Modeling and Rendering of Points with Local Geometry

Aravind Kalaiah Amitabh Varshney
Graphics and Visual Informatics Laboratory
Department of Computer Science and UMIACS
University of Maryland, College Park. MD - 20742
{ark,varshney}@cs.umd.edu
Ph:(301)-405-1213
Fax:(301)-405-6707

Abstract

We present a novel rendering primitive that combines the modeling brevity of points with the rasterization efficiency of polygons. The surface is represented by a sampled collection of *Differential Points* (DP), each with embedded curvature information that captures the local differential geometry in the vicinity of that point. This is a more general point representation that, for the cost of a few additional bytes, packs much more information per point than the traditional point-based models. This information is used to efficiently render the surface as a collection of local geometries. To use the hardware acceleration, the DPs are quantized into 256 different types and each sampled point is approximated by the closest quantized DP and is rendered as a normal-mapped rectangle. The advantages to this representation are: (1) the surface can be represented more sparsely compared to other point primitives, (2) it achieves a robust hardware accelerated per-pixel shading – even with no connectivity information, and (3) it offers a novel point-based simplification technique that factors in the complexity of the local geometry. The number of primitives being equal, DPs produce a much better quality of rendering than a pure splat-based approach. Visual appearances being similar, DPs are about two times faster and require about 75% less disk space in comparison to splatting primitives.

Keywords

Differential geometry, simplification, point sample rendering, per-pixel shading

I. INTRODUCTION

POINT -based rendering schemes [1], [2], [3], [4], [5], [6] have evolved as a viable alternative to triangle-based representations. They promise benefits over polygon-based rendering in many areas: (1) modeling and rendering complex environments, (2) a seamless hierarchical structure to balance frame-rates with visual quality, and (3) efficient streaming over the network for remote rendering [7].

Current point primitives store only limited information about their immediate locality, such as normal, sphere of influence [5], and disk of influence on the tangent plane [4]. These primitives are then rasterized with flat shading and in some cases followed up with a screen-space filtering [1], [4]. Since the primitives are flat shaded, such representations require very high sampling to obtain a good rendering quality. In other words, the rendering algorithm dictates the sampling frequency in the modeling stage. This is clearly undesirable as it may prescribe very high sampling even in areas of low spatial frequency, causing two significant drawbacks: (1) slower rendering due to increase in rendering computation and related CPU-memory bus activity, and (2) large disk and memory utilization.

In this work we propose an approach of storing local differential geometric information with every point. This information gives a good approximation of the surface distribution in the vicinity of each sampled point which is then used for rendering the point and its approximated vicinity. The total surface area that a point is allowed to approximate is bounded by the characteristics of the surface at that point. If the point is in a flat or a low curvature region of the surface then the differential information at that point can well approximate a large area of the surface around it. Alternately, if the

Fig. 1. A Rendering of the Human Head model using Differential Points

point is located on a high-frequency area of the surface then we limit the approximation to a smaller vicinity. This scheme offers the potential of controlling the local sampling density based on its surface curvature characteristics. We present a simplification algorithm that takes the initial super-sampled point-based representation and returns a sparse subset whose local sampling density reflects the local surface variation. Our rendering algorithm uses per-pixel shading when a DP is rendered with its neighborhood approximation.

Our approach has many benefits to offer:

1. **Rendering**: The surface can be rendered with fewer (point) primitives by pushing more computation into each primitive. This reduces the CPU-memory bus bandwidth and the overall amount of computation resulting in a significant speed-up. As the processor-memory speed gap increases we expect this method to get even more attractive.

2. **Storage**: The reduction in the number of primitives more than compensates for the extra bytes of information stored with each point primitive. This leads to an overall reduction in the storage requirements. This reduction also benefits faster streaming of information over the network.

3. **Generality**: The information stored with our point primitives is sufficient to derive (directly or indirectly) the requisite information for prior point primitives. Our work is primarily focused on the efficiency of per-point rendering computation. It can potentially be used in conjunction with larger point-based organization structures - hierarchical (bounding balls hierarchy [5], Layered Depth Cube

(LDC) tree [4], Layered Depth Image (LDI) tree [8]) or otherwise (LDC [3], LDI [6]).

4. **Simplification**: Recently proposed point representations have a width (or a region of influence) associated with each point which can differ from one point to another. This can lead to a significant overlap in the surface representation by the points. Our point primitive is amenable to a simplification scheme that significantly reduces the redundancy in surface representation.

In the following sections, we first mention some related works and then outline the terminology from the differential geometry literature that will be used to describe our approach. This is followed by a discussion of our sampling and simplification schemes that will output sparse point representations. We then describe our rendering algorithm and compare it with some splatting schemes. We conclude the paper with a discussion of this approach and its possible extensions.

