat{\mathbf{t}}(u, v))$. So, for a arc-length of $s(u, v)$, the normal can be obtained by using a Taylor's expansion on each individual component of equation (2.5) as follows:

$$\begin{aligned} \mathbf{N_p}(u, v) \;&=\; \sum_{\hat{\mathbf{e}}=\hat{\mathbf{u}}_{\mathbf{p}},\hat{\mathbf{v}}_{\mathbf{p}},\hat{\mathbf{n}}_{\mathbf{p}}} (\langle \mathbf{N_p}(0, 0), \hat{\mathbf{e}} \rangle + s(u, v) \, \langle \mathbf{dN_p}(\hat{\mathbf{t}}(u, v)), \hat{\mathbf{e}} \rangle + \text{Remainder Term}) \, \hat{\mathbf{e}} \\ &\approx\; \mathbf{N_p}(0, 0) + s(u, v) \, \mathbf{dN_p}(\hat{\mathbf{t}}(u, v)) \end{aligned} \tag{2.6}$$

The surface $\mathbf{S_p}$ and the normals $\mathbf{N_p}(u, v)$, give an approximation of the spatial and the normal distribution around $\mathbf{x_p}$. Note that $\mathbf{N_p}(u, v)$ is not necessarily the normal distribution of $\mathbf{S_p}$, but is just an approximation of the normals around $\mathbf{x_p}$. Since it is

only an approximation, there is a cost associated with this: the higher the arc-length, the higher the error in approximation and thus a bigger compromise in the visual quality after rendering. However an advantage to extrapolating to a larger neighborhood is that a smaller set of sampled DPs suffices to cover the whole surface, thus improving the rendering speed. We let the user resolve this tradeoff according to her needs by specifying two error tolerances that will clamp the extent of the extrapolation:

1. *Maximum principal error* (ϵ): This error metric is used to set point sizes according to their curvatures. It specifies a curvature scaled maximum orthographic deviation of $\mathbf{S_p}$ along the principal directions. We lay down this constraint as: $|\lambda_{\mathbf{u_p}}(\mathbf{X_p}(u,0) - \mathbf{x_p}(u,0))| \leq \epsilon$ and $|\lambda_{\mathbf{v_p}}(\mathbf{X_p}(0,v) - \mathbf{x_p}(0,v))| \leq \epsilon$. Since $\mathbf{S_p}$ is defined by semi-circles, we have that $\|\mathbf{X_p}(u,0) - \mathbf{x_p}(u,0)\| \leq \frac{1}{\|\lambda_{\mathbf{u_p}}\|}$. It follows that $\epsilon \leq 1$. In other words, the extrapolation is bounded by the constraints $|u| \leq u_{\epsilon,\mathbf{p}} = \frac{\sqrt{2\epsilon-\epsilon^2}}{|\lambda_{\mathbf{u_p}}|}$ and $|v| \leq v_{\epsilon,\mathbf{p}} = \frac{\sqrt{2\epsilon-\epsilon^2}}{|\lambda_{\mathbf{v_p}}|}$ as shown in Figure 2.1(a). This defines a rectangle $\mathbf{r_p}$ on $\tau_\mathbf{p}$ and bounds $\mathbf{S_p}$ accordingly since it uses the same parameterization. The ϵ constraint ensures that points of high curvature are extrapolated to a smaller area and the low-curvature points are extrapolated to a larger area.

2. *Maximum principal width* (δ): If $\lambda_{\mathbf{u_p}}$ is closer to 0, then $u_{\epsilon,\mathbf{p}}$ can be very large. To deal with such cases we impose a maximum width constraint δ. So $u_{\epsilon,\mathbf{p}}$ is computed as $\min(\delta, \frac{\sqrt{2\epsilon-\epsilon^2}}{|\lambda_{\mathbf{u_p}}|})$. Similarly, $v_{\epsilon,\mathbf{p}}$ is $\min(\delta, \frac{\sqrt{2\epsilon-\epsilon^2}}{|\lambda_{\mathbf{v_p}}|})$.

We call the surface $\mathbf{S_p}$ (bounded by the ϵ and δ constraints), the normal distribution $\mathbf{N_p}(u,v)$ (bounded by the ϵ and δ constraints) together with the rectangle $\mathbf{r_p}$ a *Differential Point* (DP) because all of these are constructed from just the second-order information at a sampled point. The above discussion has shown how the local surface and normal distribution can be represented and reconstructed using the curvature information. If the surface has additional attributes such as color, these attributes can be represented directly by using the Taylor's series approximation of equation 2.2 in the domain of the parametric space (u, v). Differential points are basic representational primitives. We will discuss how they can be used as building blocks to represent complex surfaces in chapter 3.

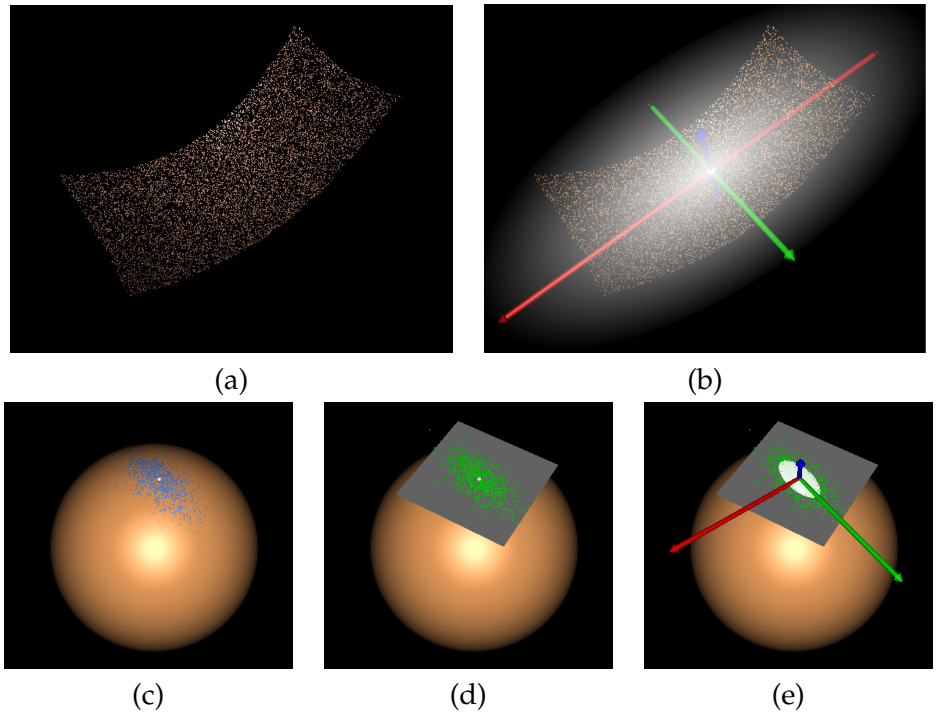


Figure 2.2: *Figure (a) shows a set of input points. Figure (b) shows the Gaussian approximation derived from the PCA analysis of the input points. Figure (c) shows the normals of a set of points on an unit sphere. The normals are shown in blue while the mean of the normals is shown in white. These normals are unwrapped to a tangent plane at the mean as shown with green points in Figure (d). Figure (e) shows the approximation of the normals by an ellipse on the tangent plane and a coordinate frame.*

2.3 Statistical Points

Prior work on statistical analysis has shown that if a set of samples exhibits a significant coherence or pattern, such a behaviour can be captured effectively using statistical models [31]. Though there are several powerful models for statistical analysis we use the simplest of them all: *Principal Component Analysis* (PCA) [31]. Our choice is guided by our desire to have a representational primitive that is not only compact but is also simple enough to be used as a rendering primitive.

The PCA of a set of N points in a d-dimensional space gives us the mean μ, an orthogonal frame f, and the standard deviation σ of the data [31]. The terms μ and σ are d-dimensional vectors and we refer to their i-th component as μ^i and σ^i respectively, where $\sigma^i \geq \sigma^j$ if $i > j$. The frame f consists of d vectors with the i-th vector referred to as f^i. In our case, the input is a set of N points with three attributes: spatial position p, normal n, and color c. We identify 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 anisotropic Gaussian distribution centered at μ_p, aligned in the directions f_p^1, f_p^2, and f_p^3, with the standard deviation along these directions being σ_p^1, σ_p^2, and σ_p^3, respectively (see Figures 2.2(a-b)). Such a distribution can be effectively visualized as an oriented ellipsoid with its intercepts being σ_p^1, σ_p^2, and σ_p^3 (see Figures 2.3(a) and 2.3(b)). Our approach can easily generalize to other local attributes such as texture coordinates.

The PCA in the RGB color space is similar to the PCA in the spatial dimensions: the (r, g, b) color values are treated as points in a three-dimensional space and a PCA in this space gives us its mean, μ_c, principal components, f_c, and the standard deviations, σ_c. We have to be a little more careful when doing PCA for the normals due to the normalization constraint. Normals can be seen as points on a unit sphere (see Figure 2.2(c-e)). We choose a longitudinal arc-length preserving parameterization because it allows us to map the Gaussian distribution from the tangent plane onto the sphere in such a way that the distribution on the sphere is also Gaussian. Note that this is not possible with mappings such as the orthographic mapping. We first orient the unit sphere such that its z-axis is along the average of the N normals (i.e. the north pole of the sphere is at the average normal). 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 though the latitudinal arc-lengths are not preserved. A PCA in this parametric space gives us an ellipse. The x- and the y-axes of the sphere are then rotated to be parallel to the axes of the ellipse. The PCA analysis of the normals thus gives us a 2D standard deviation vector σ_n and a 3D frame f_n (basis of the sphere). Note that the frame f_n effectively represents both the mean and the principal components of the normal distribution.

Since the PCA analysis is blind to surface geometry constraints it scales well to arbitrary geometries with complex topology. Moreover, we found that the PCA analysis

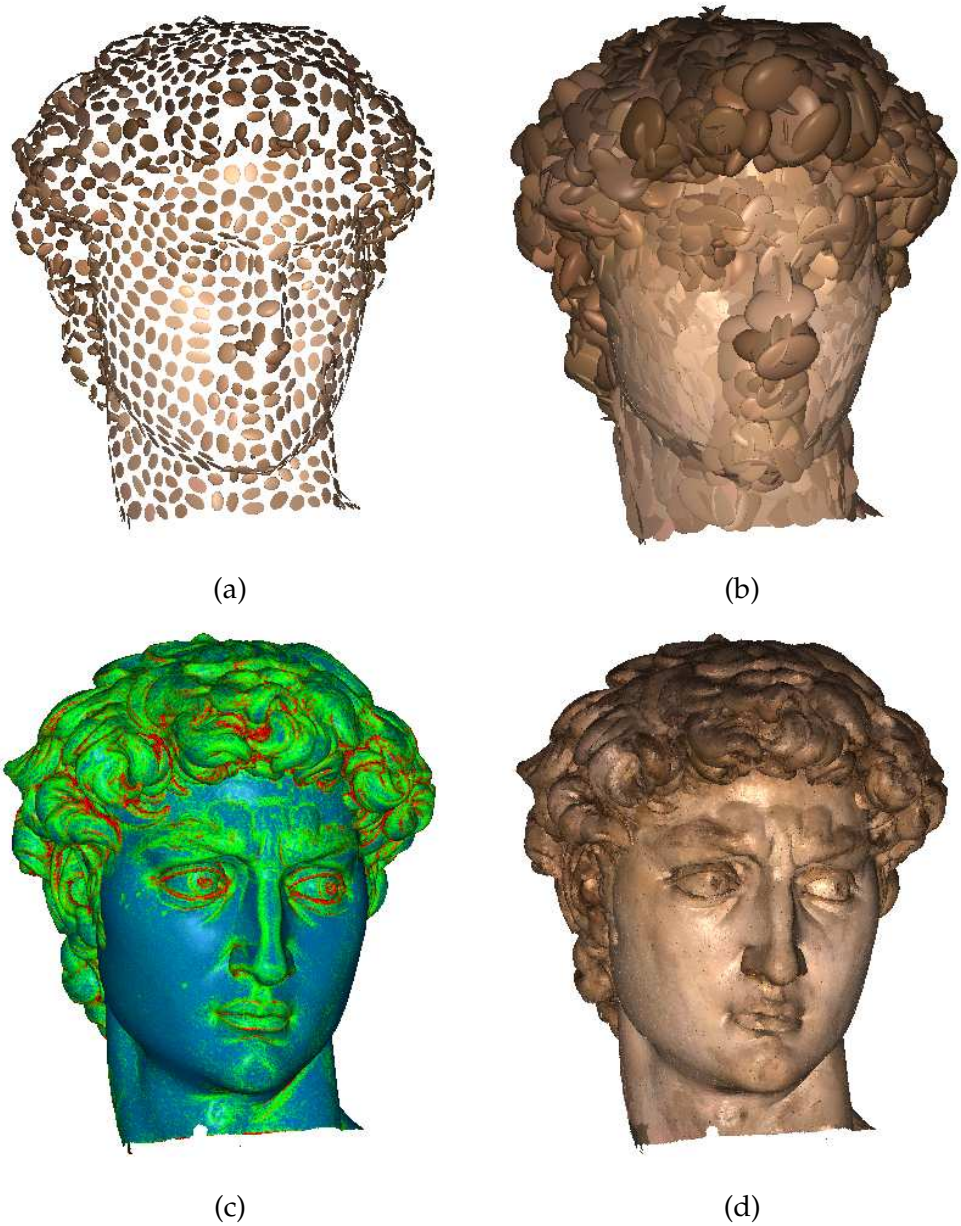


Figure 2.3: *Figure (a) shows the nodes at the mid-level resolution of the hierarchy built for the David's Head model. Each ellipsoid in this figure represents an anisotropic Gaussian distribution of the geometry with their intercepts being their corresponding standard deviation σ_p. The ellipsoids are colored by their mean color, μ_c. Figure (b) shows that scaling the ellipsoids by a factor $\gamma = 3.5$ ensures that the geometry is represented up to a Confidence Index (CI) of at least 99.7% (i.e., the ellipsoids enclose at least 99.7% of the cumulative Guassian distribution of the statistical points). Figure (c) shows the estimate of the local curvatures (the β factor) varying from high (red) to medium (green) to low (blue). Figure (d) shows the Gaussian distribution at the highest detail (after correction by the β factor).*

is a fast, simple, and robust procedure. We believe that this feature makes the PCA-based representation further attractive. However, a downside to this is that the nodes could protrude out of the surface in regions of high curvature. To correct this we observe that the ratio $\frac{\sigma_n^1}{\sigma_p^1}$ is a good estimate of the surface curvature since it captures the variation of the surface normal (see Figure 2.3(c)). Hence we scale the value of σ_p by the factor $\beta = [\eta_0 + (1-\eta_0)\min(0, (1-\eta_1\sqrt{\frac{\sigma_n^1}{\sigma_p^1}}))]$, where $0 < \eta_0, \eta_1 < 1$. The $(1-\eta_1\sqrt{\frac{\sigma_n^1}{\sigma_p^1}})$ term reduces the width of the node if the curvature is high. However, in some cases, the $(1 - \eta_1\sqrt{\frac{\sigma_n^1}{\sigma_p^1}})$ factor leads to relatively small nodes for lower resolution nodes of the hierarchy (see §3.2.1). We avoid this by using the η_0 factor which determines the proportion of the original PCA-derived width that is retained even after the curvature-related reduction of the node. Although the curvature estimate is given by $\frac{\sigma_n^1}{\sigma_p^1}$, scaling σ_p by its square root gave us better visual results. We have used values of $\eta_0 = \frac{1}{2}$ and $\eta_1 = \frac{1}{6}$ for all our experiments.

2.3.1 Unified-Attribute Statistical Samples

In the above case we did a PCA separately in each of the individual attribute spaces. This has several advantages for real-time rendering that we will discuss later. However, this approach has the disadvantage that it drops important information about the correlation between the individual attribute distributions. This can be overcome by doing a PCA in the unified space of all the attributes.

Consider a PCA analysis of the points, $\mathbf{x}_i = (x_i, y_i, z_i, \theta_i, \phi_i, r_i, g_i, b_i), \forall i = 0, \ldots, N$, in the 8D space of position (3D), normal (2D), and color (3D). Here the normals are represented by their angles $(\theta, \phi) \in ([0, \pi], (-\pi, \pi])$. A PCA analysis in this space first requires us to compute the mean. The mean in this case is the Euclidean mean in all the dimensions except in the (θ, ϕ) dimension of the normal – a spherical space. We compute the mean normal using the approach proposed by Buss and Fillmore [15]. They have outlined a method that computes weighted averages on spheres based on least squares minimization that respects spherical distances by using the logarithmic map and its inverse, the exponential map. We represent the mean normal by its angles, (μ_θ, μ_ϕ). The

next step in the PCA analysis is the computation of the covariance matrix. This requires us to define the distance vector, $\mathbf{x}_i - \mu$, between a point in space and the mean. The individual components of this difference vector are the standard Euclidean difference in all the dimensions except for the (θ, ϕ) normal space. For the difference vector in the normal space we simply use the 2D coordinates in the logarithmic space defined on the plane tangent to the unit sphere centered at the mean normal (μ_θ, μ_ϕ) [15]. The rest of the PCA analysis proceeds as usual. The eigenanalysis of the covariance matrix gives us eight 8D eigenvectors and the variances of the Gaussian distribution along these vectors. Since this approach is so general, it may also be used to represent other visual data such as light fields and surface reflectance.

2.4 Comparison and Summary

In this chapter we introduced context-aware samples. We covered previous related work in this area and discussed why context-aware samples are needed for representing large visual data. We detailed two kinds of context-aware samples. Differential points are surface-based CAS that approximates a quadratic surface around a sample point using second-order differential information at the point. Statistical points model the local context around a point using principal component analysis. Differential points are well suited for well defined surfaces since the surface coherence can be most effectively captured using this approach. On the other hand, statistical points are very robust and general and hence are suited even for ill-defined or under sampled surfaces. Statistical points may also easily generalize to a wide variety of visual information such as 3D volumes, light fields, and surface reflectance data.

Chapter 3

Geometry Representation using Context-Aware Samples

Context-aware samples (CAS) are basic primitives which act as the building blocks for representing large and complex visual data. In the previous chapter we discussed the nature and attributes of two kinds of CAS. In this chapter we will discuss how large visual data can be modeled using those CAS primitives. Modeling visual data using CAS primitives requires a careful attention to several factors such as sampling the visual data for context-aware samples, determining the attributes of the sampled CAS, handling redundancy due to the presence of multiple CAS, and building a multiresolution hierarchy over the CAS-based representation.

3.1 Differential Point Geometry

Differential Points model the surface vicinity around a sampled point. The overall surface is modeled in parts by the individual surfaces of the differential points. If the sampling of the surface is not based on the surface curvature, there can be a significant redundancy in the surface representation. This is because a region of the surface could be represented by several differential points. We minimize this redundancy with a simplification process which uses a greedy heuristic to prune the redundant differential points.

3.1.1 Sampling

In order to represent a 3D surface with differential points we first sample it. If the surface is a parametric one, such as NURBS, we sample it uniformly in the parametric domain. We use the standard techniques outlined in the differential geometry literature [29] to extract surface-curvature-related information at each sampled point. This is a fairly robust procedure that can handle most of the sampled points. However, degeneracies can arise in the form of umbilical points where the surface is either flat or spherical ($\lambda_{\mathbf{u_p}} = \lambda_{\mathbf{v_p}}$).

We derive the principal directions of such points implicitly by assigning $\hat{\mathbf{u}}_\mathbf{p}$, the direction of maximum curvature, to be the best among the projections of the x, y, and the z-axes onto the tangent plane. We then compute $\hat{\mathbf{v}}_\mathbf{p}$, the direction of minimum curvature, as $\hat{\mathbf{v}}_\mathbf{p} = \hat{\mathbf{n}}_\mathbf{p} \times \hat{\mathbf{u}}_\mathbf{p}$.

Alternately, if the surface is represented as a triangle mesh, then a NURBS surface can be fit to the triangle mesh [69] and points can be sampled using this representation. We use a more direct approach by using the vertices of the mesh as the sampled points and estimating the differential information for each point using the discrete differential geometry operators of Taubin [109]. The DPs thus obtained from the triangle mesh have the same properties as the ones obtained by sampling a NURBS surface.

3.1.2 Handling undersampling

Since the sampling is input driven (eg. for triangle meshes) or user driven (eg. through the sampling rate on a NURBS surface), it happens quite often that in some areas of the surface the samples may be spaced far apart even though the surface curvature of that region is high. If the points of such areas were to be assigned sizes using the criterion described in section 2.2 then there might be gaps in the surface coverage. This is because the surface $\mathbf{S}_\mathbf{p}$ of the DPs are bounded and may not overlap sufficiently, thus leading to gaps in the surface coverage. We deal with this issue by factoring in the distance of the vertex neighbors into the size of $\mathbf{S}_\mathbf{p}$. For each mid point $\mathbf{x}_\mathbf{p}$ of a differential point $\mathbf{p}$, we project the mid-point of every incident edge along the average of its adjacent triangle normals onto the tangent plane $\tau_\mathbf{p}$ of the DP. Then we determine a rectangle on the tangent plane $\tau_\mathbf{p}$ which encloses all the projected points. We choose the parameters of the rectangle to be such that: (1) it is axis aligned with respect to the $(\hat{\mathbf{u}}_\mathbf{p}, \hat{\mathbf{v}}_\mathbf{p})$ directions, (2) it is symmetric with respect to the origin (point $\mathbf{x}_\mathbf{p}$), and (3) its size is the smallest possible. We set the rectangle $\mathbf{r}_\mathbf{p}$ to be the larger of the rectangle computed this way and the rectangle determined from the δ and ϵ parameters (see §2.2).

(a) Without simplification (b) With simplification (c) Rendering with simplification

Figure 3.1: *Effectiveness of Simplification: (a) Wireframe rectangles corresponding to the initial (supersampled) collection of differential points from the surface of the teapot. (b) Wireframe rectangles of the differential points that are not pruned by the simplification algorithm. Simplification is done within a patch and not between patches. The strips of rectangles represent the differential points on the patch boundaries. (c) A rendering of the simplified differential point representation.*

3.1.3 Handling oversampling: Simplification

Often times some areas of the surface are sampled at a far higher rate than what is mandated by the local curvature complexity. This is especially true when we supersample the NURBS surface where our intention is to be conservative and ensure that the rectangle of each differential point overlaps sufficiently with its neighbors so that there are no holes in the surface coverage. In this section we discuss a greedy procedure that minimizes the overlap of DPs by pruning the redundant DPs. The output of this simplification process is a reduced set of DPs that represents the surface within a margin of error.

The objective of our simplification process is to prune those DPs whose geometric information is represented by the cumulative information of their neighbors within the error margins prescribed by the values of ϵ and δ. Simplification has to ensure that the output set of DPs represent the original surface without leaving any holes. This requires us to first define the region of the original surface that is covered by a DP. We do this by defining a projection set, $\mathcal{O}(\mathbf{p})$, to be the set of all points of the original surface in the vicinity of $\mathbf{x_p}$ that fall within the surface area covered by the orthographic projection of the rectangle $\mathbf{r_p}$ onto the original surface along the direction $\hat{\mathbf{n}}_\mathbf{p}$. Do Carmo [29] shows that for a vicinity around the point position $\mathbf{x_p}$, this projection (mapping) is a homeomorphism. We define an *overlap* relation between differential points as follows: A differential point $\mathbf{p}$ is said to *overlap* another differential point $\mathbf{q}$ iff $\mathcal{O}(\mathbf{p}) \cap \mathcal{O}(\mathbf{q}) \neq \phi$. It follows from

the definition that *overlap* is a symmetric relation.

Let $\mathcal{N}(\mathbf{p})$ denote the set of neighbors of a DP, $\mathbf{p}$. At the start of the simplification we initialize $\mathcal{N}(\mathbf{p})$ to include all the immediately surrounding DPs that overlap $\mathbf{p}$. If the DPs are sampled from a parametric surface, then $\mathcal{N}(\mathbf{p})$ is chosen from the 8, or the 24 nearest samples from the sampling grid of the parametric domain (DPs in the boundary can have less than 8 immediately surrounding DPs). Since overlap is a symmetric relation, we have that $\mathbf{q} \in \mathcal{N}(\mathbf{p})$ iff $\mathbf{p} \in \mathcal{N}(\mathbf{q})$. Alternately, if the differential points are sampled from a triangle mesh, then $\mathcal{N}(\mathbf{p})$ is chosen from the vertices with whom $\mathbf{p}$ shares edges. Later, when the simplification algorithm is in progress, in the event of any $\mathbf{q}_i \in \mathcal{N}(\mathbf{p})$ being pruned, $\mathcal{N}(\mathbf{p})$ is updated as follows:

$$\mathcal{N}(\mathbf{p}) \Leftarrow \mathcal{N}(\mathbf{p}) - \{\mathbf{q}_i\} \cup \{\mathbf{q}_j | \mathbf{q}_j \in \mathcal{N}(\mathbf{q}_i) \text{ and } \mathbf{q}_j \neq \mathbf{p} \text{ and } \mathcal{O}(\mathbf{q}_j) \cap \mathcal{O}(\mathbf{p}) \neq \phi\} \tag{3.1}$$

This operation deletes $\mathbf{q}_i$ from $\mathcal{N}(\mathbf{p})$ and updates it to include all the neighbors of $\mathbf{q}_i$ that overlap with $\mathbf{p}$. Lastly, we define a term that will act as the prunability criteria of a DP. A differential point $\mathbf{p}$ is said to be *enclosed* iff $\mathcal{O}(\mathbf{p}) \subseteq \bigcup_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} \mathcal{O}(\mathbf{q})$. In other words, $\mathbf{p}$ is said to be enclosed iff each point in its projection set $\mathcal{O}(\mathbf{p})$ is also in the projection set of atleast one of its neighbors. During simplification we check to make sure that only enclosed DPs are pruned.

Our simplification algorithm uses a greedy heuristic of pruning the most *redundant* point first. A DP's redundancy is a measure of how similar it is with respect to its neighbors. We quantify it by a metric, called the *redundancy factor*, $\mathcal{R}(\mathbf{p})$, which quantifies the ability of $\mathbf{p}$ to approximate the normal of its neighbors and vice versa. The higher the value of $\mathcal{R}(\mathbf{p})$, more is the redundancy of $\mathbf{p}$. The term $\mathcal{R}(\mathbf{p})$ is computed as follows:

$$\mathcal{R}(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} \left(\frac{|\langle \mathbf{N_q}, \mathbf{N_p}(u_{\mathbf{q},\mathbf{p}}, v_{\mathbf{q},\mathbf{p}}) \rangle| + |\langle \mathbf{N_p}, \mathbf{N_q}(u_{\mathbf{p},\mathbf{q}}, v_{\mathbf{p},\mathbf{q}}) \rangle|}{2\,|\mathcal{N}(\mathbf{p})|} \right) \tag{3.2}$$

where $(u_{\mathbf{q},\mathbf{p}}, v_{\mathbf{q},\mathbf{p}})$ is the coordinates of the point on $\tau_{\mathbf{p}}$ obtained by the orthographic projection of $\mathbf{x_q}$ onto $\tau_{\mathbf{p}}$ and $\mathbf{N_p}(u_{\mathbf{q},\mathbf{p}}, v_{\mathbf{q},\mathbf{p}})$ is the normal estimated at these coordinates using equation (2.6). The dot product $|\langle \mathbf{N_q}, \mathbf{N_p}(u_{\mathbf{q},\mathbf{p}}, v_{\mathbf{q},\mathbf{p}}) \rangle|$ in equation (3.2) is a measure

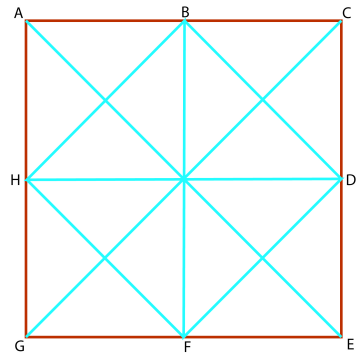
```
Simplification( )
∀ DP p
    Establish N(p)
    Compute R(p)
    Insert p in the Heap with R(p) as the key
    (Highest key is at the top of the Heap)
While Heap is not empty
    p = pop Heap
    if Enclosed(p)
        ∀ q ∈ N(p)
            delete p from N(q)
            undo influence of p on R(q)
            balance Heap
        ∀ distinct q1, q2 ∈ N(p)
            if (Overlaps(q1,q2))
                add q2 to N(q1)
                update R(q1) and balance Heap
                add q1 to N(q2)
                update R(q2) and balance Heap
        delete p
    else
        add a pointer of p to the OutputList
        (p is not pruned)
return OutputList
```

Figure 3.2: *A pseudo code of our algorithm for simplifying a set of DPs.*

of how close the actual normal at $\mathbf{q}$ is to the normal estimated at $\mathbf{x_q}$ using the curvature information at $\mathbf{p}$. If $\mathcal{R}(\mathbf{p})$ is closer to 1 then $\mathbf{p}$ is redundant because all the geometric information of $\mathbf{p}$ is already represented by its neighbors.

We start the simplification process with a binary heap of DP's with their respective redundancy factors $\mathcal{R}(\mathbf{p})$ as the key. Iteratively we pop the top of the heap and check if pruning that DP will leave any holes in the surface representation. If not, we prune that DP and update the neighborhood and the redundancy factor of all its ex-neighbors using equations (3.1) and (3.2). If the DP cannot be pruned then we mark it for output. A pseudo-code of our simplification algorithm is shown in Figure 3.2.

For $\mathbf{p}$ to be pruned it has to satisfy the correctness check: that $\mathbf{p}$ is an enclosed point, or in other words that the pruning of $\mathbf{p}$ does not leave a hole in the surface representation. This check is done by the routine Enclosed($\mathbf{p}$) of the simplification pseudo-code. Testing for the enclosure of the surfaces $\mathbf{S_p}$ can be very expensive. Instead, we approximate the



Enclosed(p)

```
1   TestLines = Sampled line segments on r_p
2   ∀q ∈ N(p)
3       ∀l ∈ TestLines
4           delete l from TestLines
5           project l along n_p onto τ_q
6           clip it against r_q
7           project back the leftover segments along n_p
            onto τ_p
8           add them to TestLines
9       if (TestLines is empty)
10          return(true)
11  return(false)
```

Figure 3.3: *The figure on the left shows the initial test line segments (AE, CG, BF, HD, BH, HF, FD, DB) of the rectangle. They are used in the Enclosed(**p**) routine (line 1) which tests if a DP, **p**, is enclosed by its neighbors.*

original surface by $\tau_{\mathbf{p}}$, and test for the enclosure of $\mathbf{p}$ within this. This test is done by an approximation method that samples line segments on $\mathbf{r_p}$ (as shown in Figure 3.3) and tests if they are fully covered by the rectangles of the DPs $\in \mathcal{N}(\mathbf{p})$. A pseudo-code of this test is shown in Figure 3.3. Since the coverage of line segments does not guarantee coverage of the entire area of $\mathbf{r_p}$, we see infrequent sliver gaps left between rectangles. We make the coverage test more conservative by scaling down the rectangles for simplification (the original rectangle sizes are used for rendering). For all our test models, a scale-down factor of 15% produced a hole-free and effective simplification. The simplification algorithm also involves a test for the overlap of $\mathbf{q}_1$ and $\mathbf{q}_2$. An approximate test for this is done by the routine Overlaps($\mathbf{q}_1$,$\mathbf{q}_2$) of the simplification pseudo-code which tests if $\mathbf{r}_{\mathbf{q}_1}$ overlaps $\mathbf{r}_{\mathbf{q}_2}$ when $\tau_{\mathbf{q}_2}$ is assumed to be the original surface and vice versa. An approximation algorithm for this test is shown in Figure 3.4.

All the approximation algorithms work well in our case owing to the similarity of neighboring rectangles in position, width, and orientation. Figure 3.1(b) shows the rectangles left after simplification in an area where curvature-related features change very quickly. A desirable feature of this simplification process is that the error metrics that it uses also control the quality of the final rendered images. This allows the user to first decide on the image quality and then get as much simplification as possible without any

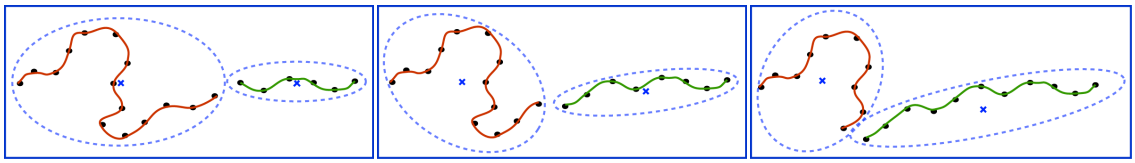
Overlaps($\mathbf{q}_1$, $\mathbf{q}_2$)

1 return (OverlapTest($\mathbf{q}_1$, $\mathbf{q}_2$) || OverlapTest($\mathbf{q}_2$, $\mathbf{q}_1$))

OverlapTest($\mathbf{q}_1$, $\mathbf{q}_2$)

1 If the orthographic projection of $\mathbf{x}_{\mathbf{q}_1}$ onto $\tau_{\mathbf{q}_2}$ falls within the bounds of $\mathbf{r}_{\mathbf{q}_2}$ then return true
2 If the orthographic projection of any of the end points of $\mathbf{r}_{\mathbf{q}_1}$ onto $\tau_{\mathbf{q}_2}$ falls within the bounds of $\mathbf{r}_{\mathbf{q}_2}$ then return true
3 If the orthographic projection of any of the edges of $\mathbf{r}_{\mathbf{q}_1}$ onto $\tau_{\mathbf{q}_2}$ intersects $\mathbf{r}_{\mathbf{q}_2}$ then return true
4 If all the above tests fail then return false

Figure 3.4: *Pseudocode of the Overlaps test which checks if two DPs, $\mathbf{q}_1$ and $\mathbf{q}_2$, overlap each other.*



(a) (b) (c)

Figure 3.5: *These figures illustrate three iterations of the clustering algorithm used for spatial partitioning of a set of points. Successive iterations reduce the distortion between the original set of points and the cluster centers (shown as blue crosses).*

loss in visual quality. Figure 6.1 shows sample models rendered with and without simplification.

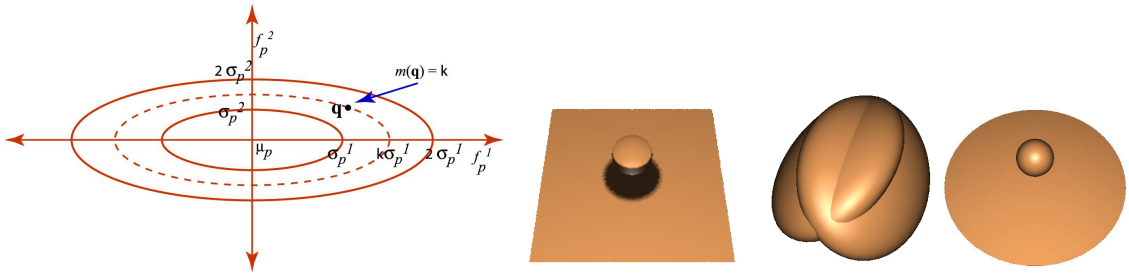
3.2 Visual Data Representation using Statistical Points

The clear advantage of the PCA representation over the surface-based differential point representation is that it can model any arbitrary collection of points rather than just a quadratic surface area. However a PCA representation of a complex set of points such as the David's Head, though compact, is clearly a coarse approximation. In this section we discuss how a large set of points can be modeled by local PCA representation. Further, our modeling method is hierarchical which gives us a multiresolution representation of the data. We approximate the original set of points by generating points from our hierarchical PCA representation.

3.2.1 Hierarchical PCA

To represent the data at different levels of detail we build a spatial hierarchy based on partitioning the input points. We compactly represent each node of the hierarchy using statistical neighborhood modeling as discussed in §2.3 and §2.3.1. We build our hierarchy in a top-down fashion by partitioning the points at each node into two sets. We prefer a top-down approach to a bottom-up approach since it organizes the expensive nearest-neighbor searches hierarchically. We use a 2-means clustering approach to partition the points into a binary hierarchy. An alternate approach would have been to use a k-means clustering where k is a variable number of partitions. However, such an approach would would be more suited to segmenting the geometry as opposed to building a balanced hierarchy.

The *distortion* of a partitioning is defined as the sum of the distances of the points from the partition's mean [31]. In our partitioning scheme we reduce this distortion by using k-means clustering with $k = 2$. We initialize the two starting means (centers) for the k-means algorithm by doing a PCA over the points and choosing $\mu_p + \frac{\sigma_p^1}{2} f_p^1$ and $\mu_p - \frac{\sigma_p^1}{2} f_p^1$ as the initial guesses. This is a reasonable assumption since the data varies maximally along f_p^1. The k-means clustering algorithm then iterates over the twin steps of partitioning the point set according to the proximity of each point to the two means and then updating the two means according to this partitioning. Figure 3.5 illustrates three iterations of the clustering algorithm. Pauly *et al.* [87] use a geometric method to separate the point set for their point-based simplification hierarchy. They separate along the principal direction f_p^1 with the separating plane passing through the mean μ_p. A similar strategy is used by Brodsky and Watson [14] for hierarchical mesh partitioning. This approach is equivalent to the first iteration of the clustering scheme. The subsequent iterations then successively reduce the distortion. We stop iterating when the difference in the average distortion between two successive iteration is less than 10^{-7} or when the number of iterations is more than 30, whichever happens earlier. Our clustering step can be made more efficient using the technique proposed by Kanungo *et al.* [62]. We terminate the hierarchical partitioning at nodes which have less than a user-specified



(a) (b) (c) (d)

Figure 3.6: *Figure (a) illustrates the Mahalanobis distance, $m(\mathbf{q})$, between a point $\mathbf{q}$ and the mean μ_p of a PCA node. Figure (b) is simple model of a sphere and a plane. Figure (c) is the partitioning of the sphere and plane model obtained by a partitioning-plane-based approach while Figure (d) is the partitioning obtained by the Mahalanobis-distance-based approach. The partitions have been rendered by the ellipsoids corresponding to their respective PCA attributes (mean μ_p, standard deviation σ_p, and principal components f_p).*

number of points (between 6 and 30 for our models).

Choosing the distance metric is a crucial issue when designing the clustering algorithm. The Euclidean distance metric is a good metric in most instances and also produces a balanced tree. However, it has a tendency to merge disjoint parts of the surface if they are close enough (see Figure 3.6(c)). This can be rectified by the Mahalanobis distance metric [31]. The Mahalanobis distance metric warps the space so that distances along the normal direction are weighed much higher than the distances along the tangential directions (see Figure 3.6(a)). The Mahalanobis distance between a point $\mathbf{q}$ and the mean μ_p of a PCA node is determined by the product of two matrices: an affine transformation matrix and a scaling matrix. The affine transformation matrix transforms the point to the coordinate frame defined by the pair (μ_p, f_p). We denote this transformation by T_p. The point is then scaled using the scaling matrix, S_p, given by:

$$S_p = \begin{bmatrix} \frac{1}{\sigma_p^1} & 0 & 0 \\ 0 & \frac{1}{\sigma_p^2} & 0 \\ 0 & 0 & \frac{1}{\sigma_p^3} \end{bmatrix}.$$

The Mahalanobis distance, $m(\mathbf{q})$ between $\mathbf{q}$ and μ_p is then simply given by, $m(\mathbf{q}) = \|S_p\, T_p\, \mathbf{q}\|_2$. This is illustrated in Figure 3.6(a).

The Mahalanobis distance metric generally leads to partitions that do not merge

Figure 3.7: *The Lucy model at various resolutions. Each ellipsoid in this figure represents an anisotropic Gaussian distribution of the geometry with their intercepts being their corresponding standard deviation σ_p. The ellipsoids are colored by their mean color, μ_c.*

disjoint parts of the surface. This is because the Mahalanobis distance metric measures distances respecting the local anisotropy of the partitions that the Euclidean metric is unable to do. However, we note here that the use of the Mahalanobis metric in partitioning is still a heuristic, although generally a better one than the Euclidean metric. When the surface is too complex to be neatly partitioned into two clearly disjoint surfaces, the use of the Mahalanobis distance metric can produce an imbalanced partitioning. Hence we use a two-pronged strategy: we first try a k-means clustering based on the Mahalanobis metric and if that partitioning turns out to be imbalanced we switch to a Euclidean-distance-based partitioning. The definition of an imbalanced partition is left to the user (we used a balancing threshold of 30% – 70% for all our models).

3.2.2 Detail Evaluation

The hierarchical PCA gives us a hierarchical Guassian probability distribution, with the probability distribution of each node given by:

$$p(x) = \frac{1}{(2\pi)^{d/2}|\sum|^{1/2}} e^{-(x-\mu)^T \sum^{-1}(x-\mu)}$$

where d is the dimensionality of the attribute and $\sum$ is the covariance matrix of the attribute values. We approximate the original set of points by generating new points. The attributes of the new points are determined by independently sampling the probability distributions in the individual attribute spaces. We determine the position attribute of the generated points by using a three-dimensional extension of the Box-Muller transform [13, 106, 123]:

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

where r_{p0}, r_{p1}, and r_{p2} are uniformly distributed random numbers in $(0, 1]$, $[0, 1]$, and $[-1, 1]$, respectively and $\tau_p = \sqrt{-2\ln(r_{p0})}$. This sampling uses a uniform parameterization of a unit sphere by using a $(\cos(\theta), \phi)$ spherical parameterization. The random values r_{p1} and r_{p2} are samples in this parameter space and spread points uniformly on the unit sphere. The value τ_p then ensures that the radial distances of the points from the mean are spread in a Gaussian manner while the scaling matrix gives the anisotropic nature to the sampling. We determine the color of these generated points by independently sampling their color space. To determine the normals of the generated points we use the Box-Muller transform to sample the tangent plane positioned at the mean normal. These points are then wrapped onto the unit sphere to reverse the sphere-to-tangent-plane mapping discussed in §2.3. The entire normal sampling procedure is given by the following set of equations that derive the (θ, ϕ) values of the normals:

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

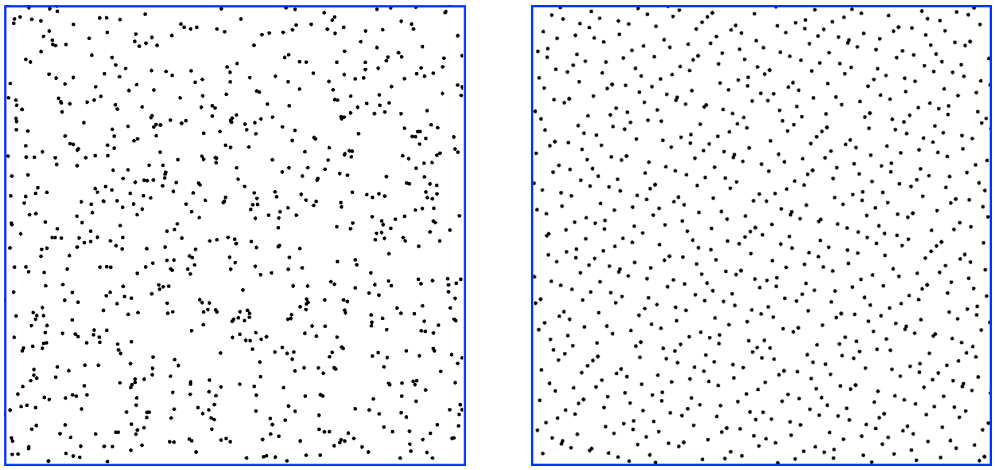
where $r_{n0} \in (0, 1]$ and $r_{n1} \in [0, 1]$ are uniform random numbers. Here, the term r_{n0} uniformly spreads the samples around the center of the tangent plane while τ_n radially distorts them to behave as a Gaussian distribution of unit variance. The (α, β) values model an anisotropic Gaussian distribution on the tangent plane. The (θ, ϕ) values are then obtained by wrapping the (α, β) values to the sphere.

The above scheme for sampling assumes that all the variances are non-zero. However, in practice we found several nodes with one or more zero variances. To deal with zero variances of σ_p^i we have a minimum threshold value (of the order 10^{-15}). Any σ_p^i is set to the maximum of itself and this threshold value. This allows us to consider only ellipsoidal (Gaussian) distributions (even if they are vanishingly thin along some dimensions) without having to worry about special cases. When there are two zero variances, we retain the principal direction derived from eigen-analysis and set the other two directions so that the z-direction of the ellipsoid points along the average normal. For the case of 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. Handling the zero variances of σ_c^i and σ_n^i is a little easier since there is no correct orientation of their principal vectors under such degeneracies (such as for the case of σ_p^i). We simply have a minimum threshold for these values.

The above guidelines for sampling the distributions raises two important questions: (1) how to minimize the distortion in the Gaussian sampling, and (2) how to minimize the number of generated points. We discuss these issues next.

3.2.3 Quasi-Random Sampling

The quality of the sampling is linked to the quality of the random number generator. While pseudo-random numbers have been used extensively for various applications, they have a high discrepancy owing to the independent sampling of each pseudo-random number [81]. In other words, deriving each pseudo-random sample independent of previous pseudo-random samples produces a non-uniform distribution as shown in Figure 3.8(a). Quasi-random numbers have been used as a substitute for pseudo-random



(a) (b)

Figure 3.8: *Figure (a) shows 800 points generated in a two-dimensional space from a pseudo-random sequence while Figure (b) shows 800 points generated using quasi-random numbers. Quasi-random numbers are preferable since they show low discrepancy which results in a more uniform distribution.*

numbers since they exhibit low discrepancy (see Figure 3.8(b)) [92]. Quasi-random numbers are generated from algebraic sequences such as the Sobol sequence [91]. They have two main features: (1) they do not have a seed value and hence generate the same random number sequence each time, and (2) successive random numbers are aware of the random numbers that were generated earlier and hence are placed so as to minimize the discrepancy. Quasi-random numbers have been used successfully in computer graphics, for instance in the Monte-Carlo integration for global illumination [65].