II. Previous Work

A. Differential Geometry and Curvature Estimation

Our approach of rendering points with their local geometry requires the knowledge of the surface variation at any given point. Classical differential geometry 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 paper we follow the terminology of M. P. do Carmo [9].

Curvature computation on parametric surfaces has a robust mathematical model. Various techniques have been designed to estimate curvature from discrete sampled representations [10], [11]. Taubin [12] estimates curvature at a mesh vertex by using the eigenvalues of an approximation matrix constructed using the incident edges. Desbrun et al. [13] define discrete operators (normal, curvature, etc.) of differential geometry using Voronoi cells and finite-element/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.

B. Acquisition and Processing

Point samples of real-world environments are acquired using several acquisition techniques [14], [15], [16], [17], [18] with the choice depending on the environment being sampled. This information is processed by surface reconstruction algorithms [19], [20] and subsequently denoised [21]. The sampled

points can also be processed directly using spectral processing techniques [22]. Alternately, the coarse triangle mesh can be fitted with parametric surfaces [23], [24] for denoising and to to aid other higher-level applications.

Point samples from synthetic environments are popularly acquired by shooting sampling rays from base image plane(s) [1], [6]. In this work we support point sampling from two kinds of surface representations: NURBS and polygonal mesh. If the input is a NURBS surface then we uniformly sample in the parametric domain of the surface. If the input is a polygonal mesh then we use its vertices as the sample points.

C. Simplification and Organization

The initial set of point samples may have significant redundancy in representing the surface due to super-sampling. The problem of pruning this set has not been given enough attention so far, but various hierarchical organization schemes have been used that develop lower frequency versions of the original set of point samples [4], [5], [8]. We use a simplification process to prune an initial set of points to obtain a sparse point representation. Turk [25] uses a point placement approach with the point density being controlled by the local curvature properties of the surface. Witkin and Heckbert [26] 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. [27] and Cohen et al. [28] document various simplification techniques. More recently, Lindstrom [29] uses error quadrics in a vertex clustering scheme to simplify complex datasets that are too large to fit into main memory.

Image-assisted organization of points [1], [3], [6] are efficient at three-dimensional transformations as they use the implicitness of relationship among pixels to achieve fast incremental computations. They are also attractive because of their efficiency at representing complex real-world environments. The multiresolution organizations [4], [5], [8] 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 [7].

D. Rendering

Darsa et al. [30], Mark et al. [31] and Pulli et al. [32] use a triangle mesh overlaid on the image sampling plane for rendering. It can be followed by a screen space compositing process. However, such systems can be expensive in computation and storage if high resolutions are desired. Levoy and Whitted [2] introduced points as a rendering primitive. It has been used by Shade et al. [6] and Grossman and Dally [1] for synthetic environments. However, raw point primitives suffer from aliasing problems and holes. Lischinski and Rappoport [3] raytrace a point dataset. Oliveira et al. [33] use image space transformations to render point samples. Rusinkiewicz and Levoy [5] use splatting of rectangle primitives for rendering. Chen and Nguyen [34] build a bounding ball hierarchy on top of a triangle mesh to render small triangles as points. Alexa et al. [35] derive a polynomial surface at each point which is then rendered by generating additional display points in a view-dependent fashion. Pfister et al. [4] follow up the splatting with a screen-space filtering process to cover up holes and to deal with aliasing problems. Zwicker et al. [36] derive a screen-space formulation of the EWA filter to render high-detail textured point samples with a support for transparency. Mueller et al. [37] achieve per-pixel shading for rectilinear grids using gradient splats.

The main focus of this paper is a novel rendering primitive that uses surface curvature properties to efficiently render an approximation of its local geometry. The main advantage of this approach is its sparse representation of the surface which leads to a significant reduction in the computational cost and CPU-memory bus traffic.

III. Differential Geometry for Surface Representation

Classical differential geometry is a study of the local properties of curves and surfaces [9]. It uses differential calculus of first and higher orders for this study. In this work we use the *regular surface* model which captures surface attributes such as continuity, smoothness, and degree of local surface variation. To quantify the surface variation around a point we use the directional curvature metric. This is a second-order description of how fast the surface varies along any tangential direction at a point on the surface. In this section we present a brief working introduction to regular surfaces with an outline of the terminology and the equations that will be used to explain our work in subsequent sections.