We use quasi-random numbers for sampling our attributes. For computing the spatial coordinates and the color of the generated points we use three-dimensional quasi-random numbers. For computing the normals of the generated points we use two-dimensional quasi-random numbers. Quasi-random numbers easily fit into our scheme of sampling the Gaussian distribution: we simply replace the pseudo-random numbers, r, with the quasi-random numbers in the sampling equations above.

3.2.4 Determining the Number of Samples

The points that we generate from the Gaussian distributions approximate the original geometry and we visualize a PCA node by rendering these generated points. While it is possible to generate any number of points, we would like to minimize this number

α^1	α^2	α^3	# pseudo	# quasi
1	1	1	1	1
2	1	1	2	2
3	1	1	6	4
4	1	1	15	4
2	2	1	3	3
3	2	1	9	7
4	2	1	23	10
3	3	1	26	7
4	3	1	51	12
4	4	1	83	26

α^1	α^2	α^3	# pseudo	# quasi
2	2	2	3	3
3	2	2	12	4
4	2	2	14	4
3	3	2	25	7
4	3	2	36	12
4	4	2	77	26
3	3	3	28	14
4	3	3	48	28
4	4	3	89	26
4	4	4	81	54

Table 3.1: *This table shows the relationship between the screen-space dimensions of an ellipsoid (in pixels) and the minimum number of generated points required to cover the screen-space projection of the ellipsoid. Here* $\alpha^i = \gamma \times \sigma_p^i$*. The "*# pseudo*" column refers to the number of samples required if pseudo-random sampling were to be used while the "*# quasi*" column refers to the case of quasi-random sampling. The points are rendered with a screen diameter of* 1.8 *pixels.*

since it directly affects the rendering cost. We link the number of generated points to the screen space dimensions of the PCA node. This is similar in spirit to the Randomized Z-Buffer idea by Wand *et al.* [116]. They uniformly sample a triangle mesh by using an analytical formula to decide the number of points to sample. In particular, if the triangle mesh projects to p pixels, they uniformly sample $O(p \log p)$ points on their triangle mesh. However, since our sampling is on a per-node basis, we have the flexibility of precomputing this relationship so that we can efficiently look it up at runtime. We use an empirical approach to build our lookup table. We choose an empirical approach over an analytic approach because the former allows us to account for the constants as well as the non-linear relationship introduced by a diverse set of factors such as discrete rasterization, hardware anti-aliasing, and the use of quasi-random numbers.

We have set up an empirical testbed where PCA nodes with different σ_p attributes are orthographically projected along their f_p^3 vector. Table 3.1 shows the relationship between σ_p and the number of points to be generated to completely cover the projection of an ellipsoid with dimensions of $\gamma \times \sigma_p$. The multiplicative factor of γ is important here. The Gaussian distribution never really goes to zero and one will have to generate an infinite number of points to cover the entire distribution. However it can be shown that the region enclosed by $\gamma = 3.5$ has a Confidence Index (CI) of at least 99.7%, i.e., it covers

at least 99.7% of the distribution. Hence we limit ourselves to generating enough points so that the screen-space area occupied by this enclosed region is covered. At render-time we estimate the z-distance of the mean μ_p from the camera and estimate the dimensions on the screen to be $\lceil F\sigma_p/z \rceil$, where F is the distance between the center of projection and the view plane. We use this to index the table for determining the number of points to generate. Our z-distance-based estimate is a conservative one, and the use of more parameters such as those from rotation and perspective projection should reduce the number of generated points further. However, our view-dependent rendering algorithm ensures that a node projects to no more than a couple of pixels and hence the only significant parameter is the distance of the node from the camera eye.

We have set a maximum threshold for the screen-space size of σ_p^i to be 4 in our case. Under our scheme for view-dependent tree traversal (see §5.2.3) larger values can occur only when the user is extremely close to the surface. For such cases one can either render using larger points or generate more points based on the Gaussian distribution parameters. Table 3.1 shows that a pseudo-random number scheme requires more samples than a scheme based on quasi-random numbers. This is natural since pseudo-random number generators exhibit greater discrepancy. Moreover, quasi-random sampling does not exhibit temporal aliasing since the quasi-random sequence does not vary on a per-frame basis.

3.2.5 Statistical Geometry Modeling in the Unified-Attribute Space

The PCA analysis in the unified-attribute space gives us a better representation of a point cloud than doing separate PCA in the individual attribute spaces since the correlation between the attributes is preserved in this case. However, unified-attribute-space PCA could still be a coarse approximation if the distribution in any of the individual attribute spaces is complex. Hence, we use the partitioning scheme of §3.2.1 to derive a hierarchical PCA distribution in the unified-attribute space. Alternately, instead of using clustering in the spatial attribute space, it is possible to partition using clustering in the unified-attribute space. However, we simply went with the former approach since it creates a

(a) (b)

Figure 3.9: *Points generated from a low-resolution cut (at level 12) of the tree. Figure (a): Points generated by sampling the individual Gaussian distributions of the position, normal, and color attributes. Figure (b): Points generated by sampling the Gaussian distribution derived from the unified attribute space. There are 4096 nodes in each figure with each node generating the same number of points that it represents. Notice the better distribution of the normals near the eyes and in the hair in Figure (b).*

more balanced tree (especially at lower resolutions).

We sample the Gaussian distribution in the unified-attribute space just like we sampled the individual attribute distributions previously. This requires us to generate Gaussian random numbers in a 8D space. We do this by sampling points uniformly on a 8D hypersphere [78] and then radially warping them according to a Gaussian distribution of unit variance. We use these Gaussian numbers and warp them according to the 8D PCA parameters of the node (mean, standard deviation, and basis frame). This distortion process is a direct generalization of the technique we used for the spatial attributes in §3.2.2. Note that the points generated this way have the proper position and color attributes. However, the normals are still in the 2D logarithmic tangent space. We convert these values to normals in 3D by using the exponential map with respect to the mean normal (μ_θ, μ_ϕ) [15].

Since this sampling is done in the unified attribute space the correlation between

the attributes is maintained and the generated points are much more closer to the underlying point cloud (see Figure 3.9). In order to determine the number of points to generate for a given node we need to determine its screen space projection area. For this we consider a 8D hyper-ellipsoid with intercepts of $\gamma\sigma$ along its respective principal axes. This is generalization of the approach we adopted for the spatial domain (see Figure 2.3). We project these intercepts to the 3D spatial domain and choose the maximum of these projections as an estimate of the radius of the sphere bounding the spatial attributes of the node. We use the screen space projection defined by this radius to index into table 3.1 for determining the number of points to generate.

3.3 Comparison and Summary

In this chapter we discussed how visual data can be modeled using context-aware samples as the basic building blocks. We saw that the differential point geometry is modeled by sampling points on the surface and computing the parameters of the differential points using eigenanalysis. We also discussed how undersampling and oversampling issues are handled. We discussed how a statistical point geometry is sampled using hierarchical partitioning. We also showed how the original data can be approximated by sampling the probability distribution derived from statistical analysis.

Differential points offer a very compact representation of the visual data. The second-order approximation of differential points did not show any noticeable visual artifacts. The representation of visual data using statistical points is more robust to noise than differential points. Also statistical points are easier to organize hierarchically.

Chapter 4

Encoding Context-Aware Samples

Context-aware samples represent the overall geometry with fewer number of samples than context-blind samples. Additionally, there is significant coherence in the actual context information of the samples and we can reduce the representational complexity even further by leveraging this coherence. In this chapter we discuss how the coherence in the context information can be exploited for compactly encoding context-aware samples. Our primary tools for this procedure are quantization and classification techniques.

4.1 Encoding Differential Point Geometry

We use coherence in the context information of the differential points using quantization. We quantize the position $\mathbf{x_p}$ of a DP $\mathbf{p}$ in 6 bytes by using 2 bytes each for the $x-$, $y-$, and $z-$coordinates. We quantize the frame into 4 bytes by quantizing the θ and ϕ angles of $\hat{\mathbf{u}}_\mathbf{p}$ and $\hat{\mathbf{v}}_\mathbf{p}$ into a byte each. We found that a 8-bit precision for quantizing the curvatures values was not enough for our test models. Hence we quantize the maximum principal curvature, $\lambda_{\mathbf{u_p}}$, with 16-bits and encode the minimum principle curvature, $\lambda_{\mathbf{v_p}}$, by encoding the ratio $\rho_\mathbf{p} = \frac{\lambda_{\mathbf{v_p}}}{\lambda_{\mathbf{u_p}}} \in [0, 1]$ with 8 bits. This amounts to quatizing a DP into 13 bytes. However, if the DPs are undersampled (ex. sampled from a triangle mesh) then the dimensions of the rectangles $\mathbf{r_p}$ of a few DPs might be different than the size automatically computed from the curvature values (see §3.1.1). If the size of the rectangle $\mathbf{r_p}$ can be computed solely from the curvature values, then a zero byte is written to the file after the first 13 bytes of the DP have been written. Otherwise, the width and the height of $\mathbf{r_p}$ are encoded in 1 byte each (the bytes being nonzero) and written after the first 13 bytes of the DP. We do not save the color for each DP, but group together DPs with the same color and write the color information once for this group.

(a) (b)

Figure 4.1: *A comparison of the rendering quality with and without encoding: (a) The bunny rendered without encoding, (b) The bunny rendered with encoding.*

4.2 Encoding Statistical Points

The PCA-based representation of a set of points is fairly compact. However, a quick look at the PCA parameters of the nodes of the hierarchy shows that there is a high coherence in the PCA parameters themselves. We use two approaches to leverage this coherence for a compact representation: classification and quantization.

4.2.1 Classification

Due to the uniform subdivision enforced by our algorithm the standard deviations σ exhibit a high level of coherence. We use a k-means clustering algorithm on the standard deviations (σ_p, σ_n, and σ_c) to derive a small number of representative variances (between 64 to 4K for each model). Figures 4.2(b) and 4.2(c) show the original values of the standard deviations σ_p and their cluster centers. We are now free to make a global lookup table of the σ cluster centers and only store the index of the best matching standard deviation with each node. This saves us significant number of bits for each node because now we use only 12 bits each for σ_p and σ_n, and 6 bits for σ_c (see Figure 4.2(a)).

4.2.2 Quantization

We use quantization to reduce the number of bits needed for the remaining attributes. To encode the frame, f_p we could quantize the quaternion coefficients corresponding to the rotation of the unit basis to f_p. This approach gives equal weight to all the three principal

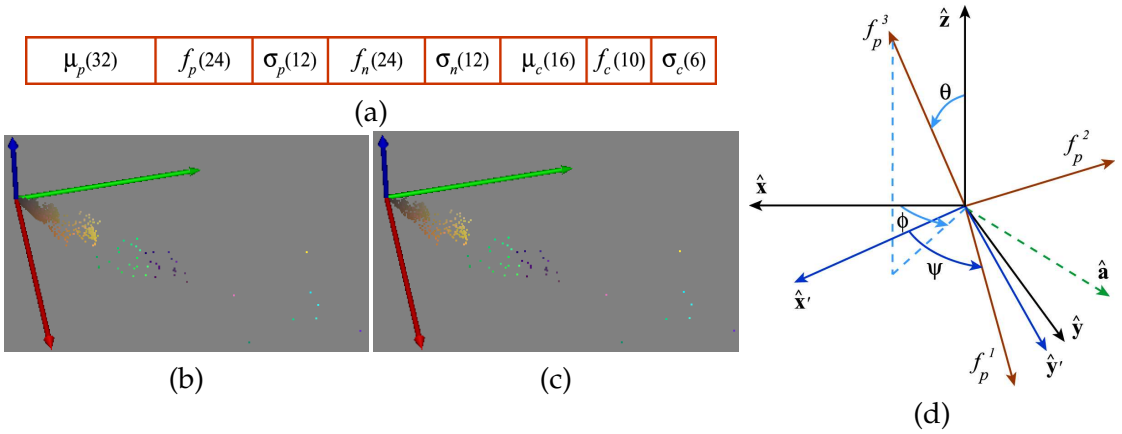


Figure 4.2: *(a): A node is quantized into 13 bytes for the spatial and normal information. Four extra bytes are used for the optional color information. The breakdown is shown in bits. (b): About 600K PCA values of σ_p for the David's Head, and (c): their 512 k-means cluster centers. (d): We quantize the frame, f_p, by quantizing its θ, ϕ, and ψ angles.*

components. However we have observed that the human eye is much more sensitive to the quantization of f_p^3, (that generally points in the direction of the local normal) than to the quantization of the other two axes. So we quantize f_p^3 separately by quantizing its θ and ϕ angles in 8 and 10 bits, respectively (see Figure 4.2(d)). To quantize the other two axes we observe that they are orthogonal in the plane normal to f_p^3. The remaining two components, f_p^1 and f_p^2, can therefore be represented by a single angle ψ. To see this, consider the rotation of the unit vectors $\hat{\mathbf{x}} = (1,0,0)$, $\hat{\mathbf{y}} = (0,1,0)$, and $\hat{\mathbf{z}} = (0,0,1)$ by an angle of θ around the axis $\hat{\mathbf{a}} = \hat{\mathbf{z}} \times f_p^3$ (see Figure 4.2(d)). If we denote the rotated vectors by $\hat{\mathbf{x}}'$, $\hat{\mathbf{y}}'$, and $\hat{\mathbf{z}}'$ respectively, then $\hat{\mathbf{z}}' = f_p^3$, while $\hat{\mathbf{x}}'$ and $\hat{\mathbf{y}}'$ reside on the plane normal to f_p^3. The angle ψ is then simply the counter-clockwise angle going from $\hat{\mathbf{x}}'$ to f_p^1. We quantize ψ by 6 bits which means that the whole frame f_p can be quantized into 24 bits.

Our method of encoding the frame f_p allows us to decode its quantized information quickly. Given the values of θ, ϕ, and ψ we can compute the frame vector f_p^3 directly as $(\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)$. To determine the other two frame vectors we first need to

(a) (b)

Figure 4.3: *Figure (a) shows view-dependent rendering on a* 512×512 *window from 191K un-encoded nodes and 824K generated points. Figure (b) shows the same rendering from encoded nodes. We encode each node to 17 bytes using quantization and classification. We noticed very little difference between rendering with encoded and unencoded data.*

compute the vector $\hat{\mathbf{x}}'$ given by [96]:

$$\begin{aligned} \hat{\mathbf{x}}' &= \cos\theta\ \hat{\mathbf{x}} + (1-\cos\theta)(\hat{\mathbf{a}}\cdot\hat{\mathbf{x}})\hat{\mathbf{a}} + \sin\theta\ (\hat{\mathbf{a}}\times\hat{\mathbf{x}}) \\ &= \begin{bmatrix} \cos\theta + f_p^3(y)\ f_p^3(y)\ (1-\cos\theta) \\ -f_p^3(x)\ f_p^3(y)\ (1-\cos\theta) \\ f_p^3(x)\ \sin\theta \end{bmatrix}, \end{aligned}$$

where $f_p^3(x)$ and $f_p^3(y)$ are the x- and y- components of f_p^3 respectively. The vector $\hat{\mathbf{y}}'$ can be computed similarly. The final frame vector f_p^1 is then given by the rotation of $\hat{\mathbf{x}}'$ and $\hat{\mathbf{y}}'$ by an angle of ψ to get :

$$\begin{aligned} f_p^1 &= \cos\psi\ \hat{\mathbf{x}}' + \sin\psi\ \hat{\mathbf{y}}' \\ f_p^2 &= \cos\psi\ \hat{\mathbf{y}}' - \sin\psi\ \hat{\mathbf{x}}' \end{aligned}$$

We speed up decoding by using a lookup table for the sine and cosine values of all the possible quantized values of the angles θ, ϕ, and ψ representing f_p. Since we use a 8-10-6 quantization of these angles, our lookup table consists of $(2^8 + 2^{10} + 2^6) \times 2 = 2688$ floating point numbers which can easily fit into present-day caches.

Quantizing the remaining information is straightforward. We quantize f_n similarly with 24 bits. We quantize f_c in 10 bits using a 4-3-3 quantization of its (θ, ϕ, ψ) values. We encode μ_p in 32 bits using a 10-11-11 quantization, where the dimension of minimum width uses a 10 bit quantization. The value of μ_c is encoded in 16 bits using a 5-6-5 quantization of its red, green, and blue values [99]. Overall, each node can be represented with 13 bytes of spatial and normal information with 4 extra bytes required for color. A complete single-precision floating-point representation would have required 96 bytes. Figure 4.3 illustrates the effectiveness of our encoding algorithm visually (the details of the view-dependent rendering algorithm will be discussed in §5.2.3).

4.3 Summary

In this chapter we have shown how the context-aware samples can be encoded. We have shown that quantization can be used to encode the parameters of the context information and classification can yield significant reduction in the representational complexity. We encode both CAS primitives with just 13 bytes for surface position and normals. We note that differential points can be represented with even fewer bytes by using classification. Our encoding had negligible impact on the quality of the rendering. Although context-aware samples need more bytes per sample when compared to simple point primitives, they have far less overall memory consumption since they are fewer in number. We present a quantitative comparison of the memory consumption of CAS primitives in chapter 6 (see Tables 6.1 and 6.2).

Chapter 5

Transmission and Rendering

Context-aware samples have several elegant properties which can be used for efficient transmission and rendering. Since they are completely independent of each other, they can be used for efficient streaming. Moreover, since they are similar to procedural or parametric representations, they shift the load of rendering from memory access to computation. This effectively addresses the growing disparities between the speeds of memory and computation (the memory wall problem) and gives us a significant rendering speedup. In the rest of this chapter we discuss the transmission and rendering of context-aware samples. For us transmission encompasses both the bandwidth for the system bus as well as the network bandwidth for remote rendering.

5.1 Differential Point Rendering

Differential points are completely independent of each other, therefore they can be simply transmitted in an order-independent manner. We render a DP by rasterizing the regional surface, $\mathbf{S_p}$, using the fragment shaders. While the differential information in a DP can be extrapolated to define a continuous spatial neighborhood, $\mathbf{S_p}$, current graphics hardware do not support such a rendering primitive. We note that the main functionality of the spatial distribution is that it derives the normal distribution around the differential point. However, it is not necessary for the rendering algorithm to use an accurate spatial distribution given the relatively small neighborhoods of extrapolation. So we use the rectangle $\mathbf{r_p}$ as an approximation to $\mathbf{S_p}$ when rasterizing $\mathbf{p}$. Since the shading artifacts are more readily discernible to the human eye the screen-space normal distribution around $\mathbf{p}$ has to mimic the normal variation around $\mathbf{p}$ on the original surface. This is done by projecting the normal distribution $\mathbf{N_p}(u, v)$ onto $\mathbf{r_p}$ and rasterizing $\mathbf{r_p}$ with a normal map of this distribution. Normal mapping is not neccesary when rendering with the more recent

graphics cards since the normals can be directly computed at every pixel using our formulations. In the following subsections we will discuss how we precompute the normal distribution, $\mathbf{N_p}(u, v)$, and then detail our run-time per-pixel shading algorithm.

5.1.1 Normal Distribution

Consider the projection of $\mathbf{N_p}(u, v)$ onto $\tau_\mathbf{p}$ using the projection $\mathcal{P}_\mathbf{p}$ discussed in §3.1.3. The resulting (un-normalized) normal distribution, $\mathbf{n_p}(u, v)$, on the tangent plane can be expressed using equation (2.6) as:

$$\mathbf{n_p}(u, v) \approx \mathbf{N_p}(0, 0) + s(u, v)\, \mathbf{dN_p}(\hat{\mathbf{t}}(u, v)) \tag{5.1}$$

It can be shown that the normal distribution, $\mathbf{n_p}(u, v)$, can be expressed in the local coordinate system $(\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z)$ of $(\hat{\mathbf{u}}_\mathbf{p}, \hat{\mathbf{v}}_\mathbf{p}, \hat{\mathbf{n}}_\mathbf{p})$ as [59]:

$$\mathbf{n_p}(u, v) \approx \hat{\mathbf{e}}_z - \left[\frac{(\lambda_{\mathbf{u_p}}\, u\, \hat{\mathbf{e}}_x + \lambda_{\mathbf{v_p}}\, v\, \hat{\mathbf{e}}_y)}{(\lambda_{\mathbf{u_p}} u^2 + \lambda_{\mathbf{v_p}} v^2)/\sqrt{u^2 + v^2}} \arcsin\left(\frac{\lambda_{\mathbf{u_p}} u^2 + \lambda_{\mathbf{v_p}} v^2}{\sqrt{u^2 + v^2}} \right) \right] \tag{5.2}$$

where $\hat{\mathbf{e}}_x = (1, 0, 0)$, $\hat{\mathbf{e}}_y = (0, 1, 0)$, and $\hat{\mathbf{e}}_z = (0, 0, 1)$ are the canonical basis in $\Re^3$. This expression has the nice feature that the normal distribution is independent of $\hat{\mathbf{u}}_\mathbf{p}$, $\hat{\mathbf{v}}_\mathbf{p}$, and $\mathbf{N_p}(0, 0)$ when specified in the local coordinate frame.

To shade a DP on a per-pixel basis we would want the normal distribution to be available at the screen space. This can be done on the GPU using its support for normal mapping. A normal map is a texture map where the red, green, and blue channels of the texture stand for the $x-$, $y-$, and $z-$ components of the normal. After the normal map is mapped to the geometry, the normals can be made available at each pixel that the geometry projects to. It is expensive to compute a normal map at run-time for each combination of $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$. Hence we pre-compute normal maps in the local coordinate frame for different values of the principal curvatures $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$. At run time we normal map the rectangle $\mathbf{r_p}$ with the closest resembling normal map amongst these pre-computed normal maps. A drawback to this scheme is that since $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$ are unbounded quantities it is impossible to compute all possible normal maps. To get around this problem,

we introduce a new term, $\rho_{\mathbf{p}} = \frac{\lambda_{\mathbf{v_p}}}{\lambda_{\mathbf{u_p}}}$, and note that $-1 \leq \rho_{\mathbf{p}} \leq 1$ because $|\lambda_{\mathbf{u_p}}| \geq |\lambda_{\mathbf{v_p}}|$. We then rewrite the local normal distribution of equation(5.2) using $\rho_{\mathbf{p}}$ as:

$$\mathbf{n_p}(u,v) \approx \hat{\mathbf{e}}_z - (u\,\hat{\mathbf{e}}_x + \rho_{\mathbf{p}}\,v\,\hat{\mathbf{e}}_y)\frac{\arcsin(\lambda_{\mathbf{u_p}}\psi_{\mathbf{p}}(u,v))}{\psi_{\mathbf{p}}(u,v)} \tag{5.3}$$

where $\psi_{\mathbf{p}}(u,v) = (u^2 + \rho_{\mathbf{p}}v^2)/\sqrt{u^2+v^2}$. Now consider a normal distribution for a differential point $\mathbf{m}$ whose $\lambda_{\mathbf{u_m}} = 1$:

$$\mathbf{n_m}(u,v) \approx \hat{\mathbf{e}}_z - (u\,\hat{\mathbf{e}}_x + \rho_{\mathbf{m}}\,v\,\hat{\mathbf{e}}_y)\frac{\arcsin(\psi_{\mathbf{m}}(u,v))}{\psi_{\mathbf{m}}(u,v)}$$

The only external parameter to $\mathbf{n_m}(u,v)$ is $\rho_{\mathbf{m}}$. Since $\rho_{\mathbf{m}}$ is bounded, we pre-compute a set, $\mathcal{M}$, of normal distributions for discrete values of ρ and store them as normal maps. Later, at render time, we normal map the rectangle $\mathbf{r_m}$ by the normal map whose ρ value is closest to $\rho_{\mathbf{m}}$. To normal map a general differential point $\mathbf{p}$ using the same set of normal maps, $\mathcal{M}$, we use the following lemma:

Lemma 1 *When expressed in their respective local coordinate frames,* $\mathbf{n_p}(u,v) \approx \mathbf{n_m}(\lambda_{\mathbf{u_p}}u, \lambda_{\mathbf{u_p}}v)$ *where* $\mathbf{m}$ *is any DP with* $\lambda_{\mathbf{u_m}} = 1$ *and* $\rho_{\mathbf{m}} = \rho_{\mathbf{p}}$.