A regular surface is a differentiable surface which is non self-intersecting and which has a unique

tangent plane at each point on the surface. It is defined as a *set*, $\mathbf{S}$, of points in $\Re^3$ such that every element, $\mathbf{p}$, of this set has a *neighborhood* $\mathbf{V} \subset \Re^3$ and a parameterizing map $\mathbf{x} : \mathbf{U} \rightarrow \mathbf{V} \cap \mathbf{S}$ (where $\mathbf{U}$ is an open subset of $\Re^2$) satisfying the following conditions [9]:

1. $\mathbf{x}$ is differentiable
2. $\mathbf{x}$ is a homeomorphism
3. Each map $\mathbf{x} : \mathbf{U} \rightarrow \mathbf{S}$ is a regular patch

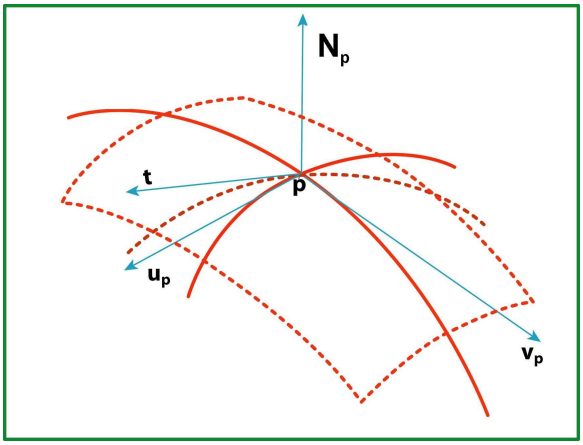


Fig. 2. Neighborhood of a Differential Point

Consider the unit normal, $\mathbf{N_p}$, at any point $\mathbf{p}$ on the surface. As one moves outside of $\mathbf{p}$ along the surface, the normal may change. This variation can be expressed as a function of the characteristics of $\mathbf{p}$ and the tangential direction of motion, $\hat{\mathbf{t}}$, by a linear map, $\mathbf{dN_p} : \Re^3 \rightarrow \Re^3$, called the differential of the normal at $\mathbf{p}$. $\mathbf{dN_p}(\hat{\mathbf{t}})$ gives the first-order normal variation at the point $\mathbf{p}$ along the direction $\hat{\mathbf{t}}$ and this vector is tangential to the surface at $\mathbf{p}$. The Jacobian matrix of $\mathbf{dN_p}$ has two eigenvectors, $\hat{\mathbf{u}}_\mathbf{p}$ and $\hat{\mathbf{v}}_\mathbf{p}$, together called the *principal directions*. The *direction of maximum curvature*, $\hat{\mathbf{u}}_\mathbf{p}$, is the tangential direction along which the normal shows maximal variation at $\mathbf{p}$. The eigenvalue associated with $\hat{\mathbf{u}}_\mathbf{p}$, $-\lambda_{\mathbf{u_p}}$, is a measure of the normal variation along $\hat{\mathbf{u}}_\mathbf{p}$. The term $\lambda_{\mathbf{u_p}}$ is the curvature at $\mathbf{p}$ along the direction $\hat{\mathbf{u}}_\mathbf{p}$ and is called the *maximum normal curvature*. Similarly $\hat{\mathbf{v}}_\mathbf{p}$ is called the *direction of*

minimum curvature and has an associated eigenvalue, $-\lambda_{\mathbf{v_p}}$. These attributes are related as follows:

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

where $\langle \cdot, \cdot \rangle$ is the vector dot product and $\times$ is the vector cross product operator. These relationships are illustrated in Figure 2. 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:

$$\begin{aligned} \mathbf{dN_p}(\hat{\mathbf{t}}) &= \mathbf{dN_p}(u\hat{\mathbf{u}}_\mathbf{p} + v\hat{\mathbf{v}}_\mathbf{p}) \\ &= u\mathbf{dN_p}(\hat{\mathbf{u}}_\mathbf{p}) + v\mathbf{dN_p}(\hat{\mathbf{v}}_\mathbf{p}) \\ &= -(\lambda_{\mathbf{u_p}} u\hat{\mathbf{u}}_\mathbf{p} + \lambda_{\mathbf{v_p}} v\hat{\mathbf{v}}_\mathbf{p}) \end{aligned} \tag{1}$$

Similarly, it can be shown that the normal curvature along $\hat{\mathbf{t}}$, $\lambda(\hat{\mathbf{t}})$, is given by [9]:

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

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

A salient feature of the regular surface model is that it can give a local description at a point without indexing any global properties. This gives complete independence to a point which defines its own local geometry without any reliance, explicit or implicit, on the immediate sampled neighborhood or on any other global property. We use this feature to render the surface as a set of locally defined geometries of varying surface area. This feature is also exploited for a variety of other applications in computer graphics. Interrante [38] uses it for visualizing iso-surfaces. Garland and Heckbert [39], among several others, use it for mesh simplification. Guskov et al. [21] use curvature information for surface denoising.

Most of today's virtual environments can be characterized as a collection of smooth surfaces meeting at points, curves, or areas of varying orders of discontinuity. This is in contrast to a regular surface

which, by definition, is differentiable and thus has continuous partial derivatives of all orders. However, as explained in the next section, we need the surface to be only second-order continuous to extract properties that will be used for rendering. Discontinuities of third or higher order, in most instances, are not easily discernible and thus we do not make any effort towards reproducing them visually here. However discontinuities of the zeroth, first, and second order are visually noticeable. While a regular surface model cannot represent these points explicitly, we use an implicit procedure to represent them. If such discontinuities are a collection of points or curves, then we sample at the second-order continuous neighborhood of these points and the discontinuities will be maintained implicitly by the intersection of the influence of these sampled points. If however, such points of discontinuity cover an area on the surface, or no such neighborhood exists, then they can be represented by point-primitives that do not use second-order information [1], [4], [5], [6], or by a polygonal model.

IV. Sampling and Processing

Our fundamental rendering primitive is a point with differential geometry information. Virtual environments, however, are modeled using other representations such as NURBS or a triangle mesh. We derive our rendering information by sampling points on these models and extracting differential geometry information per sampled point. To ensure that the information stored in each point collectively represents the whole surface area, we over-sample initially, and follow it up with a simplification process guided by a user-specified error bound. This reduces the overlap of the area of influence of the sampled points while still ensuring that the surface area is fully represented. This is a pre-process and the output is saved in a render-ready format that is an input to our rendering algorithm outlined in Section V.

A. Differential Points

We call our rendering primitive as a *differential point* (DP). A DP, $\mathbf{p}$, is constructed from a sample point and has the following parameters: $\mathbf{x_p}$ (the position of the point), $\lambda_{\mathbf{u_p}}$ and $\lambda_{\mathbf{v_p}}$ (the principal curvatures), and $\hat{\mathbf{u}}_\mathbf{p}$ and $\hat{\mathbf{v}}_\mathbf{p}$ (the principal directions). From these, we derive the unit normal, $\hat{\mathbf{n}}_\mathbf{p}$, and the tangent plane, $\tau_\mathbf{p}$, of $\mathbf{p}$. This information represents a coordinate frame and second-order information at each DP. We extrapolate this information to define a surface, $\mathbf{S_p}$, that will be used to approximate

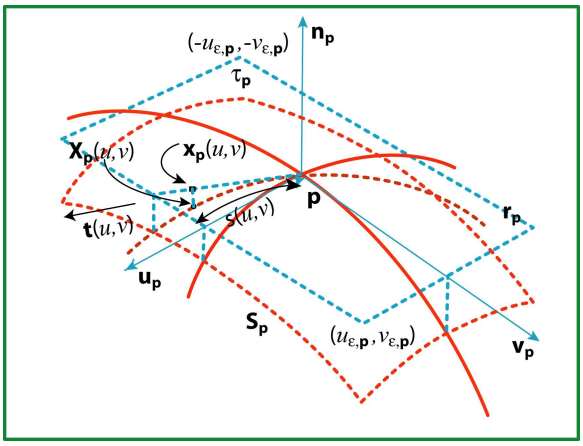


Fig. 3. Tangent plane parameterization of a Differential Point: The tangent plane $\tau_\mathbf{p}$ is parameterized by the (u,v) coordinates. $\mathbf{S_p}$ uses the same parameterization which relates it to $\tau_\mathbf{p}$ under the projection $\mathcal{P}_\mathbf{p}$. $u_{\epsilon,\mathbf{p}}$ is defined by the constraint $\|\mathbf{X_p}(u,0) - \mathbf{x_p}(u,0)\| \leq \frac{\epsilon}{\|\lambda_{\mathbf{u_p}}\|}$.

the neighborhood of $\mathbf{x_p}$. 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 3.

To aid our simplification and rendering algorithms we define a coordinate system on $\tau_\mathbf{p}$ and $\mathbf{S_p}$. The tangent plane $\tau_\mathbf{p}$ is parameterized by (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 $\hat{\mathbf{n}}_\mathbf{p} = \mathbf{N_p}(0,0)$. We use lower-case characters or symbols for terms related to $\tau_\mathbf{p}$ and we use upper-case characters or symbols for terms related to $\mathbf{S_p}$. A notable exception to this rule is the arc-length $s(u,v)$.