Proof: First, we make an observation that $\lambda_{\mathbf{u_p}}\psi_{\mathbf{p}}(u,v) = \psi_{\mathbf{p}}(\lambda_{\mathbf{u_p}}u, \lambda_{\mathbf{u_p}}v)$. Using this observation, the tangent plane normal distribution at $\mathbf{p}$ (equation (5.3)) can be re-written as:

$$\begin{aligned}
\mathbf{n_p}(u,v) &\approx \hat{\mathbf{e}}_z - \left[((\lambda_{\mathbf{u_p}}u)\hat{\mathbf{e}}_x + \rho_{\mathbf{p}}(\lambda_{\mathbf{u_p}}v)\hat{\mathbf{e}}_y)\,\frac{\arcsin(\psi_{\mathbf{p}}(\lambda_{\mathbf{u_p}}u, \lambda_{\mathbf{u_p}}v))}{\psi_{\mathbf{p}}(\lambda_{\mathbf{u_p}}u, \lambda_{\mathbf{u_p}}v)}\right] \\
&= \hat{\mathbf{e}}_z - \left[((\lambda_{\mathbf{u_p}}u)\hat{\mathbf{e}}_x + \rho_{\mathbf{m}}(\lambda_{\mathbf{u_p}}v)\hat{\mathbf{e}}_y)\,\frac{\arcsin(\psi_{\mathbf{m}}(\lambda_{\mathbf{u_p}}u, \lambda_{\mathbf{u_p}}v))}{\psi_{\mathbf{m}}(\lambda_{\mathbf{u_p}}u, \lambda_{\mathbf{u_p}}v)}\right] \\
&\approx \mathbf{n_m}(\lambda_{\mathbf{u_p}}u, \lambda_{\mathbf{u_p}}v)
\end{aligned}$$

□

The above lemma shows that a screen-space normal distribution for any general DP $\mathbf{p}$ can be obtained by normal mapping the rectangle $\mathbf{r_p}$ with the best matching normal map $\mathbf{n_m}(\cdot,\cdot)$ with a scaling factor of $\lambda_{\mathbf{u_p}}$.

5.1.2 Shading

We discussed the mechanism for delivering the normal at pixel that a DP projects to. In this section we discuss how this information be used to compute the lighted color of each pixel. For specular shading, apart from the local normal distribution, we also need a local half-vector distribution. For this we use the cube-vector-mapping functionality [67] offered in the nVIDIA GeForce series of GPUs which allows us to specify un-normalized vectors at each vertex of a polygon and obtain linearly-interpolated and normalized versions of these on a per-pixel basis. We use the cube-vector map to specify a un-normalized half vector at each vertex of $\mathbf{r_p}$ which delivers a normalized half vector at each pixel that $\mathbf{r_p}$ occupies. Per-pixel specular shading is achieved by using the per-pixel normal (from the normal map) and half vector (from the cube-vector map) for illumination computations in the register combiners. A similar technique is used for diffuse shading.

Let $\hat{\mathbf{h}}_\mathbf{p}$ denote the (normalized) half (halfway) vector at the point position $\mathbf{x_p}$ and let $\mathbf{H_p}(u,v)$ denote the (un-normalized) half vector at a point $\mathbf{X_p}(u,v)$ on the surface $\mathbf{S_p}$ with $\mathbf{H_p}(0,0) = \hat{\mathbf{h}}_\mathbf{p}$. Let $\mathbf{h_p}(u,v)$ be the (un-normalized) half-vector distribution on the tangent plane $\tau_\mathbf{p}$ obtained as a result of applying the projection $\mathcal{P}_\mathbf{p}$ on $\mathbf{H_p}(u,v)$. It can then be shown that [59]:

$$\mathbf{h_p}(u,v) \approx \mathbf{H_p}(0,0) + u\frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} + v\frac{\partial}{\partial v}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} \tag{5.4}$$

where the partial differential can be shown to be of the form [59]:

$$\frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} = \frac{((\hat{\mathbf{l}}_\mathbf{p}\cdot\hat{\mathbf{e}}_x)\hat{\mathbf{l}}_\mathbf{p} - \hat{\mathbf{e}}_x)}{\|\mathbf{a}-\mathbf{x_p}\|} + \frac{((\hat{\mathbf{w}}_\mathbf{p}\cdot\hat{\mathbf{e}}_x)\hat{\mathbf{w}}_\mathbf{p} - \hat{\mathbf{e}}_x)}{\|\mathbf{b}-\mathbf{x_p}\|} \tag{5.5}$$

where $\mathbf{a}$ is the position of the light, $\mathbf{b}$ is the position of the eye, and $\hat{\mathbf{l}}_\mathbf{p} = \frac{\mathbf{a}-\mathbf{x_p}}{\|\mathbf{a}-\mathbf{x_p}\|}$ and $\hat{\mathbf{w}}_\mathbf{p} = \frac{\mathbf{b}-\mathbf{x_p}}{\|\mathbf{b}-\mathbf{x_p}\|}$ are the respective normalized light and view vectors at $\mathbf{x_p}$. The subtraction and the dot products in equation (5.5) are simple operations and can be done fast. However, the square root and the division operations are expensive. Both of these operations are combined by the fast inverse-square-root approximation [115] and in practice, we have

Display( )

(Let $\mathcal{M}$ be the set of normal maps computed for quantized values of ρ. It is computed and loaded into texture memory at the program start time)
Clear the depth buffer and the color buffers
Configure the register combiners for diffuse shading
$\forall$ DP $\mathbf{p}$
    $\mathcal{M}_{\mathbf{p}}$ = normal map $\in \mathcal{M}$ whose ρ is closest to $\rho_{\mathbf{p}}$
    Map $\mathcal{M}_{\mathbf{p}}$ onto $\mathbf{r_p}$
    Compute the light vector, $\mathbf{l_p}(\cdot,\cdot)$, at the vertices of $\mathbf{r_p}$
    Use the light vectors to map a cube vector map onto $\mathbf{r_p}$
    Render $\mathbf{r_p}$
Clear the color buffer after loading it into the accumulation buffer
Clear the depth buffer
Configure the register combiners for specular shading
$\forall$ DP $\mathbf{p}$
    $\mathcal{M}_{\mathbf{p}}$ = normal map $\in \mathcal{M}$ whose ρ is closest to $\rho_{\mathbf{p}}$
    Map $\mathcal{M}_{\mathbf{p}}$ onto $\mathbf{r_p}$ (The details from the last pass can be cached if desired)
    Compute the half vector, $\mathbf{h_p}(\cdot,\cdot)$, at the vertices of $\mathbf{r_p}$
    Use the half vectors to map a cube-vector map onto $\mathbf{r_p}$
    Render $\mathbf{r_p}$
Add the accumulation buffer to the color buffer
Swap the front and the back color buffers

Figure 5.1: *Differential Point Rendering Algorithm*

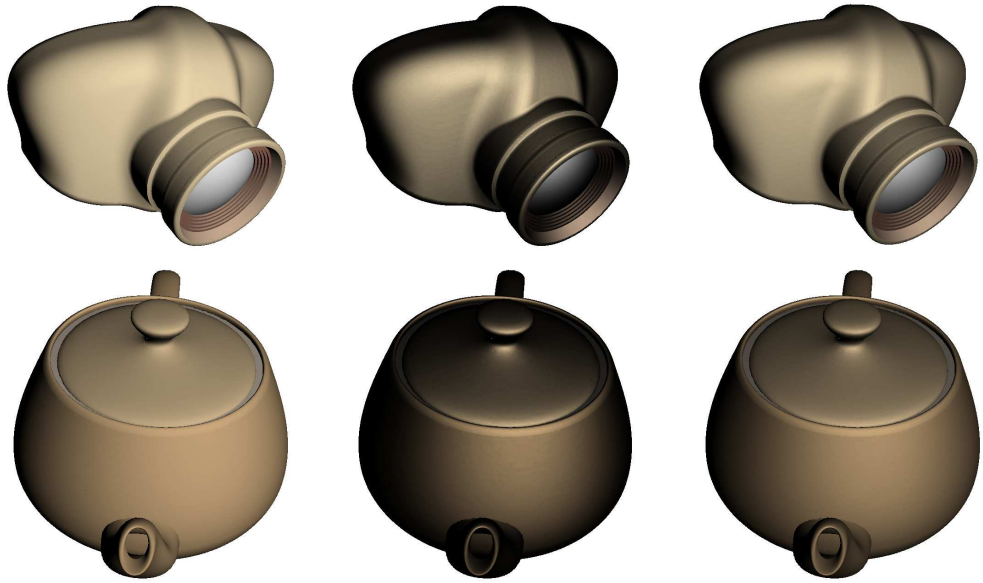
found that this approximation causes no compromise in visual quality.

The light-vector distribution on $\tau_{\mathbf{p}}$ can be derived similarly to be:

$$\mathbf{l_p}(u,v) \approx \hat{\mathbf{l}}_{\mathbf{p}} - u\hat{\mathbf{e}}_x - v\hat{\mathbf{e}}_y$$

The normal ($\mathbf{n_p}(u,v)$), half-vector ($\mathbf{h_p}(u,v)$), and the light-vector ($\mathbf{l_p}(u,v)$) distribution around $\mathbf{p}$ can be delivered on a per-pixel basis using the normal-mapping and cube-mapping mechanisms discussed above. We use them to determine the lighted color of each pixel by computing the term $\alpha\ \mathbf{n_p}(u,v) \cdot \mathbf{l_p}(u,v) + \beta\ \mathbf{n_p}(u,v) \cdot \mathbf{h_p}(u,v)$, where $\alpha, \beta \in [0,1]$. The overall rendering algorithm is given in Figure 5.1.

In summary, in order to color the pixels that a DP projects to, we need two operations: (1) computing the relevant vectors (coordinates) for texture mapping (CPU-end



(a) Diffuse Illumination (b) Specular Illumination (c) Diffuse and Specular Illumination

Figure 5.2: *Examples illustrating differential point rendering under various illumination schemes.*

computation) and (2) per-pixel shading (GPU-end computation). For the first part we map the rectangle $\mathbf{r_p}$ by two textures: the normal map and the half vector (or light vector) map. Normal-mapping involves choosing the best approximation to the normal distribution from the set of pre-computed normal maps $\mathcal{M}$ and computing the normal-map coordinates (u, v) for the vertices of $\mathbf{r_p}$. Half-vector mapping involves computing the unnormalized half vectors at the vertices of $\mathbf{r_p}$ using equation (5.4) and using them as the texture coordinates of the cube vector map that is mapped onto $\mathbf{r_p}$. The cube-vector mapping hardware delivers a per-pixel (normalized) half vector obtained as result of a linear interpolation between the half vectors specified at the vertices of $\mathbf{r_p}$. Per-pixel shading is achieved at the fragment shaders (previously known as register combiners) using the (per-pixel) normal and half vectors [67]. If both diffuse and specular shading are desired then shading is done in two passes with the accumulation buffer being used to buffer the results of the first pass. We use a two-pass scheme because nVIDIA GeForce2 allows only two textures at the register combiners. If three textures are accessible at the combiners (as in GeForce3 or higher) then both the diffuse and specular illumination can be done in one pass. In presence of multiple light sources, we do a separate rendering pass for each

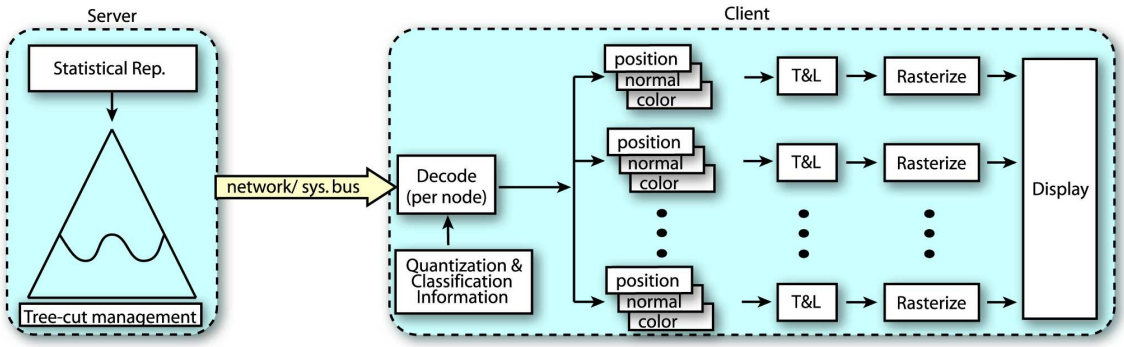


Figure 5.3: *Client-Server rendering: The server selects the level of detail to be used for rendering in a view-dependent manner. The nodes of the appropriate level of detail are transmitted to the client, which is either the graphics card or a remote rendering device. The client renders each node by generating points and their attributes from the statistical information of the node.*

light source, using the accumulation buffer for intermediate results.

5.2 Statistical Point Generation

Statistical points, like differential points, are processing-order independent. In addition, statistical samples have the advantage of a hierarchical structure which can be leveraged in several ways for transmission and rendering. In this section we discuss how the robust and simple representation of statistical points translates to efficient transmission and rendering.

5.2.1 Client-Server Model

We use a client-server model for transmission and rendering of statistical points. This model applies to both transmission on the system bus and transmission over the network. The underlying idea behind our client-server model is that the server sends only the PCA parameters to the client and the client renders that PCA node by sampling the requisite number of points. This is illustrated in Figure 5.3. A time-line illustration of our client-server architecture is shown in Figure 5.4.

We deal with three kinds of rendering devices: (1) GPU, (2) remote computer, and (3) PDA. The GPU represents a single-system computer where the CPU sends the geometry information to the GPU for rendering. This is consistent with the architecture of graphics interfaces such as OpenGL and DirectX that allow the CPU to treat the GPU as a

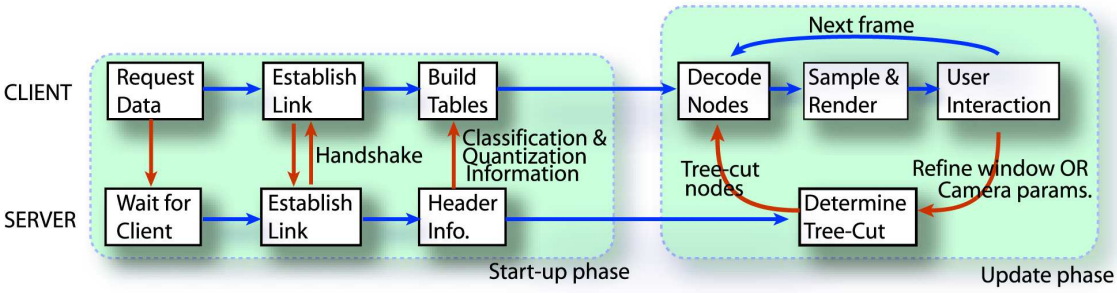


Figure 5.4: *A time-line illustration of our client-server architecture. A blue arrow represents a change of state at the server or the client, while a red arrow represents a flow of information between the server and the client. Depending on the bandwidth of the communication channel, this architecture can be used for a per-frame* view-dependent *rendering or a client-feedback-based* on-demand *rendering.*

client accessed through device drivers. We make no distinction between GPU and other client rendering devices since the bottleneck is generally the communication bandwidth that we wish to reduce.

Transmission to the client involves two phases: the initial *startup* phase and the per-frame *update* phase. This is illustrated in Figure 5.4. In the startup phase the client receives global information about the geometry. This constitutes the classification and quantization information discussed in §4.2. The classification information consists of the classes of the standard deviations σ_p, σ_n, and σ_c. The quantization information consists of the bit distribution for μ_p, μ_c, f_p, f_n, and f_c. This information sets up the client to decode the PCA nodes as they arrive.

We have experimented with two kinds of client-server rendering frameworks – *on-demand* and *view-dependent* rendering. The on-demand rendering is more suitable for applications that involve less synchronous communication on lower-bandwidth communication channels such as the Wi-Fi 802.11x and cell phone networks. The view-dependent rendering requires a greater synchronous, per-frame communication with the server and is better suited for time-critical applications on higher bandwidth communication channels such as the system bus and dedicated fiber-optic networks.

5.2.2 On-demand Transmission

In *on-demand rendering* the user selects a subset of the model using a refinement window. The client requests the server to update the nodes in that window. The server sends back

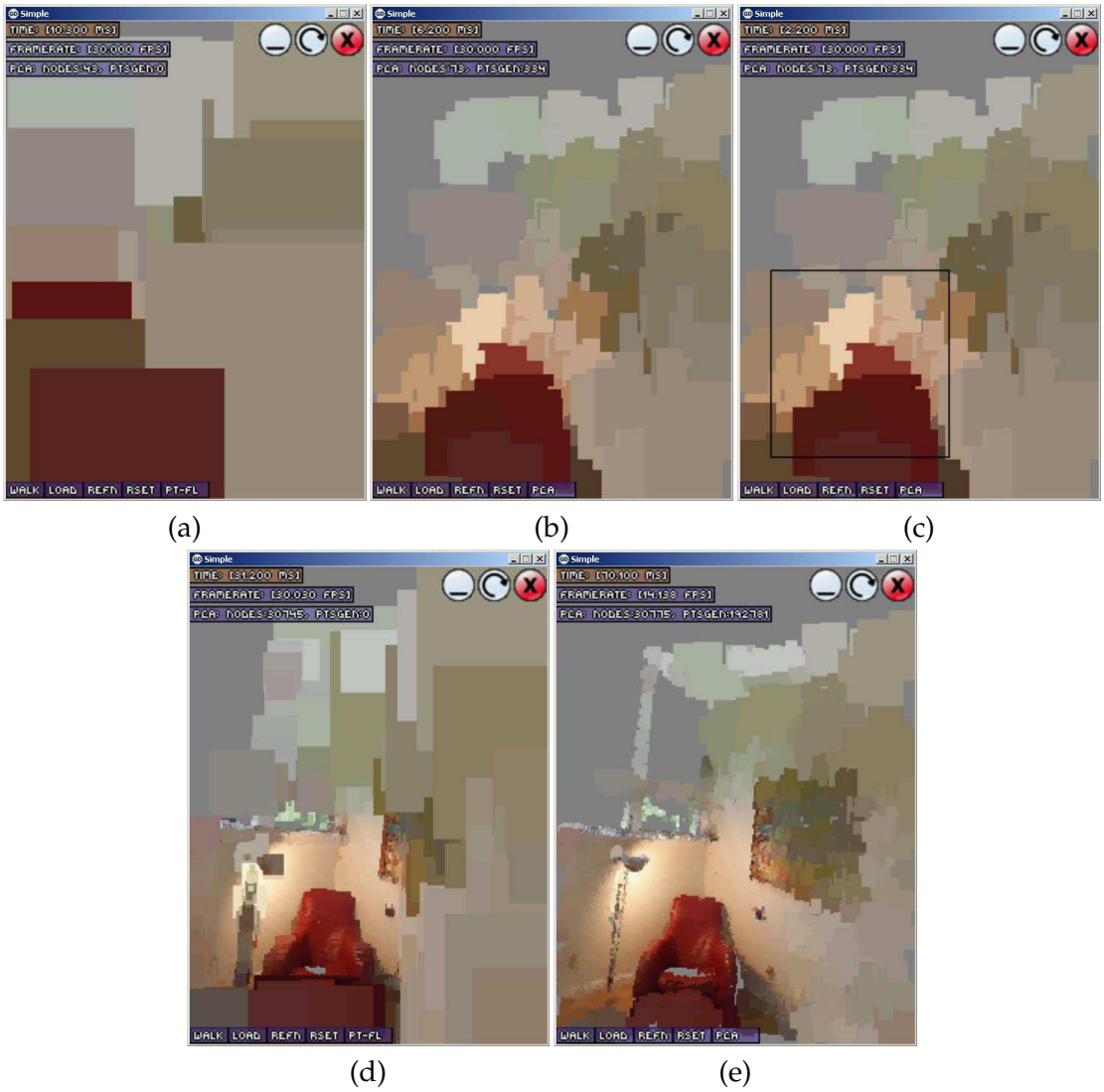


Figure 5.5: *On-demand rendering: We show the rendering of PCA nodes on a remote PC with (a) square splats and (b) with quasi-random sampling. The client selects a refinement window in Figure (c). Figures (d) and (e) are the rendering of the refined nodes with square splats and quasi-random sampling, respectively. The figures show that quasi-random sampling conveys more information for the same number of nodes. However, the software rendering at the client was twice as slow.*

the encoded PCA information of the refined nodes (see Figures 5.5 and 5.6). Here the server can either maintain a mirror state of the client and its level-of-detail information or the client can send its past transactions so that the server can determine the current level of detail. In the first case the client only has to send the parameters of its camera and the refinement window. This leads to less flow of information between the two, but comes at the cost of the server memory. In the second case the bandwidth used by the client is still small, although there is some computational load on the server. We have used the first case in our experiments although the latter case may be more suitable when scaling to a large number of clients. We have tested our on-demand framework on a variety of communication channels such as Wi-Fi 802.11b, Ethernet LAN, the Internet, and USB. Our client rendering devices for these experiments were a remote PC and a PDA.