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:

$$\mathbf{N_p}(u,v) \;=\; \sum_{\hat{\mathbf{e}}=\hat{\mathbf{u}}_\mathbf{p},\hat{\mathbf{v}}_\mathbf{p},\hat{\mathbf{n}}_\mathbf{p}} \langle \mathbf{N_p}(u,v), \hat{\mathbf{e}} \rangle \; \hat{\mathbf{e}} \tag{3}$$

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 (3) as follows:

$$\begin{aligned} \mathbf{N_p}(u,v) \;&=\; \sum_{\hat{\mathbf{e}}=\hat{\mathbf{u}}_\mathbf{p},\hat{\mathbf{v}}_\mathbf{p},\hat{\mathbf{n}}_\mathbf{p}} (\langle \mathbf{N_p}(0,0), \hat{\mathbf{e}} \rangle + s(u,v) \; \langle \mathbf{dN_p}(\hat{\mathbf{t}}(u,v)), \hat{\mathbf{e}} \rangle + \text{Remainder Term}) \; \hat{\mathbf{e}} \\ &\approx\; \mathbf{N_p}(0,0) + s(u,v) \; \mathbf{dN_p}(\hat{\mathbf{t}}(u,v)) \end{aligned} \tag{4}$$

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 neccessarily 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 3. 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 that the "flatter" 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 $\mathbf{r_p}$ as a differential point because all of these are constructed from just the second-order information at a sampled point. While it is desirable to render $\mathbf{p}$ using $\mathbf{S_p}$, such a rendering primitive is not supported by current graphics hardware. Instead $\mathbf{p}$ is rendered using $\mathbf{r_p}$. This is explained in more detail in Section V.

B. Sampling

Given a 3D model, it is first sampled by points. Using the inherent properties of the surface representation, we extract differential geometry information at each of these sampled points. If the surface is a parametric one, such as NURBS, it is sampled uniformly in the parametric domain and standard techniques outlined in the differential geometry literature [9] are used to extract the relevant information at each sampled point. The principal directions cannot be uniquely determined at the umbilical points were the surface is either flat or is spherical ($\lambda_{\mathbf{u_p}} = \lambda_{\mathbf{v_p}}$). At such points the direction of maximum curvature, $\hat{\mathbf{u}}_\mathbf{p}$, is assigned to be the best of the projection of the x, y, and the z-axis onto the tangent plane. The direction of minimum curvature $\hat{\mathbf{v}}_\mathbf{p}$ can be computed from $\hat{\mathbf{u}}_\mathbf{p}$ and $\hat{\mathbf{n}}_\mathbf{p}$.

If the surface is a triangle mesh, then a NURBS surface can be fit to the triangle mesh [23], [24] 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 extracting differential information for each point using the techniques developed by Taubin [12]. The DPs thus obtained from the triangle mesh have the same properties as the ones obtained by sampling a NURBS surface. However, 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 IV-A then there might be gaps in the surface coverage. This is because the ϵ and δ values might prescribe small sizes to the surfaces $\mathbf{S_p}$ thus leaving holes in the surface coverage because the points may not be close enough for the assigned sizes to overlap. To deal with this issue we also factor in the distance of the neighbors

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

Fig. 4. Effectiveness of Simplification: (a) Wireframe rectangles corresponding to the initial (super-sampled) 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.

from a sample point (mesh vertex) in determining the point size as follows. For each point $\mathbf{p}$, the midpoint of every incident edge is projected along the average of its adjacent triangle normals onto the tangent plane $\tau_{\mathbf{p}}$ of the point. Then a rectangle is sized to contain these points such that it is: (1) axis aligned with respect to the $(\hat{\mathbf{u}}_{\mathbf{p}}, \hat{\mathbf{v}}_{\mathbf{p}})$ directions, (2) symmetric with respect to the origin (point $\mathbf{x}_{\mathbf{p}}$), and (3) its size is the smallest possible. The rectangle $\mathbf{r}_{\mathbf{p}}$ is set to the smallest rectangle that encloses the rectangle computed this way and the rectangle determined using the steps outlined in section IV-A.

C. Simplification

Initially the surface is super-sampled so that the rectangle of each differential point overlaps sufficiently with its neighbors without leaving any holes in the surface coverage. While this gives a complete surface representation, it also contains redundant samples. So we follow up the point sampling with a simplification process that prunes redundant DPs.

Simplification works by pruning those DPs whose geometric information is represented by the cumulative information of their neighbors within the error margins ϵ and δ. For this purpose, we first define a projection set, $\mathcal{O}(\mathbf{p})$, for each point $\mathbf{p}$. It denotes the set of all points of the original surface in the vicinity of $\mathbf{x}_{\mathbf{p}}$ that fall within the surface area covered by the orthographic projection of the rectangle $\mathbf{r}_{\mathbf{p}}$ onto the original surface along the direction $\hat{\mathbf{n}}_{\mathbf{p}}$. do Carmo [9] shows that for a vicinity around the point position $\mathbf{x}_{\mathbf{p}}$, 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.

Simplification involves comparing a point with its "neighbors" - denoted by the set $\mathcal{N}(\mathbf{p})$. $\mathcal{N}(\mathbf{p})$ is

initialized 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 $\mathbf{p} \in \mathcal{N}(\mathbf{q})$. If the differential points are sampled from a triangle mesh, then $\mathcal{N}(\mathbf{p})$ is chosen from the vertices with whom it 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:

$$\mathcal{N}(\mathbf{p}) \Leftarrow \mathcal{N}(\mathbf{p}) - \{\mathbf{q}_i\} \cup \{\mathbf{q}_j | \mathbf{q}_j \in \mathcal{N}(\mathbf{q}_i) \text{ and } \mathbf{q}_j \neq \mathbf{p} \text{ and } \mathcal{O}(\mathbf{q}_j) \cap \mathcal{O}(\mathbf{p}) \neq \phi\} \tag{5}$$

This operation updates $\mathcal{N}(\mathbf{p})$ by deleting $\mathbf{q}_i$ from it and adding to it 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 assumes 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. It is quantified 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 $\mathbf{p}$. $\mathcal{R}(\mathbf{p})$ is computed as follows:

$$\mathcal{R}(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} \left(\frac{|\langle \mathbf{N}_{\mathbf{q}}, \mathbf{N}_{\mathbf{p}}(u_{\mathbf{q},\mathbf{p}}, v_{\mathbf{q},\mathbf{p}}) \rangle| + |\langle \mathbf{N}_{\mathbf{p}}, \mathbf{N}_{\mathbf{q}}(u_{\mathbf{p},\mathbf{q}}, v_{\mathbf{p},\mathbf{q}}) \rangle|}{2\,|\mathcal{N}(\mathbf{p})|} \right) \tag{6}$$

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 $\mathbf{x}_{\mathbf{q}}$ onto $\tau_{\mathbf{p}}$ and $\mathbf{N}_{\mathbf{p}}(u_{\mathbf{q},\mathbf{p}}, v_{\mathbf{q},\mathbf{p}})$ is the normal estimated at these coordinates using equation (4). The dot product $|\langle \mathbf{N}_{\mathbf{q}}, \mathbf{N}_{\mathbf{p}}(u_{\mathbf{q},\mathbf{p}}, v_{\mathbf{q},\mathbf{p}}) \rangle|$ in equation (6) is a measure of how close the actual normal at $\mathbf{q}$ is to the normal estimated at $\mathbf{x}_{\mathbf{q}}$ 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.

After $\mathcal{R}(\mathbf{p})$ has been computed, $\mathbf{p}$ is inserted into a binary heap with $\mathcal{R}(\mathbf{p})$ as the key. After all the DPs are represented in the heap, an iterative process is started which pops the top of the heap and checks if pruning that DP will leave any holes in the surface representation. If not, then the point is pruned

and the neighborhood and the redundancy factor of all its ex-neighbors are updated using equations (5) and (6) respectively. Otherwise the point is marked for output. A pseudo-code of simplification is as follows:

```
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
```

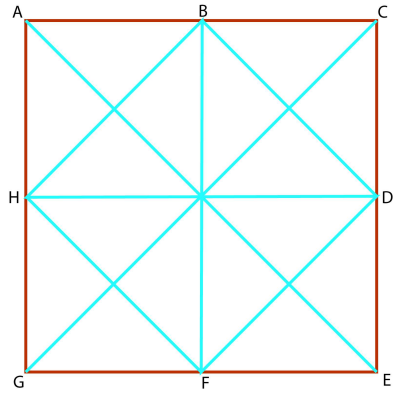


Fig. 5. The line segments AE, CG, BF, HD, BH, HF, FD, DB comprise the initial set of test line segments for the routine Enclosed($\mathbf{p}$)

For $\mathbf{p}$ to be pruned it has to qualify 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 original surface by $\tau_{\mathbf{p}}$, and test for the enclosure of $\mathbf{p}$ within this framework. This test is done by an approximation method that samples line segments on $\mathbf{r_p}$ (as shown in Figure 5) 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 as follows:

```
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)
```