5.2.3 View-dependent Transmission

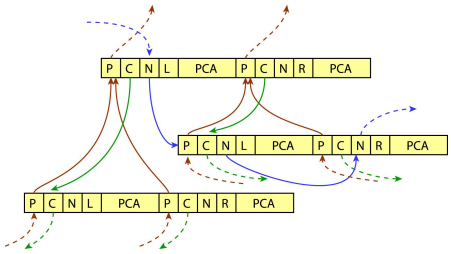
In *view-dependent rendering* we update the level of detail at each frame depending on the proximity of the object to the eye of the virtual camera. An appropriate level of detail in the hierarchy is maintained as a level cut across the hierarchy tree or a *tree cut*. Thus, in regions where higher detail is desired the tree cut is close to the leaves of the hierarchy and in regions of low detail the cut is closer to the root. Before we discuss the details of our view-dependent level-of-detail determination we will discuss our hierarchy tree data-structure which is crucial from an implementation stand point. Our hierarchy tree data structure is similar to B-Trees (see Figure 5.7(a)). In each node we store a pointer to the parent, a pointer to the next node in the tree-cut, and a pointer to its left child. We do not need to store a pointer to both the children since siblings are stored in consecutive memory locations – hence the right child is only a pointer increment away from the left child. We also store the encoded PCA attributes(13 or 17 bytes) at a node and an extra byte which is set to 1 iff the child is a right child. In all we use between 26 to 30 bytes for each node. The compact size of the node leads to a nice caching performance.

We maintain the tree cut by maintaining a pointer to the first node of the tree cut and a *next* pointer in each subsequent node. The server initially sets the tree-cut to be

(a) (b) (c)

(d) (e)

Figure 5.6: *On-demand rendering: These figures show the same sequence of operations as in Figure 5.5 on a PDA client.*



(a)

```
Server( )
1.  For every node, n, in the tree-cut
2.    Decode( n )
3.    If ( CanCull( n ) )
4.      Merge( n )
5.      continue
6.    p ← NumPointsToGenerate( n )
7.    If ( NeedSplit( n, p ) )
8.      n ← Split( n )
9.      Decode( n )
10.     p ← NumPointsToGenerate( n )
11.   SendToClient( n, p )
12.   If ( CanMerge( p ) )
13.     Merge( n )
```

(b)

Figure 5.7: *View-dependent rendering data structure and algorithm. The tree data structure has following elements – P: Parent pointer, C: Child pointer, N: Next tree-cut pointer, L/R: Left/Right sibling, PCA: encoded PCA parameters.*

at half the maximum level of the hierarchy. Then, at each frame the server traverses the cut and adjusts it in a view-dependent fashion. The adjustments include checking for view-frustum culling and back-face culling as well as the use of screen-space projection area.

We implement view-frustum culling by approximating a node by a sphere of radius $\gamma\sigma_p^1$ (as in §3.2.4). We use a normal-cone-based back-face culling test [70] with the radius of the cone being $\gamma\sigma_n^1$. If the node can be culled, the server merges the node and its sibling to its parent if: (1) the node is a right child, (2) the previous node in the cut is its (left) sibling, and (3) the previous node was also culled. Pseudocode's *Merge*( ) function (Figure 5.7(b)) implements this. If the node is not culled, the server estimates the screen-space area of the node and looks up the number of points to render from the table 3.1. If the screen-space area of the node is above a maximum threshold (set to 2 in all our tests) then the server splits the node. The split node is replaced in the tree-cut by its children. The server merges the node if its screen-space area is below a threshold (set to 1 in all our cases) and moves on to the next node in the tree-cut. The server sends the encoded PCA attributes of the unculled nodes to the client-rendering device, such as the GPU, with information on the number of points to be generated. The client renders the PCA nodes

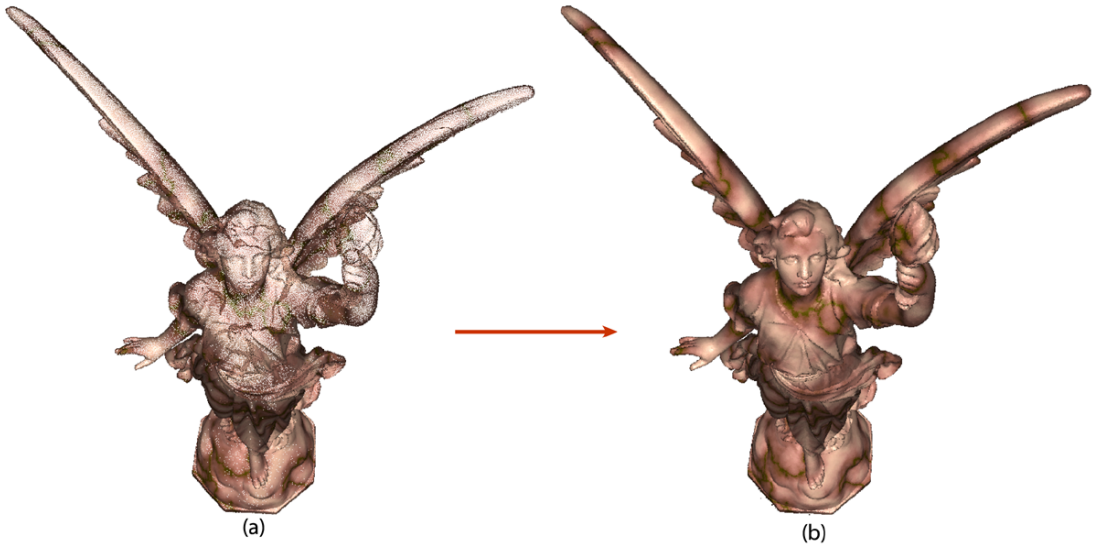


Figure 5.8: *Figure (a) shows the means of the nodes of the tree-cut during view dependent rendering. Figure (b) shows the rendering of the model using quasi-random sampling at the GPU.*

by generating the points and their attributes. If the client is a remote PC or a PDA then it can generate the points directly on the CPU. If the client is the GPU then we can generate points on it directly [61]. Figure 5.8 illustrates the view of both the server and the client during view-dependent rendering.

5.2.4 Anti-aliased Rendering

The PCA nodes are rendered as the points that are generated from their PCA attributes. While rendering these generated points we have to deal with two issues: (1) temporal aliasing, and (2) spatial aliasing. The temporal aliasing artifacts arise for pseudo-random sampling where new points are generated for every frame. Our approach of using quasi-random sampling gets rid of temporal aliasing since the generated points are from the same set for every frame. Spatial screen anti-aliasing can be effectively done using the hardware support for anti-aliasing (see Figures 6.4(b) and 6.4(c)). We have used the 8× `Quincunx` multisampling feature of the NVIDIA GPUs which comes with a very small overhead cost on rendering. Also, rendering from encoded data did not show any noticeable artifacts (see Figure 4.3).

5.3 Comparison and Summary

In this chapter we discussed how the nice features of context-aware samples translates to efficient transmission and rendering. We saw that differential points can be rendered by rasterizing their rectangles and computing the color at each pixel they occupy by using the fragment shaders. We also saw that statistical point geometry can be rendered by transmitting the nodes of a cut of the tree to the client which then renders the node by generating points. We saw that statistical point geometry is suitable for both on-demand and view-dependent transmission and that they support a variety of clients such as remote PC, PDA, or a graphics card. In the next chapter we will present the results of our experiments and compare it to previous work.

Chapter 6

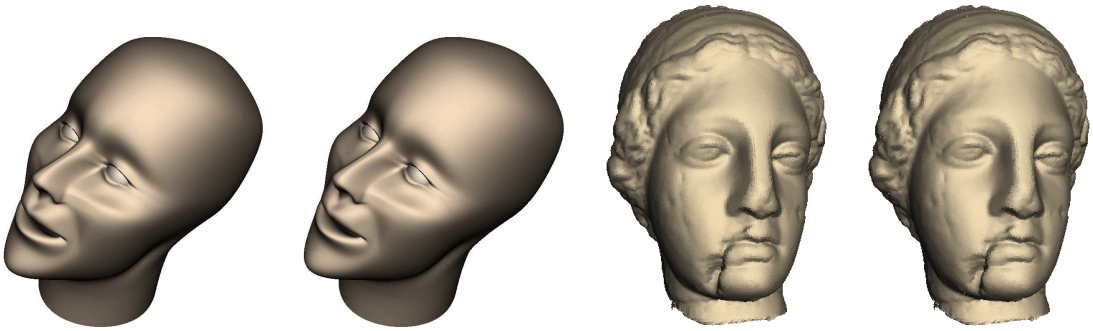
Results and Applications

In the previous chapters we have discussed the construction, representation, transmission, and rendering of context-aware samples. In this chapter we will discuss the main implementation details of these stages. We will also highlight the advantages of context-aware samples by comparing them with alternate approaches and quantify their relative benefits. There are two prevailing approaches in the research community for representing the geometry using independent samples. While context-aware samples subscribe to the approach of having smart samples, the alternate approach is to have context-blind samples which do not encode much local context information in each sample but represent the overall geometry by populating more samples for a given surface area. The best-known technique in the second category is splatting. In this case the samples are simply points with a normal and a tangential disk and they are rendered by blending the tangential disks in screen space. We compare both differential points and statistical points with splatting.

6.1 Differential Points

We have implemented differential points on a PC with a 866MHz Pentium 3 processor with 512MB RDRAM and a nVIDIA GeForce2 Graphics card with 32MB of DDR RAM. We did all our rendering-related tests on a 800×600 window. We used 256 normal maps ($|\mathcal{M}| = 256$) corresponding to uniformly sampled values of ρ and we built a linear mipmap on each of these with the highest resolution being 32×32. The resolution of the cube-vector map was 512×512×6.

We have tested differential points on five models: the Utah teapot, a human head model, a camera prototype (all NURBS models), the Stanford bunny, and the Cyberware venus model (triangle mesh models). In case of a NURBS surface the component



(a) (b) (c) (d)

Figure 6.1: *Rendering quality with and without simplification. Head Model ($\epsilon = 0.012$, $\delta = 2.0$) (a) Without simplification, (b) With simplification. Venus Model ($\epsilon = 10^{-6}$, $\delta = 0.05$) (c) Without simplification, (d) With simplification*

.

patches are sampled uniformly in the parametric domain and simplified independent of each other. The main parameter of the sampling process is the ϵ parameter. A smaller ϵ requires a higher sampling frequency. The main role of δ is in areas where curvature changes fast. In such surfaces, δ ensures that the rectangles from the low curvature region do not block the nearby rectangles in the higher curvature regions. The δ term also ensures that the rectangles do not overrun the boundary significantly. We use a simple binary heap for heap operations in the simplification process. The main functional bottleneck in the pre-processing stage is the test for enclosure in the simplification process. Since every DP popped from the heap is tested for enclosure, the number of enclosure tests is equal to the number of sampled DPs. Irrespective of the amount of super-sampling of a model, simplification yielded similar results on all attempts that shared the same error metrics (ϵ and δ). The effectiveness of simplification is summarized in table 6.1 and is illustrated in Figure 6.1. While simplification does not cause any loss of visual quality, it can lead to an order-of-magnitude speed-up in rendering and can save substantial storage space. While simplification reduces the number of primitives, quantization has the orthogonal influence of reducing the storage space for each primitive. We have found that the quantization of a DP to 13 bytes does not lead to any drop in visual quality (see Figure (4.1)).

The results reported in Table 6.1 are with dynamic illumination (the light and half vectors are computed for each DP in each frame). Both the specular and diffuse shading

Without Simplification	*Teapot*	*Camera*	*Head*	*Bunny*	*Venus*
Number of Points	156,800	216,712	376,400	34,834	134,359
Disk Space w/o encoding (in MB)	9.19	12.69	22.06	1.77	6.82
Disk Space w/ encoding (in MB)	1.99	2.75	4.77	0.51	1.99
Pre-processing Time (in seconds)	22.5	15.25	22.2	1.2	3.25
Frames per second (Diffuse)	2.13	1.59	0.89	9.09	2.44
Frames per second (Specular)	2.04	1.52	0.88	9.05	2.38
With Simplification	*Teapot*	*Camera*	*Head*	*Bunny*	*Venus*
Number of Points	25,713	46,077	64,042	34,350	92,608
Disk Space w/o encoding (in MB)	1.51	2.70	3.75	1.75	4.64
Disk Space w/ encoding (in MB)	0.32	0.59	0.82	0.50	1.34
Pre-processing Time (in seconds)	146.5	178.17	485.5	7.15	76.92
Frames per second (Diffuse)	12.51	6.89	5.26	9.11	3.57
Frames per second (Specular)	11.76	6.67	5.13	9.09	3.45

Table 6.1: *Summary of results: The teapot, camera, and the head are derived from NURBS and the Stanford bunny and the Cyberware venus are derived from a triangle mesh*

are done at the hardware level. However, nVIDIA GeForce2 does not support a hardware implementation for the accumulation buffer. Instead, the accumulation buffer is implemented in software by the OpenGL drivers. So the case with both diffuse and specular illumination can be slow. However this is not an issue for the modern GPUs where fragment shaders are much more capable. We could render about about $330,000$ DPs per second with diffuse illumination. Both the diffuse and specular illumination passes take around the same time. The main bottleneck in rendering is the bus bandwidth and the pixel-fill rate. This can be seen by noting that specular and diffuse illumination give around the same frame rates even though the cost of computing the half vectors is higher than the cost of computing the light vectors and that the specular illumination pass has more computation per-pixel than the diffuse illumination pass.

The context information of differential points gives them greater expressiveness than context-blind samples. However, context-blind samples have the advantage that their rendering-related computations are much simpler. We compare differential point rendering with splatting of context-blind samples. We construct the context-blind samples simply by replacing the contextual information of DPs with a bounding ball. We consider the following screen-space splatting primitives for context-blind samples:

1. **Square Primitive**: They are squares parallel to the view plane with a width equal to

the radius of the bounding sphere [99]. They are rendered with Z-buffering enabled but without any blending.

2. **Rectangle Primitive**: Consider a disc on the tangent plane of the point, with a radius equal to the radius of the bounding ball. Also consider a plane parallel to the view plane and located at the position of the point. An orthogonal projection of the disc on this plane results in an ellipse. The rectangle primitive is obtained by fitting a rectangle around the ellipse with the sides of the rectangle being parallel to the principal axes of the ellipse [89]. The rectangle primitives are rendered with Z-buffering but without any blending.

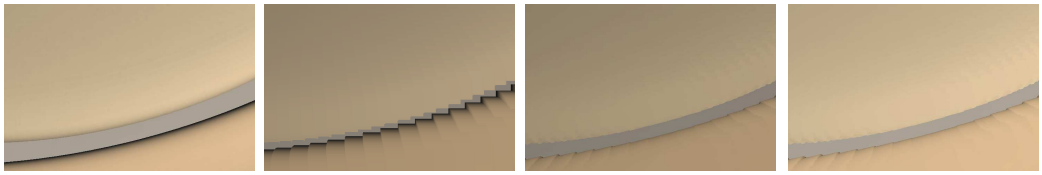
3. **Elliptical Primitive**: We initialize 256 texture maps representing ellipses (with a unit radius along the semi-major axis) varying from a sphere to a nearly "flat" ellipse. The texture maps are not Gaussian, they just have an alpha value of 0 in the interior of the ellipse and 1 elsewhere. At run time, we texture map the rectangle primitive with a scaled version of the closest approximation of its ellipsoid. We the render the texture-mapped rectangles with a small depth offset and enable their blending [99]. We have implemented this in hardware using the register combiners.

We compare differential point rendering with the above splatting primitives using three test cases. In the first test case we represent the model with the same number of DPs and splat primitives and compare their rendering quality. In the second test case we control the sampling of DPs and the context-blind samples so that they yield approximately similar visual quality of rendering. In the third test case we control the sampling of DPs and context-blind samples so that they all have the same rendering speed. Our results are summarized in table 6.2 and Figure 6.2. For the first test case we found that DPs deliver a much better rendering quality for the same number of primitives. DPs especially fared well in high curvature areas which are not well modeled and rendered by the splat primitives. Moreover, DPs had nearly the same frame rates as the ellipsoidal primitive. But DPs were slower than the square and rectangle primitives and required more disk space.

We control the sampling of the points for the remaining two test cases using uni-

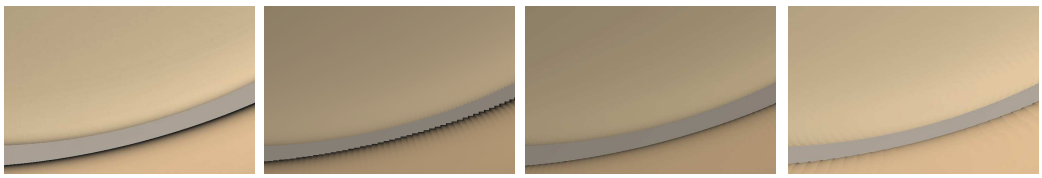
Statistical Highlights		Rendering Primitive			
		DP	*SP*	*RP*	*EP*
Test 1	Number of Points	156,800	156,800	156,800	156,800
	Storage Space (in MB)	9.19	4.90	4.90	4.90
	Frames per second (Diffuse)	2.13	11.76	10.52	2.35
Test 2	Number of Points	156,800	1,411,200	1,155,200	320,000
	Storage Space (in MB)	9.19	44.10	36.10	10.01
	Frames per second (Diffuse)	2.13	1.61	1.49	1.16
Test 3	Number of Points	156,800	1,036,800	819,200	180,000
	Storage Space (in MB)	9.19	32.4	25.6	5.6
	Frames per second (Diffuse)	2.13	2.05	2.06	2.02

Table 6.2: *Comparison with Splatting Primitives: (*Test 1*) Same Number of Rendering Primitives, (*Test 2*) Approximately similar rendering quality, (*Test 3*) Similar frame rates. DP = Differential Points, SP = Square Primitive, RP = Rectangle Primitive, and EP = Elliptical Primitive.*



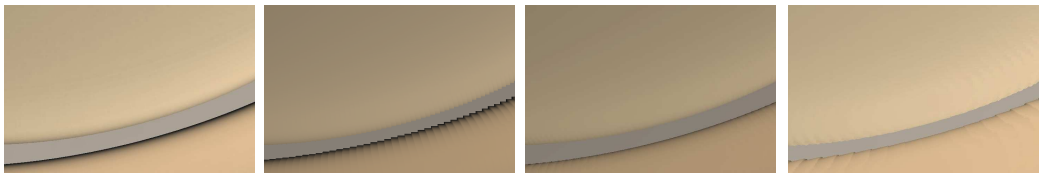
(2.13 fps) (11.76 fps) (10.52 fps) (2.35 fps)

Test 1: Comparison of rendering quality for the same number of primitives (157K points)



(157K points, 2.13fps) (1411K points, 1.61 fps) (1155K points, 1.49 fps) (320K points, 1.16 fps)

Test 2: Comparison of primitives for similar rendering quality



(a) Differential Points (157K points) (b) Square Primitive (1037K points) (c) Rectangle Primitive (819K points) (d) Elliptical Primitive (180K points)

Test 3: Comparison of rendering quality for a rendering speed of 2.1 fps

Figure 6.2: *Selected areas of rendering of the teapot model for the three test cases*

form sampling. Results from the second test show that DPs clearly out-perform the splatting primitives both in frame rates and in the storage space requirements. The third test shows that for the same frame rates DPs produced better rendering quality using fewer rendering primitives.

6.2 Statistical Point Geometry

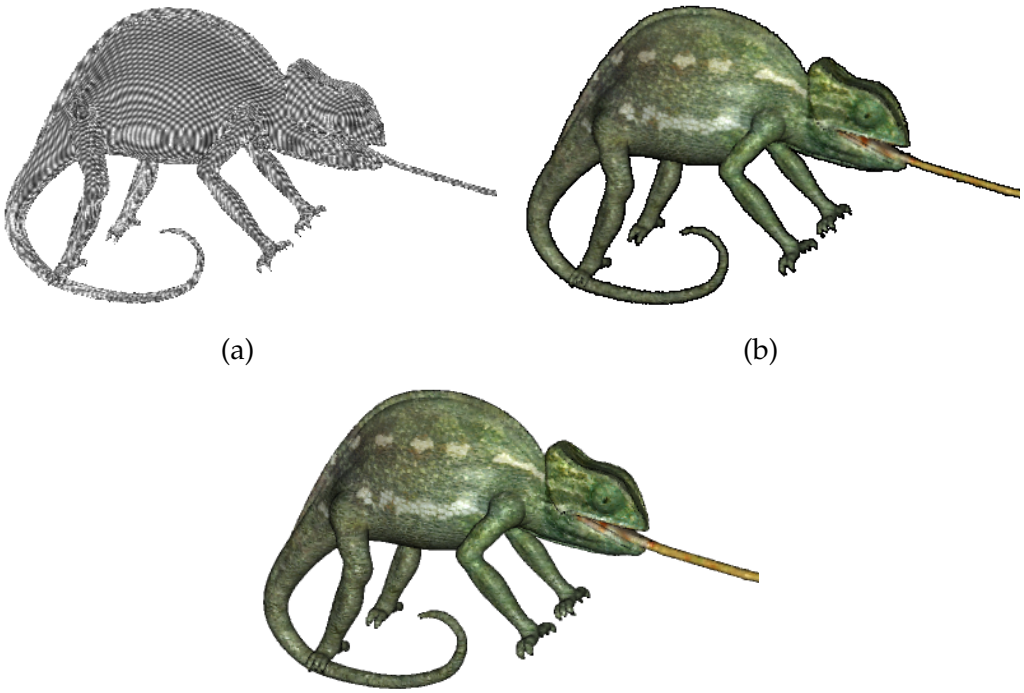
In this section we will quantify the benefits of statistical samples. Like we did for differential points, we will first compare the quality and speed of rendering of statistical points with that of splatting. We will then highlight the balance and efficiency of our hierarchical representation by comparing it against an octree-based representation. Afterwards, we will go on to discuss some of the applications of a statistical representation.

We did all our tests on a 2.4 GHz Pentium IV PC with 2GB RAM and a NVIDIA GeForceFX 5800 GPU. Our test models were the Stanford's David's Head, the full David's Statue, the Lucy model, and the St. Matthews face. We have added colors to the David and Lucy models by solid texturing. We have also tested our work on two raw LIDAR range scans of the Murder Scene (courtesy of the 3rd Tech Inc.). Except for registration, we did not do any other processing on the two scans. These datasets took no more than two hours of preprocessing each, with the classification and quantization phase taking up most of the time. For classifying the variances we used a naive partition-based k-means clustering scheme. Advanced clustering schemes should improve this number dramatically [31].

6.2.1 Comparison to Splatting

Since splatting uses two-dimensional tangent plane Gaussian distributions it is natural to ask if our three-dimensional Gaussian nodes can be used for splatting as well. In this section we show how the statistical information can be used for splatting and compare the speed and rendering quality of splatting with that of statistical point generation.

Splatting requires a 2D tangent plane weight function at each point. The points are rendered by projecting the support of the weight function to the screen and accu-



(a) (b)

(c)

Figure 6.3: *Figure (a): The per-pixel cumulative weight accumulated in the second pass of the splatting algorithm. Figure (b): The final rendering after per-pixel normalization at 9 FPS (42.6K surfels). Figure (c): Rendering of the model by points generation at 29 FPS (42.6K nodes, 79.7K generated points)*

mulating the weighted color contribution at each pixel. The final color at the pixel is computed by normalizing the color by the cumulative weight contribution from the individual points/Surfels. We refer the reader to [95, 129] for a thorough treatment of splatting.

We can splat our nodes with the help of elliptical surfels derived from our nodes. We represent the elliptical surfels by considering the two most significant components of the ellipsoidal distribution. Hence the surfels are centered at the means μ_p, and have standard deviations of σ_p^1 and σ_p^2, along the vectors f_p^1 and f_p^2 respectively. We modify the Gaussian weight function of the surfel as follows:

$$w(u,v) = \max\Big(\exp\Big(-\frac{1}{2}\Big(\Big(\frac{u}{\sigma_p^1}\Big)^2 + \Big(\frac{v}{\sigma_p^2}\Big)^2\Big)\Big) - \exp\Big(-\frac{1}{2}\gamma^2\Big), 0\Big)$$

In this function, the γ factor acts to limit the infinite support of the Gaussian function. We choose $\gamma = 3.0$ which corresponds to a Confidence Index (CI) greater than 99.5%. We ren-

Model	Tree	# Node	# Leaf	APR	σAPR	# NTC	FPS
David Head	Octree	1012K	784K	5.19	33.4K	201.8K	6.3
	SPG	903K	452K	1.24	1.14	80.4K	9.7
David Statue	Octree	1882K	1445K	5.04	11.4K	53.3K	24.1
	SPG	1843K	921K	1.20	1.16	21.8K	35.2
Lucy	Octree	1525K	1204K	12.25	4.9K	74.0K	17.6
	SPG	1330K	665K	1.17	1.06	28.3K	26.8

Table 6.3: *Comparison of our hierarchy (SPG) with an octree-based point hierarchy. APR: average partitioning ratio, i.e. the average ratio of the largest and smallest cardinalities amongst the children of a node. σAPR: same as APR except we compare the maximum and minimum values of σ_p^1 amongst the children of a node. NTC: Number of nodes in the tree cut. FPS: Rendering speed in frames per second. For the NTC and FPS comparisons we rendered both hierarchies with view-dependent rendering (without normal culling) on a 512×512 window at 2.5× distance from the object center.*

der each surfel as a tangent plane rectangle centered at μ_p, with widths of $2\gamma\sigma_p^1$ and $2\gamma\sigma_p^2$ along f_p^1 and f_p^2 respectively. We map each such rectangle with a texture corresponding to a spherically symmetric weight function with a standard deviation of $\frac{1}{\gamma}$. We can then deliver the value of the weight function at each pixel simply by assigning texture coordinate values of (0,0), (1,0), (1,1), and (0,1) to the corners of the rectangle [95]. We found that the quality of rendering by our statistical point generation scheme is comparable to that of splatting (see Figures 6.3(b) and 6.3(c)). Moreover, point generation is about $3\times$ faster than splatting. This is mainly because splatting needs three rendering passes as opposed to one in our case.

6.2.2 Comparison to Octree-based Representations

Hierarchical representation of point geometry based on the octree hierarchy is very popular [11, 125, 129]. The primary advantages of an octree-based hierarchy are: (1) The implicit structure of the octree can be used to efficiently represent the means of the nodes [11], (2) The octree structure can be used for reducing the cumulative computation in applications such as hierarchical rendering [11] and hierarchical computation of the covariance matrix [84]. However, a key disadvantage of the octree subdivision is that it can be highly imbalanced. To illustrate the importance of a balanced tree we did an octree subdivision of the point set and computed the PCA attributes of the points in each node. We cut off the octree subdivision when the number of points in a node was less

than the user-specified cutoff value (we used the same value as for our method). We then rendered the dataset by selecting a cut in the octree and generating points per node using our technique. For view-dependent rendering, we simply used the recursive-tree-traversal technique of QSplat [99], in combination with our method of estimating the screen-space area of the node. Our findings are shown in table 6.3. The table shows that our method leads to a tree with a lesser number of nodes and also has less number of leaf nodes. The table also shows that the average partitioning ratio (APR), i.e. the average ratio of the number of points in the largest and the smallest children of a node, is much closer to 1 in our case. This shows that our partitioning is much more balanced than a plain octree-based partitioning. Moreover, when we compared the largest standard deviations (σ_p^1) of the children and took the ratio of the maximum and minimum of these values, our numbers were more closer to 1 (see the "σAPR" column). This shows that not only does our partitioning balance the number of points, it also balances the volume of the partitions. Also note that a 1-to-2 partitioning offers a finer control in setting the tree cut when compared to a 1-to-8 partitioning. This advantage, combined with the balanced nature of our tree, gave us a big reduction in the number of nodes in the tree cut during view-dependent rendering (see the "# TCN" column of table 6.3). This typically translates to a higher rendering speed since the main bottlenecks are at the CPU and the AGP bus. Both the renderings were made without normal culling. For the results shown in table 6.3 we used the recursive tree traversal of QSplat [99] for rendering both trees. The rendering speeds using our method (without normal culling) was roughly twice as fast as the octree case.

6.2.3 Compression

The representation of the geometry as a hierarchical probability distribution is very effective for compression. Given any set of points, a typical un-compressed 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. Our PCA representation can encode any set of points with just 13 bytes, which means that we start saving with a PCA representation as soon as the

(a) (b) (c) (d)

Figure 6.4: *Figure (a) shows that the basic shape of the Lucy model is captured with just 32 PCA nodes. Figures (b) and (c) show a closeup of the Lucy model when rendered with points generated from level 19 of the hierarchy with 24 levels. We generated 14 million points from 480K nodes for the entire model. This corresponds to 2.32 bits/vertex approximation of the geometry and normals with about 71dB PSNR (Hausdorff) error. Figure (c) is rendered with hardware anti-aliasing, while Figure (b) is rendered without anti-aliasing. Figure (d) shows the original Lucy model (rendered with anti-aliasing).*

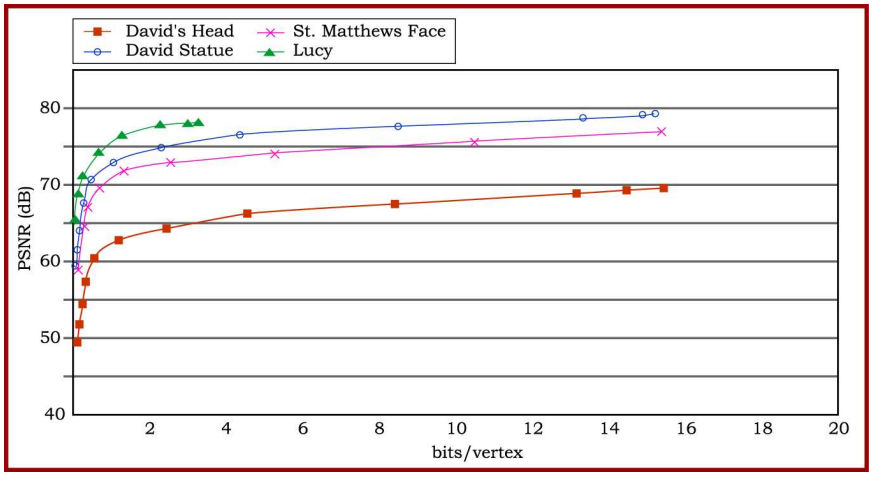
number of points in the set exceeds two. The processing of the Lucy dataset yielded us a tree of about 1.33 million nodes of which about 665K nodes are at the leaf level. We classified the variances into 2400 classes of spatial variances σ_p, 1800 classes of normal variances σ_n, and 64 classes of color variances σ_c. While the original 14 million points of the Lucy dataset required about 112MB of data, our total representation including the hierarchy and the classification requires about 18MB. We can achieve significant compression by substituting the original point set with the points generated with quasi-random sampling.

The compression however comes at the cost of an approximation error. Figure 6.4(a) shows the nodes of the Lucy model at a coarse resolution. Figure 6.4(b) shows the approximation of the Lucy dataset with 2.32 bits per vertex for geometry and normals. We measure the approximation error as the Peak Signal to Noise Ratio (PSNR) as measured by the Hausdorff distance metric [90]. At each node, we generate the same number of points as the number of points that the node represents and determine the nearest original point of that node for each generated point. This nearest-neighbor association is a conservative estimate of the Hausdorff distance between the original and the generated

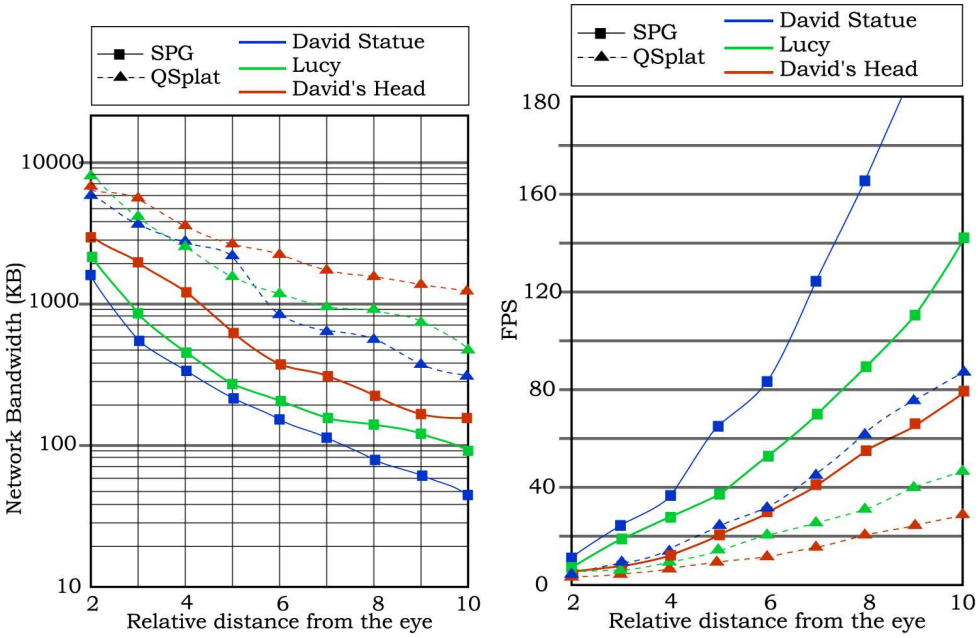
points. The PSNR is given as $20 \log_{10}(Peak/d)$, where d is the root-mean-squared distance of the generated points from the original points in the Hausdorff distance metric and $Peak$ is the length of the diagonal of the bounding box. Figure 6.5(a) shows our rate-distortion curve for various datasets. Our results compare well to the compression results by Praun and Hoppe [90]. We also compare our compression to that of Botsch *et al.* [11]. While the PSNR error rates for their compression are not available, we could get a rendering quality similar to theirs using 13.25 bits/vertex for position and normal (David's Head). In the compression chart of Figure 6.5(a) this corresponds to 8.66 bits/vertex for encoding just the position. For the David's Head model, Botsch *et al.* [11] needed 10.2 bits/vertex (position and normal) on the hard disk after gzipping, and their memory foot print was 32 bits/vertex. Our byte requirements are the same for both hard disk and memory. Hence octree is better when it comes to storage on disk while our approach is better in terms of memory footprint. Note that the memory footprint is especially important when visualizing large models.

6.2.4 Network Bandwidth Reduction

The compression of the geometry reduces the storage size on the disk. However, the growing use of graphics over networks makes geometry bandwidth reduction very important. This can be critical for several communication channels such as the Internet, Wi-Fi 802.11x, Universal Serial Bus (USB), and DSL links over land lines. Moreover, geometry bandwidth is also an issue for distributed-computing environments where the bandwidth is not large enough to keep the graphics cards busy [53]. The nodes of our tree, given their compact representation and order independence, are well suited for reducing the network bandwidth in client-server settings. To illustrate the reduction in the network bandwidth we set up an experiment where the camera eye is placed at various distances relative to the object center and the object is visualized in a view-dependent fashion. For every such distance, we rotated the object around an axis aligned with the y-axis of the camera and we measured the average network bandwidth required to transmit the PCA information of the nodes. We did all our tests on a 1024×1024 test window.

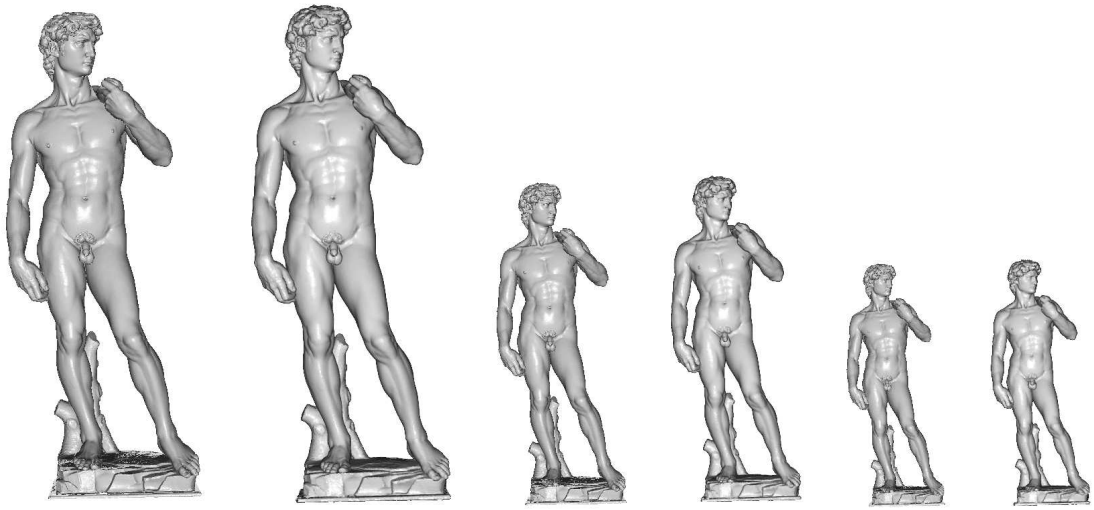


(a)



(b) (c)

Figure 6.5: *Figure (a) shows our rate-distortion curves for compressing various models. Figure (b) shows the reduction in network bandwidth while Figure (c) shows our rendering speedup. Comparisons in Figures (b) and (c) are with respect to QSplat.*



(a) (b) (c) (d) (e) (f)

	(a) QSplat	(b) SP	(c) QSplat	(d) SP	(e) QSplat	(f) SP
# Nodes	124K	121K	667K	44K	47K	26K
Frames/sec	4.8	10.2	9.6	25.0	14.3	31.2
Bandwidth	4.9 MB	1.6 MB	2.7 MB	0.57 MB	1.9 MB	0.35 MB
# Gen. Pts.	-	311K	-	121K	-	73K

Figure 6.6: *We compare results from view-dependent rendering of Statistical Points (SP) with that of QSplat for varying distances of the eye from the object center. The terms of comparison are the number of nodes chosen for rendering, the frame rates, the geometry bandwidth, and the number of generated points.*

We compared the results of our approach with QSplat [99, 100]. The results are shown in Figure 6.5(b) and a few snapshots of the test are shown in Figure 6.6. 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. The results show that we consistently achieve several-fold reduction in network bandwidth. This may be attributed to the better representation of the local geometry by our anisotropic probability distribution than by the isotropic-spherical approximation of QSplat. However, we note that this improvement is at the cost of approximately regenerating the original data.

6.2.5 Rendering

The best rendering quality currently available for rendering points is through splatting [11, 129]. Statistical point generation when combined with hardware FSAA can give high quality rendering as well. It can handle high frequency textures and delivers a quality that is comparable to splatting (see Figures 6.3(b) and 6.3(c)). In addition, statistical

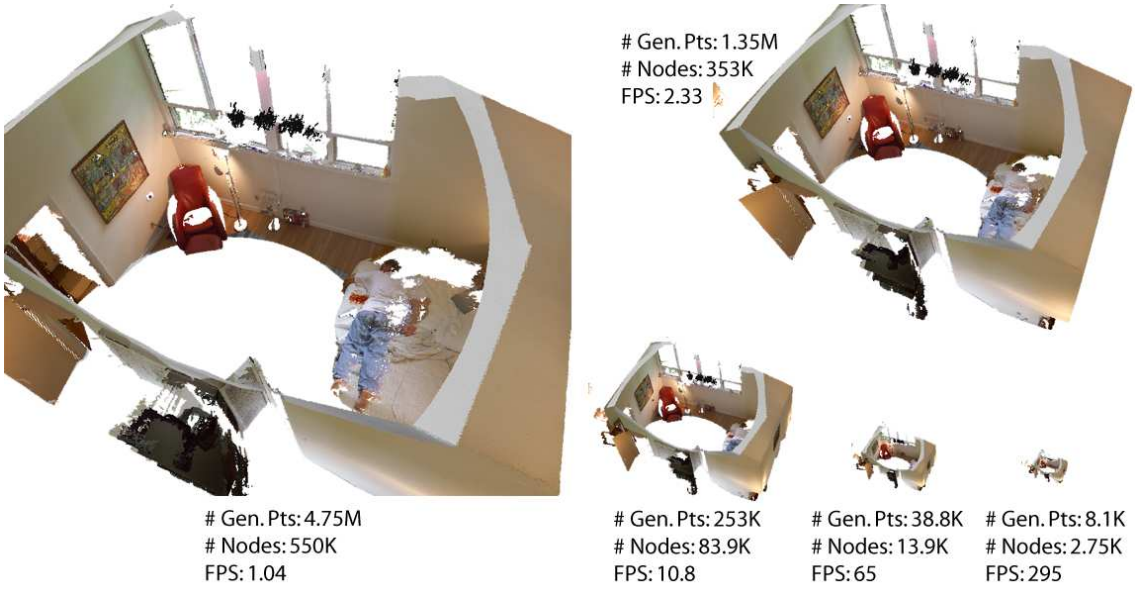


Figure 6.7: *The Murder Scene model as seen from various distances from the eye. These renderings were made on a* 1024×1024 *window. Note that the noise in the scanned data (black cloud) and edges are well handled.*

point generation has the benefit of a single pass rendering which allows it to outperform the speed of splatting by a factor of $2\times$ to $3.5\times$ during view-dependent rendering.

Since statistical generation can predict the geometry instead of fetching it from the memory, fewer nodes suffice during view-dependent rendering. This enhances the efficiency of view-dependent rendering in two ways: (1) it reduces the number of memory fetches at the server (CPU), and (2) it decreases the bus-bandwidth to the client (GPU). These factors together give us a significant speedup. Sampling on the GPU is about 30% faster than sampling on the CPU. We expect this factor to be even better with more programmable GPUs. However, for on-demand rendering we are about half as slow as rendering the nodes as opaque rectangles, but our rendering quality is much better as illustrated in Figures 5.5 and 5.6. Figure 6.7 illustrates view-dependent rendering of the Murder Scene model.

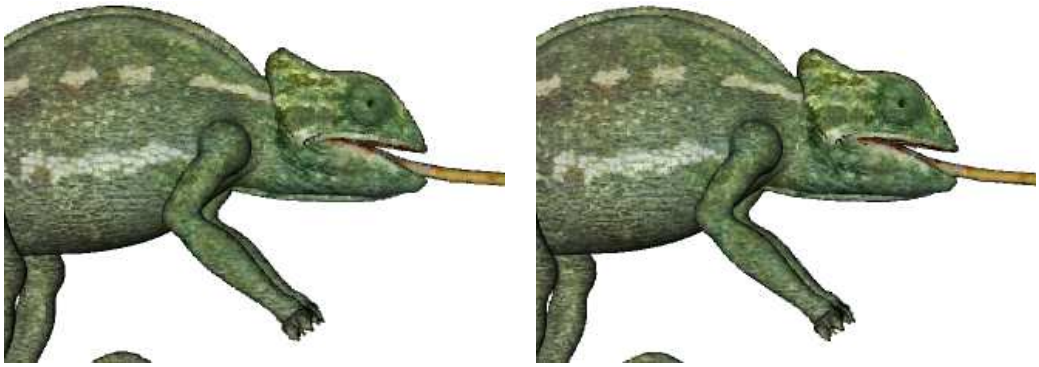
Statistical point generation was able to deliver about 29 FPS for a VLOD (view-dependent level-of-detail) rendering of the Chameleon model on a 512×512 window. Its rendering speed – 10 FPS for the Davids Head model on a 512×512 window – is better than the VLOD splatting scheme of [84] (1 FPS). It is comparable to the speed

of non-hierarchical splatting of [95], which ran at 19 FPS on a GeForce4 Ti4400 for the Chamaleon model. Botsch and Kobbelt [10] got superior speeds of 70 FPS for a non-hierarchical rendering of the Chameleon model by keeping the geometry in the video memory using the *ARB_vertex_buffer_object* extension. Guennebaud and Paulin [48] could achieve similar rendering speeds for comparable geometry sizes. Our approach matches the rendering speed (9.5 FPS) of Botsch and Kobbelt [10] for the David's Head model on a 512×512 window. Moreover, since Botsch and Kobbelt [10] use the video memory to store the geometry they cannot accommodate rendering of large models (for example, they have to subsample the Davids Head model to about 1 million points). This is not a problem for SPG since the main dataset resides in the system memory. In addition, SPG can deliver much higher rendering speeds when the object is far away. Therefore with respect to the current state of the art, we are comparable to non-hierarchical-splatting at full screen resolution and significantly better than VLOD-based splatting.

While splatting can deliver a high-quality rendering it can be slower than a point- or a quad-based rendering. The best publicly available software for fast point-based rendering of large datasets is QSplat [99]. We outperform QSplat by a factor of $2\times$ to $3\times$ (see Figures 6.6 and 6.5(c)). Dachsbacher *et al.* [24] map QSplat to GPU and render their nodes as opaque squares. By keeping the entire dataset on the graphics card they can deliver a rendering speed of nearly 50 million points per second (MPS) on ATI Radeon 9700. We could get a rendering speed of 56 MPS with color for the David's Statue model. In addition, since our dataset is system memory resident, we can handle much larger datasets and deliver a more detailed rendering.

6.3 Unified-Attribute Statistical Points

Generation of samples in the combined 8D space is more expensive than generating the attributes in the individual attribute spaces. This is because the cost of matrix fetch and matrix-vector multiplication is higher in the 8D space. We also end up generating more points since the spherical screen-space area estimation is quite conservative. Generating points on the GPU is not viable since transferring the 64-element matrix (per-node)



(a) (b)

Figure 6.8: *Figure (a) shows the view-dependent rendering of the Chameleon model on a 512×512 window using PCA analysis in the individual attribute spaces (33.7K nodes, 61.08K generated points, 31.2 FPS) while Figure (b) shows a view-dependent rendering of the Chameleon model built using PCA in the unified attribute space (34.7K nodes, 645K generated points, 1.6 FPS).*

to the GPU can be quite expensive. Moreover, while the correlation of the attributes of the generated points is informative in the lower-resolutions, we found that the correlation does not add much perceptual improvement in the higher-resolutions (see Figures 6.8(a) and 6.8(b)). These two factors combine to make the PCA in the unified PCA space unattractive for view-dependent rendering. A view-dependent rendering of the David's Head model gave us a rendering speed of about 1 frame per second on a 512x512 window. However, this approach can serve to represent lower-resolution versions of the data very well. Alternately, one could represent the lower resolution nodes in the unified PCA space and the higher resolution nodes in the independent attribute spaces to achieve the best of both worlds.