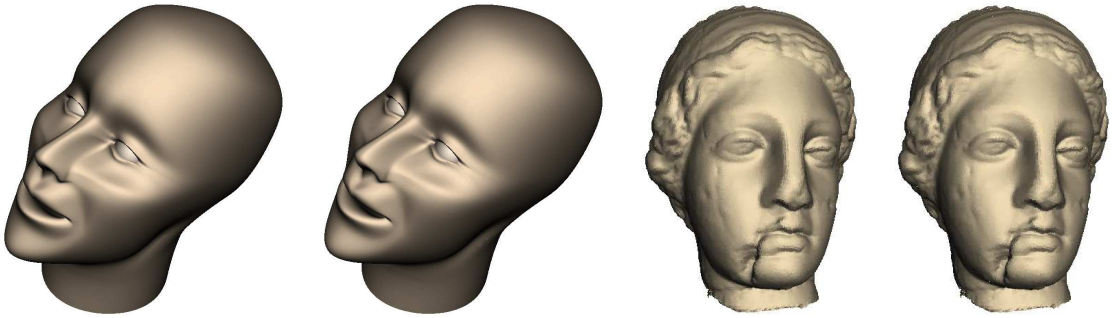
Since the coverage of line segments does not guarantee coverage of the entire area of $\mathbf{r_p}$, we see infrequent sliver gaps left between rectangles. We make the coverage test more conservative by scaling down the rectangles for simplification (the original rectangle sizes are used for rendering). 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 as follows:

```
Overlaps(q1, q2)
1   return (OverlapTest(q1, q2) || OverlapTest(q2, q1))

OverlapTest(q1, q2)
1   If the orthographic projection of x_q1 onto τ_q2 falls within the
    bounds of r_q2 then return true
2   If the orthographic projection of any of the end points of r_q1
    onto τ_q2 falls within the bounds of r_q2 then return true
3   If the orthographic projection of any of the edges of r_q1 onto
    τ_q2 intersects r_q2 then return true
4   If all the above tests fail then return false
```

All the approximation algorithms work well in our case owing to the similarity of neighboring rectangles in position, width, and orientation. Figure 4(b) shows the rectangles leftover after simplification



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

Fig. 6. 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

in an area where curvature-related features change very fast. 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 loss in perceptual quality. Figure 6 shows sample models rendered with and without simplification.

V. Rendering

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 $\mathbf{r_p}$ is used 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.

A. Normal Distribution

Consider the projection of $\mathbf{N_p}(u, v)$ onto $\tau_\mathbf{p}$ using the projection $\mathcal{P}_\mathbf{p}$ discussed in Section IV-A. The resulting (un-normalized) normal distribution, $\mathbf{n_p}(u, v)$, on the tangent plane can be expressed using equation (4) as:

$$\mathbf{n_p}(u, v) \approx \mathbf{N_p}(0, 0) + s(u, v)\, \mathbf{dN_p}(\hat{\mathbf{t}}(u, v)) \tag{7}$$

The tangent $\hat{\mathbf{t}}(u,v)$ and the arc-length $s(u,v)$ terms can be expanded as follows:

$$\hat{\mathbf{t}}(u,v) = \frac{(u\ \hat{\mathbf{u}}_\mathbf{p} + v\ \hat{\mathbf{v}}_\mathbf{p})}{\sqrt{u^2+v^2}} \tag{8}$$

$$s(u,v) = \frac{\arcsin(\lambda_\mathbf{p}(\hat{\mathbf{t}}(u,v))\sqrt{u^2+v^2})}{\lambda_\mathbf{p}(\hat{\mathbf{t}}(u,v))} \tag{9}$$

Using substitutions from equations (1), (2), (8), and (9) in equation (7) we get:

$$\mathbf{n}_\mathbf{p}(u,v) \approx \mathbf{N}_\mathbf{p}(0,0) - \left[\frac{(\lambda_{\mathbf{u}_\mathbf{p}}\ u\ \hat{\mathbf{u}}_\mathbf{p} + \lambda_{\mathbf{v}_\mathbf{p}}\ v\ \hat{\mathbf{v}}_\mathbf{p})}{(\lambda_{\mathbf{u}_\mathbf{p}}u^2 + \lambda_{\mathbf{v}_\mathbf{p}}v^2)/\sqrt{u^2+v^2}} \arcsin\left(\frac{\lambda_{\mathbf{u}_\mathbf{p}}u^2 + \lambda_{\mathbf{v}_\mathbf{p}}v^2}{\sqrt{u^2+v^2}}\right)\right]$$

It 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:

$$\mathbf{n}_\mathbf{p}(u,v) \approx \hat{\mathbf{e}}_z - \left[\frac{(\lambda_{\mathbf{u}_\mathbf{p}}\ u\ \hat{\mathbf{e}}_x + \lambda_{\mathbf{v}_\mathbf{p}}\ v\ \hat{\mathbf{e}}_y)}{(\lambda_{\mathbf{u}_\mathbf{p}}u^2 + \lambda_{\mathbf{v}_\mathbf{p}}v^2)/\sqrt{u^2+v^2}} \arcsin\left(\frac{\lambda_{\mathbf{u}_\mathbf{p}}u^2 + \lambda_{\mathbf{v}_\mathbf{p}}v^2}{\sqrt{u^2+v^2}}\right)\right] \tag{10}$$

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 $\Re^3$. Note that when specified in the local coordinate frame, the normal distribution is independent of $\hat{\mathbf{u}}_\mathbf{p}$, $\hat{\mathbf{v}}_\mathbf{p}$, and $\mathbf{N}_\mathbf{p}(\mathbf{0},\mathbf{0})$. So, the rendering algorithm uses a local coordinate system for the shading so that the normal distribution can be computed for one combination of $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$ and the same distribution is re-used to render all DPs with that combination of principal curvatures.