6.4 Summary

In this chapter we discussed the implementation details and quantified the benefits of context aware samples. We found that CAS reduce the bandwidth significantly and push the load of visualization from memory to computation. The overall effect of these influences is that CAS are highly efficient for storage, transmission, and rendering of large graphics models.

Chapter 7

Hierarchical Shadow Computation using Statistical Points

Shadows are an important visual cue in the understanding of complex models such as the ones encountered in medical visualization. Prior research in shadow computation has shown that shadowing is a spatial or a volumetric attribute as opposed to being a surface attribute. In previous chapters we have shown that the surface can be visualized directly using CAS primitives and that no explicit surface reconstruction is necessary. We believe that surface reconstruction is not necessary for several other applications. In this chapter we validate our belief by showing how CAS primitives can be used for efficient shadow computation.

Traditional shadow computation for surfaces involves determining the actual surface point that occludes a given light ray. In this chapter, we propose making this visibility decision based on the statistical properties of the surface. This approach is more geared towards complex surfaces where there is no single large occluder due to which an actual ray-surface intersection is computationally expensive. Our hierarchy of statistical points enables us to make the visibility decision faster. This is mainly because our algorithm allows us to limit the number of ray-node intersection tests.

7.1 Previous Work in Shadow Computation

Shadow computation has been pursued with interest since the early days of visualization as it falls in the broad area of visibility computation. We only provide a brief summary here and refer the reader to Durand's doctoral thesis [33] for a thorough survey. There are two classes of shadow computation algorithms: object-space-based approaches and image-space-based approaches. The early object-space methods involved visibility determination by navigating spatial hierarchies with ray-tracing. The bounding volume hierarchy was successfully used for such complex geometry as fractal trees and stochas-

tic terrain surfaces [64, 97]. Such hierarchies have since popularly used convex objects such as sphere, cube, and in general, enclosing parallel slabs, as the bounding volumes.

Ellipsoid was first introduced as a bounding volume for ray-tracing stochastic surfaces [12]. Since the surface was obtained by a recursive subdivision of an initial mesh, Bouville used the hierarchy to fit ellipsoids at each level of the recursion and achieved superior results compared to a plain polygonal bounding volume. For architectural scenes where the occluders are often large polygons, polygonal bounding volumes, octree-based [43], grid-based [16, 41, 56, 68, 124], and BSP tree-based spatial subdivision schemes [1, 85] were found to be more efficient. In contrast to object-space methods, analytical methods offer higher accuracy of visibility, albeit at a higher cost of computation and robustness issues. Of these, shadow volume techniques [8, 19, 20] have generally been used in conjunction with the Z-buffer. Discontinuity meshing techniques [30, 32, 34, 51, 112] have been used for generating precise soft shadows in conjunction with global illumination.

Image-space-based methods primarily involve using the hardware z-buffer for solving the visibility problem and are generally fast and robust compared to the object space approaches. The traditional shadow-buffer technique [121] has also been used for generating soft shadows by multiple sampling of an area light source [50] and more efficiently by using convolution [105]. The inherent aliasing of the shadow buffer can be countered with view-dependent methods such as the hierarchical Z-buffer [44], adaptive shadow maps [38], and perspective shadow maps [107]. However, since they require shadow computation at every frame, they are more efficient for animated environments and large polygonal environments where shadows could lie within a single large polygon. Stamminger and Drettakis [107] also discuss extending perspective shadow maps for procedural point geometry where the sampling density can be changed in a view-dependent fashion.

Shadow computation for volume rendering is supported in hardware by deep shadow maps. More recently, Nulkar and Mueller [82] precompute the visibility using light volumes and achieve high-quality rendering using image-aligned splatting. Zhang and Crawfis [128] reduce the memory requirement by using a splatting volume renderer

at both the light source and the eye point.

7.2 Overview of our approach

Shadow computation on point-sample geometry involves addressing two key considerations: correctness and efficiency. Points being dimensionless primitives, they cannot be used as occluders. What we need is a good estimate of the region around a point to determine if it blocks the light ray in question. We use a statistical measure of the point neighborhood for this decision. Our hierarchical clustering algorithm hierarchically partitions the geometry into leaves representing small regions of the geometry. Each node of this hierarchy is characterized by its PCA signature. This information can be used to determine, with varying degrees of confidence, if that node blocks any given light ray. Note that one could alternately use other bounding primitives such as a tangent disk, sphere, cube, etc. and use the traditional ray-object intersection test.

Our approach is similar in principle to the traditional bounding-volume-based approach. One could also consider an analytical object-space solution for point geometry, but it is likely to be expensive. Alternately, one could use image-based methods such as the shadow-buffer algorithm in conjunction with fast point-based-rendering techniques [59, 99], but it would suffer from inherent aliasing problems. Adaptive view-dependent shadow-map-based methods [38, 44, 107] are also possible. However, such a method would require shadow computation at each frame. Moreover, object-space-based visibility methods are still essential for other applications such as global illumination [33] and precomputed radiance transfer [104].

7.3 Hierarchical Shadow Computation

The basic operation in our shadow-computation algorithm is the statistical-point-light-ray intersection test. This is illustrated in 2D in Figure 7.1. We will pose it as a visibility problem. Let $\mathbf{l}$ and $\mathbf{p}$ be two points in the space of the frame of the statistical node $\mathbf{n}$. To determine if the line segment $\overline{\mathbf{lp}}$ intersects the geometry represented by the node $\mathbf{n}$ we need to figure out the closest point on the line segment $\overline{\mathbf{lp}}$ to $\mu_{\mathbf{n}}$. We measure distances in

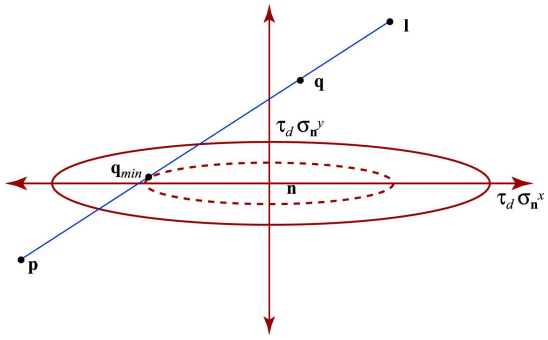


Figure 7.1: *The statistical-point-light-ray intersection test: To check if the light ray originating at* $\mathbf{l}$ *reaches the point* $\mathbf{p}$ *we determine the "nearest point"* $\mathbf{q}_{min}$ *on the line segment* $\overline{\mathbf{lp}}$.

the Mahalanobis metric to minimize the function $m(\mathbf{q}, \mathbf{n}) = \|M_{\mathbf{n}}\mathbf{q}\|$, we first note that any point $\mathbf{q}$ on the line segment $\overline{\mathbf{lp}}$ can be parameterized as $\mathbf{q} = \mathbf{p} + t(\mathbf{l} - \mathbf{q})$, where $0 \leq t \leq 1$. The nearest point can now be found by substituting this in the equation $\frac{\mathrm{d}(m(\mathbf{q},\mathbf{n}))}{\mathrm{d}t} = 0$ to get:

$$\mathbf{q}_{min} = \mathbf{p} - \frac{\mathbf{p}^T M_{\mathbf{n}}^T M_{\mathbf{n}}(\mathbf{l} - \mathbf{p})}{\|M_{\mathbf{n}}(\mathbf{l} - \mathbf{p})\|^2}(\mathbf{l} - \mathbf{q}). \tag{7.1}$$

We note that $m(\mathbf{q}_{min}, \mathbf{n})$ is a measure of how close the light ray is to the points represented by the node and we call it the *distance metric*, $d(\mathbf{p}, \mathbf{n}, \mathbf{l})$. If we assume a Gaussian distribution within the node, then $d(\mathbf{p}, \mathbf{n}, \mathbf{l})$ corresponds to a Confidence Interval (CI) of $\mathrm{erf}(\frac{d(\mathbf{p},\mathbf{n},\mathbf{l})}{\sqrt{2}})$, where $\mathrm{erf}(\ )$ is the standard error function. We set a threshold, τ_d, for the value of $d(\mathbf{p}, \mathbf{n}, \mathbf{l})$ and conclude that the light ray is not occluded by the node if $d(\mathbf{p}, \mathbf{n}, \mathbf{l}) > \tau_d$. For all our tests we set $\tau_d = 3.5$ which corresponds to a CI of at least 99.7%. In other words if $d(\mathbf{p}, \mathbf{n}, \mathbf{l}) > 3.5$, then the chances of finding a occluding point to the light ray is at most 0.3%. If the value of t is not within the range $[0, 1]$, then we set $d(\mathbf{p}, \mathbf{n}, \mathbf{l})$ to infinity to signify the fact that there is no occlusion. If the distance $\|M_{\mathbf{n}}\mathbf{p}\| \leq \tau_d$ then the query point $\mathbf{p}$ is within the valid bounds of the node $\mathbf{n}$. So in this case we set $d(\mathbf{p}, \mathbf{n}, \mathbf{l})$ to zero. This ensures that our hierarchical shadow computation algorithm goes down one level in the hierarchy to check if the point is indeed inside the node $\mathbf{n}$. If however, $\mathbf{n}$ is a leaf node, then we set $d(\mathbf{p}, \mathbf{n}, \mathbf{l})$ to infinity.

If $d(\mathbf{p}, \mathbf{n}, \mathbf{l}) \leq \tau_d$, then the light ray may or may not be occluded. The chances of the node occluding the light ray are higher if it represents a flat region of the surface.

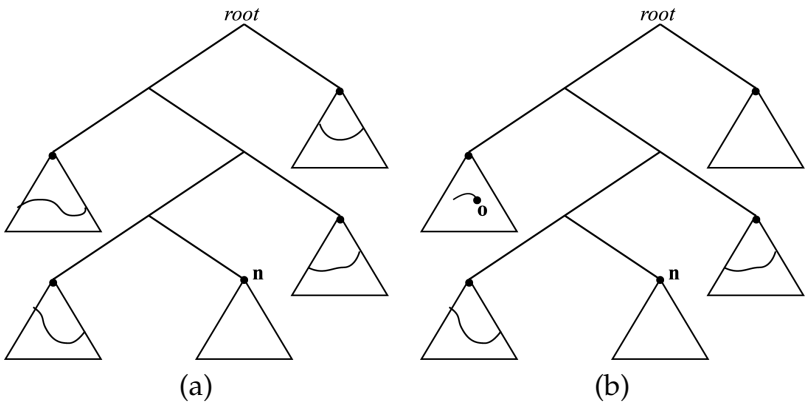


Figure 7.2: *Hierarchical shadow computation for the node* $\mathbf{n}$ *involves checking all subtrees in the path from* $\mathbf{n}$ *to the root. Case (a): Light ray is not occluded. A cut is found in each subtree such that no node in the cut occludes* $\mathbf{n}$*. Case (b): Light ray is occluded. An occluder node,* $\mathbf{o}$*, is found in one of the subtrees. The occluder node does not have to be a leaf node and not all subtrees may have to be searched.*

We define a *confidence metric*, $c(\mathbf{n})$, to be a measure of the confidence with which we can conclude that a light ray is occluded if its $d(\mathbf{p}, \mathbf{n}, \mathbf{l}) \leq \tau_d$. We use the third variance, $\sigma^3_{\mathbf{n}}$, as this metric since it a good measure of the flatness of the surface. As with the distance metric, we use a threshold τ_c for the confidence metric as well. If $c(\mathbf{n}) \leq \tau_c$ then we conclude that the ray is occluded. Since the model is normalized to fit into a unit cube, the value τ_c is not dependent on the input point cloud. We used a τ_c value of 10^{-4} for all our test cases.

The overall shadow computation is done in a hierarchical manner. The user specifies the accuracy of the shadows by choosing the minimum value of $\|\sigma_{\mathbf{n}}\|$ at which the shadow computations are made. We do an inorder traversal of the tree to form a tree-cut involving the nodes that are either leaf nodes or their $\|\sigma_{\mathbf{n}}\|$ is less than the threshold. We then traverse along the cut and determine the visibility of each node $\mathbf{n}$ from the light source $\mathbf{l}$. This test is described in pseudo-code by the function *IsShadowed*($\mathbf{n}$, $\mathbf{l}$) in Figure 7.3. This function tests if the mean $\mu_{\mathbf{n}}$ of a given node $\mathbf{n}$ is visible from $\mathbf{l}$. It does so by traversing the tree from $\mathbf{n}$ to the root and determining if the subtree at each ancestor node (the one that does not belong to the path) occludes the node $\mathbf{n}$. This way of the tree traversal ensures that we do not encounter the case where the surface point at which the visibility is being tested falls within the given node. The occlusion test of each subtree is done by the function *IsSubtreeOccluding*($\mathbf{p}$, $\mathbf{n}$, $\mathbf{l}$) shown in Figure 7.3. This is a recursive

```
IsSubtreeOccluding( p, n, l )
1  If ( IsLeaf( n )
2    return ( d(p, n, l) ≤ τ_d );
3  If ( d(p, n, l) > τ_d )
4    return false;
5  Else If ( c(n) ≤ τ_c )
6    return true;
7  If ( IsSubtreeOccluding( p, n.child[0], l ) )
8    return true;
9  return ( IsSubtreeOccluding( p, n.child[1], l ) );

IsShadowed( n, l )
1  For each subtree, s, in the path from n to the root
2    If ( IsSubtreeOccluding( μ_n, s.root, l ) )
3      return true;
4  return false;
```

Figure 7.3: *Hierarchical algorithm to determine the shadowing of the node* $\mathbf{n}$ *by the light source* $\mathbf{l}$.

test which terminates if it has found an occluding node or if it has found a cut in the subtree such that all of its nodes fail the distance-metric test. This is illustrated in 2D in Figure 7.2. Note that in lines 7, 8, and 9 of the *IsSubtreeOccluding*( ) pseudo-code we have the option of choosing the first child to to be subject to recursion. The child whose distance metric is smaller is chosen first since it has a better chance of finding an occluder, if any.

The inorder determination of the tree-cut ensures that spatially close nodes are ordered close to each other. We use this to exploit the coherence in shadows. The main observation is that any occluder of a mean $\mu_{\mathbf{n}}$ is also likely to occlude the mean of another node that is spatially close to $\mathbf{n}$. This coherence gets stronger the closer the cut is to the leaf levels. We exploit this coherence by simply caching the last occluder. So we first test if $\mu_{\mathbf{n}}$ is occluded by the cached node and move on to the hierarchical test only if this fails. This simple scheme gave us a significant reduction in the number of statistical-point-light-ray tests when a node is occluded.

We modify our approach built around hard shadows for computing soft shadows efficiently. We assume a spherical area light source and at each query point we compute the fraction of the light area that is visible. For this we take the traditional approach of

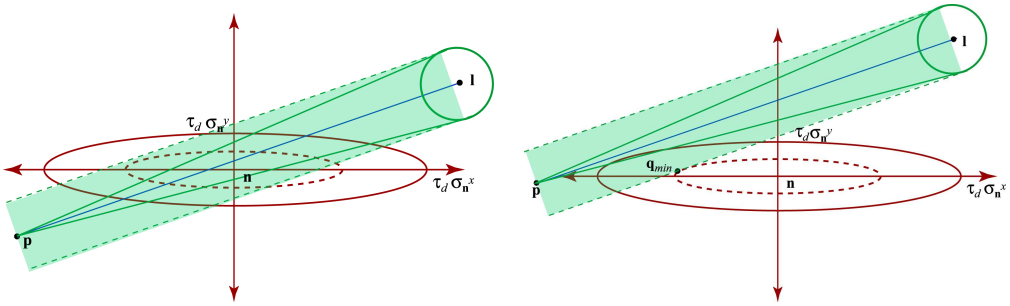


Figure 7.4: *The statistical-point-light-cone intersection test: A test to check if any of the light rays originating at the area light source* $\mathbf{l}$ *reaches the point* $\mathbf{p}$*. The visibility light cone is approximated by a cylinder. There are two cases to consider: (a) The mean* $\mu_\mathbf{n}$ *falls within the volume of the cylinder, (b) The Mahalanobis distance between the node* $\mathbf{n}$ *and any point on the surface of the cylinder is less than* τ_d*. We test the cases in this order and conclude that we have a potential (partial) occlusion if either of them is a success.*

sampling points on the surface of the light source, and at each query point, we determine the fraction of these points that are visible. The basic test in this case is a statistical point-light cone visibility test. This is illustrated in 2D in Figure 7.4. This test analytically determines if there will be an occlusion for any of the light rays originating from the source. If this test is successful then we do an individual statistical-point-light-ray intersection test between the query point $\mathbf{p}$ and the points of the light source.

We approximate the statistical-point-light-cone intersection test with a statistical point-light cylinder intersection test. We place a cylinder that tightly encloses the spherical light source and is directed along the line between the query point $\mathbf{p}$ and the center of the light source, $\mathbf{l}$. To do the statistical-point-light-cylinder visibility test efficiently we identify two test cases. First, if the mean $\mu_\mathbf{n}$ falls within the volume of the cylinder then there is at least a partial occlusion and conclude that the test is positive. If this test fails then there is a point on the surface of the cylinder which has the least Mahalanobis distance with respect to the node $\mathbf{n}$. This point has to be tangential to some concentric ellipsoid of $\mathbf{n}$ and hence should lie on the plane determined by the points $\mathbf{p}$, $\mu_\mathbf{n}$, and the center of the light source, $\mathbf{l}$. So we determine the line segment that corresponds to the intersection of this plane with the cylinder. There are two such line segments and we determine the one that lies between $\overline{\mathbf{lp}}$ and $\mu_\mathbf{n}$. We then use the statistical-point-light-ray intersection test to determine the distance metric between this line segment and $\mu_\mathbf{n}$. If this metric is less than or equal to τ_d then we conclude that there is a potential (partial)

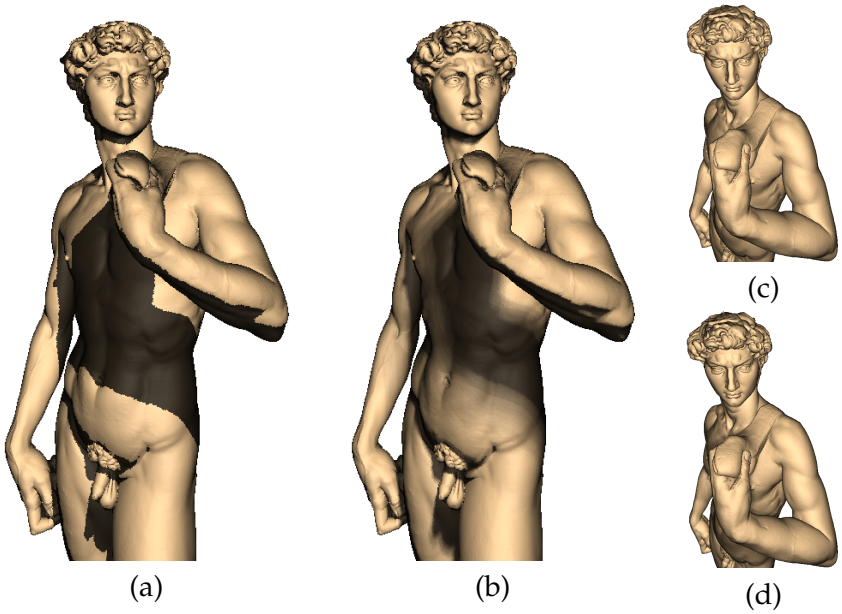


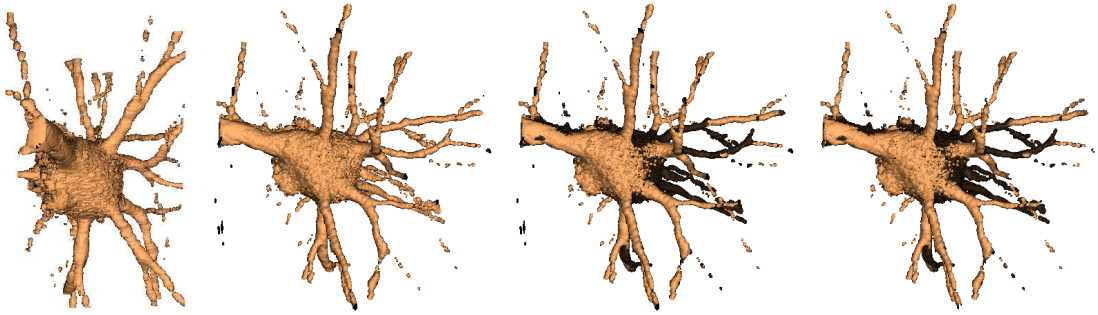
Figure 7.5: *Figures (a) and (b) are the renderings of the David with hard and soft shadows respectively. Figure (c) is the rendering of hard shadows with the view point being at the light source. Figure (d) is the soft shadow analogy of Figure (c).*

occlusion of the query point $\mathbf{p}$ by the node $\mathbf{n}$.

The soft-shadow computation is done hierarchically using an algorithm similar to the one described in Figure 7.3. We traverse along the inorder cut of the tree and determine the fraction of the light source points that are visible at each node $\mathbf{n}$ of the cut. This test is done by travelling along the path of the tree from $\mathbf{n}$ to the root and determining all the light rays that are occluded by the other subtree at each of the ancestor nodes. This is a recursive test similar to the function *IsSubtreeOccluding( )* of Figure 7.3. At each node we do a statistical-point-light-cylinder intersection test. If this test is successful then we recurse to test its children. We terminate the recursion if the node is a leaf or if its confidence metric $c(\mathbf{n}) < \tau_c$. The actual statistical-point-light-ray intersection tests between $\mu_{\mathbf{n}}$ and the points of the light source are only performed at this stage. We keep track of the light rays that are already occluded and do the test only for those that are not occluded. In the context of the *IsSubtreeOccluding( )* pseudo-code, this is done at lines 2 and 6. The subtree recursion and the upward tree traversal to the root is terminated when all the light rays are occluded. This corresponds to line 7 of the *IsSubtreeOccluding( )* pseudo-code and line 2 of the *IsShadowed( )* pseudo-code.

	UNC Brain	Nerve Cell	David's Head	David (Full)
# points (in millions)	5.18	1.16	2.0	4.13
# nodes	1231K	532K	897K	1844K
Max. tree depth	24	24	22	25
Avg. # tests (occlusion)	52.17	48.04	35.49	28.11
Avg. # tests (no occlusion)	208.03	145.43	160.14	191.37
Avg. # tests (overall)	62.75	81.80	65.7	75.48
Time (hard shadows)	32.57s	17.85s	24.28s	56.75s
Time (soft shadows)	5.7m	6.2m	11.3m	130.1m

Table 7.1: *Summary of results for hierarchical shadow computation*



(a) (b) (c) (d)

Figure 7.6: *Figure (a) is the view of the Nerve Cell as seen from the light source. Figure (b) is a plain rendering without any shadows while Figures (c) and (d) are renderings with hard and soft shadows respectively.*

7.4 Results and Conclusions

We did all our tests on a 2.4GHz Pentium4 based PC with 2GB RAM and a NVIDIA Quadro4 graphics card. We tested our work on two kinds of models: medical datasets and scanned models. The medical datasets (UNC brain and the nerve cell) were obtained by sampling points on the isosurfaces extracted from their volume grid representation. We also tested our work on the Stanford's David's Head model and the full David version. These models took no more than an hour to build the hierarchy.

For the test cases we computed the shadows using leaf-level cuts. The results are summarized in table 7.3. Row four of this table lists the average number of statistical-point-light-ray intersection tests performed for an occluded query point during hard shadow computation. Row five lists this number when the point query is not occluded. The former number is smaller because of the early exit strategy and occluder caching.

Row six lists the overall average number of statistical-point-light-ray intersection tests per query point while rows seven and eight list the total time taken for the hard and soft shadow computation respectively. The results show that the (amortized) number of occlusion tests for a given query point is of the order of $O(\log n)$, where n is the number of nodes. Overall this is a $O(n \log n)$ algorithm since occlusion is tested at about $O(n)$ points. Note that unlike the shadow-buffer-based techniques, we do not have to recompute the shadows on a per-frame basis since the computation is done in the object space. We did not do any back-face culling of query points facing away from the light source. This is because the isosurfaces may not have a well-defined side. When back-face culling was used for the David's Head and the full David model we got a further reduction of about 40% over the number reported here. We believe that the time complexity can be greatly reduced by hierarchically doing a PCA node-PCA node occlusion test to reduce the number of occlusion queries. We leave this for future work. Figures 7.5 and 7.6 show example renderings for the test datasets. For computing the soft shadows we uniformly distributed 40 points on the spherical light source. We got as much as a 70% reduction in computation complexity compared to a brute-force method that would test each individual light ray separately.

Chapter 8

Conclusions and Future Work

In this dissertation we have presented a novel approach for representing visual data using context-aware samples (CAS). Context-aware samples are samples with embedded information that models the local vicinity of the sample. The embedded information allows us to capture the distribution of the visual data around the samples. We represent the overall visual data by the union of the individual contexts of the CAS. Our approach is a marked shift from traditional representations that use interpolation of individual samples or use modeling approaches such as parametric or implicit surfaces.

We distinguish between two kinds of context-aware samples: surface-based and space-based. Differential points are surface-based CAS that use the basic principles of differential geometry to capture the surface around the sample point using the local curvature information at the sample point. The local vicinity of a differential point is approximated by a second-order surface whose bounds are inversely related to the local curvature. Our simplification algorithm prunes excess differential points using a greedy pruning procedure. This feature allows us to sample the surface adaptively by allocating more samples to areas of high surface curvature. We render the differential points by rasterizing the local shape and coloring the screen pixels using the fragment shaders. Our experimental results show that for similar rendering quality the differential points are faster to render than the splatting primitives. Our results also show that differential points can produce much better rendering quality than splatting for the same frame rates.

Statistical points are space-based CAS that model the local context around a sample point as a Gaussian probability distribution. Given a raw set of sample points, we convert it to a statistical-point-based representation by hierarchically partitioning it in the spatial domain. We use k-means clustering to ensure a fair partitioning of the points at each step of this hierarchy-building process. For each node of the hierarchy we derive a single statistical point that represents all the raw sample points of that node. This

statistical point can be derived either by using PCA in the individual attribute spaces or by using a PCA in the unified attribute space of the raw sample points. We approximate the original sampled data by sampling the probability distribution of the statistical points. We render our statistical-point-based representation by selecting a cut in the hierarchy and sampling points for the selected nodes of the hierarchy. The user can choose between an on-demand or a view-dependent way to determine the cut. Our approach allows us to render on a variety of client rendering devices such as the GPU, remote PC, or a PDA. Our experimental results show that statistical points can be used as a compressed representation. Our results also show that statistical points are nearly twice as fast as the state-of-the-art in point-based rendering and that they deliver a high-quality rendering comparable to that of splatting.

8.1 Conclusions

The main features of CAS primitives are that they are independent of each other and that they have embedded local context information. These features translate to several advantages such as:

- *Robustness of representation*: Context-aware samples are robust at representing visual data. Differential samples, for example, do not have some of the differentiability constraints of parametric surfaces. Similarly, statistical points can represent arbitrary visual data and can easilt work with noisy data..

- *Rendering from compressed data*: The context of one CAS primitive can capture information about several sample points in the vicinity. This leads to much less number of samples and a overall reduction in storage space. For example, differential points require much less storage space for the same rendering quality as splatting. Similarly, statistical points achieve much better compression and give a substantial reduction in the network bandwidth when compared to splatting. Such rendering from compressed data contributes to an improved rendering performance in several ways: (1) it leads to lesser number of memory fetches, (2) it reduces the geometry bandwidth, and (3) it makes better use of the SPMD capabilities of modern

GPUs.

- *High-quality rendering*: The contextual information of the CAS primitives lead to better rendering quality. For example, differential points were found to give better rendering quality for the same rendering speed as splatting. Similarly, statistical points were at least twice as fast as splatting with comparable rendering quality.

- *Flexible streaming*: The order independence of the CAS allows us to stream them to the client device in a very flexible manner. For example, the on-demand rendering feature of statistical points requires little or no maintenance on the client side.

In short, the context-aware primitives offer an efficient representation for the storage, transmission, and rendering of visual data. This is a validation of the hypothesis of this dissertation as stated in §1.3.

8.2 Future Work

In this dissertation we have shown how context-aware samples can be used for efficient storage, transmission, and rendering. We have also shown how the statistical points can be used for shadow computation. We believe that CAS primitives can be used for many more applications. Differential points have already been used by other for global illumination of point-based geometries [117, 118]. Curvature information can also be used for surface-based operations such as texture synthesis over the surface. Our hierarchy of statistical points can be used for approximate nearest neighbor search [5]. The algorithm for this would look for the nearest neighbor of a sample point by recursing through the hierarchy and pruning subtree searches based on the probability of the search point with respect to the probability distribution of the subtree. An extension of our shadow computation algorithm can be used for efficient global illumination using ray tracing or photon mapping. A similar hierarchical approach can also be used to speed up applications such as collision detection. We believe that there is a potential to compress CAS-based representations even further. One possible approach to this could to use a spanning tree to link nearby CAS primitives and encode such a tree using delta encoding.

Our CAS primitives are disjoint and independent primitives. While this feature has several advantages it also has a disadvantage in that it can lead to visually noticeable discontinuities in regions of relatively low sampling. This problem can be potentially solved by using the partition of unity reconstruction. For example, a continuous surface can be reconstructed using the approach of radial basis functions where the Guassian probability distribution of statistical points acts as the weight function while the distance function is the distance along the f^3 axis of the node. Using modern GPUs such an implicit surface can be potentially rendered without any surface reconstruction.

In this dissertation we have considered two scenarios: the first scenario is the one in which all the attributes are represented with context-aware samples – this is the scenario that we have presented. A second scenario that has been explored by others is to mix point-based representations with triangle meshes [18, 28]. A third scenario could involve using CAS primitives for only a few select attributes while the rest of the attributes are represented using traditional methods such a linear interpolation. For example, consider the problem of visualizing the satellite images of the Earth. In this case the geometry of the earth could be considerably coarse as compared to the image resolution. In such cases the geometry could be represented using the traditional triangle meshes while the texture could be streamed to the client as CAS primitives.

Over the course of this dissertation we have come to believe that higher-order representations of the data could have a major impact on the representation of future visual datasets. Visual data representation using techniques such as dimensionality reduction, probability distribution functions, and kernel-space mapping appears promising. Mathematical models such as the Poisson point process can be very efficient for both representation and visualization. Also, we feel that there are many benefits to developing a mathematical model for fitting representations to the visual data that are global, parametric, and stochastic. We believe such higher-level representations can also be used for other operations such as data synthesis and querying. In this dissertation we have laid the basic steps towards this goal. We believe that the insights that we have gained during our research will plant the seeds for further research in this direction.

8.3 Data and Funding Acknowledgments

We thank Marc Levoy and the Stanford Graphics Laboratory for providing us with the Bunny, David's Head, David, Lucy, and the St. Matthews face models. We thank Robert McNeel & Associates for the head model, the camera model, and for the openNURBS code. Also thanks to Cyberware Inc. for the venus model. We thank 3rd Tech Inc. for the Murder Scene dataset. Also many thanks to the Computer Graphics Lab of ETH-Zurich for the Chameleon dataset. We thank Klaus Mueller for the UNC Brain dataset and the nerve cell dataset. We thank Dirk Bartz for providing us with the the blood vessel dataset. The nerve cell dataset is originally provoided by Noran Instruments. Last, but not the least, we would like to acknowledge NSF funding grants IIS00-81847, ACR-98-12572 and DMI-98-00690.

Bibliography

[1] P. K. Agarwal, J. Erickson, and L. J. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles (extended abstract). In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 107–116, January 1998.

[2] 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.

[3] P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of SIGGRAPH 2001*, pages 195–202, August 2001.

[4] N. Amenta, M. Bern, and M. Kamvysselis. A New Voronoi-Based Surface Reconstruction Algorithm. In *Proceedings of SIGGRAPH 98*, pages 415–422, 1998.

[5] S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *Proceedings of the Data Compression Conference (DCC'93)*, pages 381–390. IEEE Press, 1993.

[6] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proceedings of SIGGRAPH'95*, pages 109–118, August 1995.

[7] J. A. Beraldin, F. Blais, L. Cournoyer, M. Rioux, S. F. El-Hakim, R. Rodell, F. Bernier, and N. Harrison. Digital 3D imaging for rapid response on remote sites. In *Proceedings of 2nd International Conference on 3-D Imaging and Modelling*, pages 34–43, 1999.

[8] P. Bergeron. A general version of Crow's shadow volumes. *IEEE Computer Graphics and Applications*, 6(9):17–28, September 1986.

[9] 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.

[10] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern GPUs. In *Pacific Graphics'03*, pages 335–343, 2003.

[11] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Rendering Techniques'02*, pages 53–64. Eurographics, 2002.

[12] C. Bouville. Bounding ellipsoids for ray-fractal intersection. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 45–52, July 1985.

[13] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *Ann. Math. Stat.*, 28:610–611, 1958.

[14] D. Brodsky and B. Watson. Model simplification through refinement. In *Proceedings of Graphics Interface 2000*, pages 221–228, 2000.

[15] S. R. Buss and J. P. Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics*, 20(2):95–126, 2001.

[16] F. Cazals, G. Drettakis, and C. Puech. Filtering, clustering and hierarchy construction: a new solution for ray-tracing complex scenes. *Computer Graphics Forum*, 14(3):371–382, August 1995.

[17] C. F. Chang, G. Bishop, and A. Lastra. LDI Tree: A hierarchical representation for image-based rendering. In *Proceedings of SIGGRAPH'99*, pages 291–298, 1999.

[18] B. Chen and M. X. Nguyen. POP: A hybrid point and polygon rendering system for large data. In *IEEE Visualization'01*, pages 45–52, October 2001.

[19] N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. volume 23, pages 99–106, July 1989.

[20] Y. Chrysanthou and M. Slater. Shadow volume BSP trees for computation of shadows in dynamic scenes. In *1995 Symposium on Interactive 3D Graphics*, pages 45–50, April 1995.

[21] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[22] J. Cohen, D. Luebke, M. Reddy, A. Varshney, and B. Watson. Advanced issues in level of detail. In *Course notes(41) of SIGGRAPH 2000*, July 2000.

[23] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *IEEE Visualization '99*, pages 67–72, 1999.

[24] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. *ACM Transactions on Graphics*, 22(3):657–662, 2003.

[25] K.J. Dana, B. van Ginneken, S.K. Nayar, and J.J. Koenderink. Reflectance and texture of real world surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.

[26] L. Darsa, B. C. Silva, and A. Varshney. Navigating static environments using image-space simplification and morphing. In *Symposium on Interactive 3D Graphics*, pages 25–34, April 1997.

[27] M. F. Deering. Geometry compression. In *Proceedings of SIGGRAPH'95*, pages 13–20, August 1995.

[28] T. K. Dey and J. Hudson. PMR: Point to Mesh Rendering, A Feature-Based Approach. In *IEEE Visualization'02*, pages 155–162, October 2002.

[29] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall. Inc., Englewood Cliffs, New Jersey, 1976.

[30] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. *Computer Graphics*, 28(Annual Conference Series):223–230, July 1994.

[31] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.

[32] F. Duguet and G. Drettakis. Robust epsilon visibility. *ACM Transactions on Graphics*, 21(3):567–575, July 2002.

[33] F. Durand. *3D Visibility: analytical study and applications*. PhD thesis, Université Joseph Fourier, Grenoble I, July 1999. http://www-imagis.imag.fr.

[34] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In *SIGGRAPH 97 Conference Proceedings*, pages 89–100, August 1997.

[35] D. Ebert, F. Musgrave, P. Peachey, K. Perlin, and S. Worley. *Texturing & Modeling: A Procedural Approach*. AP Professional, San Diego, 3rd edition, 2002.

[36] M. Eck and H. Hoppe. Automatic reconstruction of B-spline surfaces of arbitrary topological type. In *Proceedings of SIGGRAPH'96*, pages 325–334, August 1996.

[37] C. Fermuller, Y. Aloimonos, and A. Brodsky. New eyes for building models from video. *CGTA: Computational Geometry: Theory and Applications*, 15:3–23, 2000.

[38] R. Fernando, S. Fernandez, K. Bala, and D. P. Greenberg. Adaptive shadow maps. In *SIGGRAPH 2001 Conference Proceedings*, pages 387–390, August 2001.

[39] S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva. Progressive point set surfaces. *ACM Transactions on Graphics*, 22(4):997–1011, 2003.

[40] L.De Floriani, L.Kobbelt, and E. Puppo. A Survey on Data Structures for Level-Of-Detail Models. In N.Dodgson, M.Floater, and M.Sabin, editors, *Advances in Multiresolution for Geometric Modelling, Series in Mathematics and Visualization*, pages 49–74. Springer Verlag, 2004.

[41] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986.

[42] P.-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics*, 21:372–379, 2002. (also in Proceedings of SIGGRAPH'02).

[43] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

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

[45] J. P. Grossman and William J. Dally. Point sample rendering. In *Rendering Techniques '98*, Eurographics, pages 181–192. Springer-Verlag Wien New York, 1998.

[46] X. Gu, S. Gortler, and H. Hoppe. Geometry images. In *Proceedings of SIGGRAPH 2002*, pages 355–361, August 2002.

[47] G. Guennebaud, L. Barthe, and M. Paulin. Deferred Splatting. In *Computer Graphics Forum*, volume 23, pages 1–11. September 2004. (Also in Proceedings of EUROGRAPHICS'04).

[48] G. Guennebaud and M. Paulin. Efficient screen space approach for Hardware Accelerated Surfel Rendering. In *Vision, Modeling and Visualization, Munich*, pages 1–10. IEEE Signal Processing Society, November 2003.

[49] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proceedings of SIGGRAPH 99*, pages 325–334, 1999.

[50] P. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 24(4):309–318, August 1990.

[51] P. S. Heckbert. Discontinuity meshing for radiosity. In D. Paddon, A. Chalmers, and F. Sillion, editors, *Rendering Techniques '92*, Eurographics, pages 203–216. Consolidation Express Bristol, 1992.

[52] A. Hilton, J. Illingworth, and T. Windeatt. Statistics of surface curvature estimates. *Pattern Recognition*, 28(8):1201–1222, 1995.

[53] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: A stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics*, 21(3):693–702, July 2002.

[54] M. Isenburg and J. Snoeyink. Face fixer: Compressing polygon meshes with properties. In *Proceedings SIGGRAPH 2000*, pages 263–270, 2000.

[55] H. W. Jensen. Global illumination using photon maps. In *Rendering Techniques '96*, pages 21–30, 1996.

[56] David Jevans and Brian Wyvill. Adaptive voxel subdivision for ray tracing. pages 164–172, June 1989.

[57] C.R. Johnson and A.R. Sanderson. A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics and Applications*, 23(5):6–10, September 2003.

[58] A. Kalaiah and A. Varshney. Differential point rendering. In *Rendering Techniques '01*, Eurographics, pages 139–150. Springer-Verlag Wien New York, 2001.

[59] 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.

[60] A. Kalaiah and A. Varshney. Statistical point geometry. In *Eurographics Symposium on Geometry Processing*, pages 113–122, June 2003.

[61] A. Kalaiah and A. Varshney. Non-deterministic geometry representation for efficient transmission and rendering. *ACM Transactions on Graphics (To Appear)*, 2005.

[62] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.

[63] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *Proceedings of SIGGRAPH 2000*, pages 279–286, 2000.

[64] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, August 1986.

[65] A. Keller. Quasi-Monte Carlo Methods in Computer Graphics: The Global Illumination Problem. In *Lectures in Applied Mathematics*, volume 32, pages 455–469. SIAM, 1996.

[66] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proceedings of SIGGRAPH 2000*, pages 271–278, 2000.

[67] M. J. Kilgard. A practical and robust bump-mapping technique for today's GPUs. In *Game Developers Conference*, July, 2000 (available at http://www.nvidia.com).

[68] K. S. Klimaszewski and T. W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, January 1997.

[69] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of SIGGRAPH'96*, pages 313–324, August 1996.

[70] 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.

[71] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings of SIGGRAPH 96*, pages 31–42, 1996.

[72] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo Project: 3D scanning of large statues. In *Proceedings of SIGGRAPH 2000*, pages 131–144, July 2000.

[73] 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.

[74] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH 2000*, pages 259–262, July 2000.

[75] D. Lischinski and A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98*, Eurographics, pages 301–314. Springer-Verlag Wien New York, 1998.

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

[77] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. In *1997 Symposium on Interactive 3D Graphics*, pages 7–16, April 1997.

[78] G. Marsaglia. Choosing a point from the surface of a sphere. *Ann. Math. Stat.*, 43(2):645–646, April 1972.

[79] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Proceedings of VisMath'02*, 2002.

[80] K. Mueller, T. Moller, and R. Crawfis. Splatting without the blur. In *IEEE Visualization'99*, pages 363–370, October 1999.

[81] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992.

[82] M. Nulkar and K. Mueller. Splatting with shadows. In *International Workshop on Volume Graphics'01*, pages 35–50, 2001.

[83] M. M. Oliviera and G. Bishop. Image based objects. In *ACM Symposium of Interactive 3D Graphics*, pages 191–198, 1999.

[84] R. Pajarola. Efficient level-of-details for point based rendering. In *Proceedings IASTED Computer Graphics and Imaging Conference (CGIM)*, 2003.

[85] M. S. Paterson and F. F. Yao. Binary partitions with applications to hidden-surface removal and solid modelling. In *Proceedings of the Fifth Annual Symposium on Computational Geometry (Saarbrücken, FRG, June 5–7, 1989)*, pages 23–32, New York, 1989. ACM, ACM Press.

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

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

[88] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics*, 22(3):641–650, July 2003.

[89] 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.

[90] E. Praun and H. Hoppe. Spherical parametrization and remeshing. *ACM Transactions on Graphics*, 22(3):340–349, July 2003.

[91] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, 2 edition, January 2003.

[92] W. H. Press and S. A. Teukolsky. Quasi- (that is, sub-) random numbers. *Computers in Physics*, 3(6):76–79, 1989.

[93] P. Rademacher and G. Bishop. Multiple-center-of-projection images. In *Proceedings of SIGGRAPH 98*, pages 199–206, August 1998.

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

[95] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Eurographics'02*, pages 461–470, September 2002.

[96] J. Ritter. Fast 2D-3D Rotation. In A. Glassner, editor, *Graphics Gems*, pages 440–441. Academic Press, Boston, 1990.

[97] S. M. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 110–116, July 1980.

[98] H. Rushmeier, G. Taubin, and A. Guéziec. Applying shape from lighting variation to bump map capture. In *Rendering Techniques'97*, pages 35–44. Springer-Verlag Wien New York, June 1997.

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

[100] 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.

[101] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

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

[103] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *Proceedings of SIGGRAPH 98*, pages 231–242, August 1998.

[104] P.-P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics*, 21(3):527–536, July 2002.

[105] C. Soler and F. X. Sillion. Fast calculation of soft shadow textures using convolution. In *SIGGRAPH 98 Conference Proceedings*, pages 321–332, July 1998.

[106] J. Spanier and E. M. Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley, New York, NY, 1969.

[107] M. Stamminger and G. Drettakis. Perspective shadow maps. In *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, pages 557–562, 2002.

[108] E. M. Stockely and S. Y. Wu. Surface parameterization and curvature measurement of arbitrary 3-D objects: Five practical methods. *Pattern Analysis and Machine Intelligence*, 8:833–840, August 1992.

[109] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Fifth International Conference on Computer Vision*, pages 902–907, 1995.

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

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

[112] Seth J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics*, 26(2):139–148, July 1992.

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

[114] G. Turk. Re-tiling polygonal surfaces. In *Proceedings of SIGGRAPH 92*, pages 55–64, July 1992.

[115] K. Turkowski. Computing the inverse square root. In A. Paeth, editor, *Graphics Gems*, volume 5, pages 16–21. Academic Press, 1995.

[116] 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.

[117] M. Wand and W. Straβer. Multi-resolution point-sample raytracing. In *Proceedings of Graphics Interface'03*, 2003.

[118] M. Wand and W. Straβer. Real-time caustics. *Computer Graphics Forum*, 22(3):611–620, 2003. (Also in Proceedings of EUROGRAPHICS'03).

[119] T. Welsh and K. Mueller. A frequency-sensitive point hierarchy for images and volumes. In *IEEE Visualization'03*, pages 425–432, October 2003.

[120] S. H. Westin, J. R. Arvo, and K. E. Torrance. Predicting reflectance functions from complex surfaces. volume 26, pages 255–264, July 1992.

[121] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics*, 12(3):270–274, August 1978.

[122] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH 94*, pages 269–278, July 1994.

[123] T.-T. Wong, W.-S. Luk, and P.-A. Heng. Sampling with hammersley and halton points. *Journal of Graphics Tools*, 2(2):9–24, 1997.

[124] A. Woo. Recursive grids and ray bounding box comments and timings. *Ray Tracing News*, 10(3), 1997.

[125] J. C. Woolley, D. Luebke, and B. Watson. Interruptible rendering. In *SIGGRAPH'02 Technical Sketch*, page 205, 2002.

[126] J. Wu and L. Kobbelt. Optimized sub-sampling of point sets for surface splatting. In *Proc. of Eurographics*, pages 643–652, 2004.

[127] Wm. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, March 1995.

[128] C. Zhang and R. Crawfis. Volumetric shadows using splatting. In *Proc. IEEE Visualization'02*, pages 85–92. IEEE Computer Society, October 2002.

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