To shade a DP on a per-pixel basis we would want the normal distribution to be available at the screen space. The only hardware support to specify such a normal distribution is normal-mapping. It would be very expensive to compute such a normal map at run-time for each combination of $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$. So a normal map can be pre-computed in the local coordinate frame for various quantized values of the principal curvatures $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$, and at run time $\mathbf{r}_\mathbf{p}$ can be normal-mapped by the closest resembling normal map. However, a drawback to such a scheme of quantization 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, $\rho_\mathbf{p} = \frac{\lambda_{\mathbf{v}_\mathbf{p}}}{\lambda_{\mathbf{u}_\mathbf{p}}}$, and note that $-1 \leq \rho_\mathbf{p} \leq 1$ because $|\lambda_{\mathbf{u}_\mathbf{p}}| \geq |\lambda_{\mathbf{v}_\mathbf{p}}|$. The local normal distribution from equation(10) can be rewritten using $\rho_\mathbf{p}$ as:

$$\mathbf{n}_\mathbf{p}(u,v) \approx \hat{\mathbf{e}}_z - (u\ \hat{\mathbf{e}}_x + \rho_\mathbf{p}\ v\ \hat{\mathbf{e}}_y)\frac{\arcsin(\lambda_{\mathbf{u}_\mathbf{p}}\psi_\mathbf{p}(u,v))}{\psi_\mathbf{p}(u,v)} \tag{11}$$

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 $\lambda_{\mathbf{u}_\mathbf{m}} = 1$:

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

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, $\mathbf{r_m}$ is normal-mapped 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 (11)) can be re-written as:

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

Using *Lemma 1*, a general $\mathbf{r_p}$ is normal-mapped with an appropriate normal map $\mathbf{n_m}(\cdot,\cdot)$ with a scaling factor of $\lambda_{\mathbf{u_p}}$.

B. Shading

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 [40] functionality offered in the nVIDIA GeForce series of GPUs which allows one 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 $\mathbf{H_p}(u,v)$. Similarly, let $\hat{\mathbf{l}}_\mathbf{p}$ denote the (normalized) light vector at $\mathbf{x_p}$ and let $\mathbf{l_p}(u,v)$ and $\mathbf{L_p}(u,v)$ denote the (un-normalized) light vector distribution on $\tau_\mathbf{p}$ and $\mathbf{S_p}$ respectively with

$\mathbf{L_p}(0,0) = \hat{\mathbf{l}}_\mathbf{p}$. Also, let $\hat{\mathbf{w}}_\mathbf{p}$ denote the (normalized) view vector at $\mathbf{x_p}$ and let $\mathbf{w_p}(u,v)$ and $\mathbf{W_p}(u,v)$ denote the (un-normalized) view vector distribution on $\tau_\mathbf{p}$ and $\mathbf{S_p}$ respectively with $\mathbf{W_p}(0,0) = \hat{\mathbf{w}}_\mathbf{p}$.

Applying a method similar to the one used for the derivation of equation (7), $\mathbf{h_p}(u,v)$ can be written as:

$$\begin{aligned} \mathbf{h_p}(u,v) &\approx \mathbf{H_p}(0,0) + s(u,v)\,\mathbf{dH_p}(\hat{\mathbf{t}}(u,v)) \\ &\approx \mathbf{H_p}(0,0) + \sqrt{u^2+v^2}\,\mathbf{dH_p}(\hat{\mathbf{t}}(u,v)) \end{aligned} \tag{12}$$

where $\sqrt{u^2+v^2}$ is substituted as an approximation for $s(u,v)$. The half vector gradient can be expanded as:

$$\mathbf{dH_p}(\hat{\mathbf{t}}(u,v)) = \frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} \frac{u}{\sqrt{u^2+v^2}} + \frac{\partial}{\partial v}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} \frac{v}{\sqrt{u^2+v^2}}$$

and using this result, equation (12) can be re-written as:

$$\mathbf{h_p}(u,v) \approx \mathbf{H_p}(0,0) + u\frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} + v\frac{\partial}{\partial v}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} \tag{13}$$

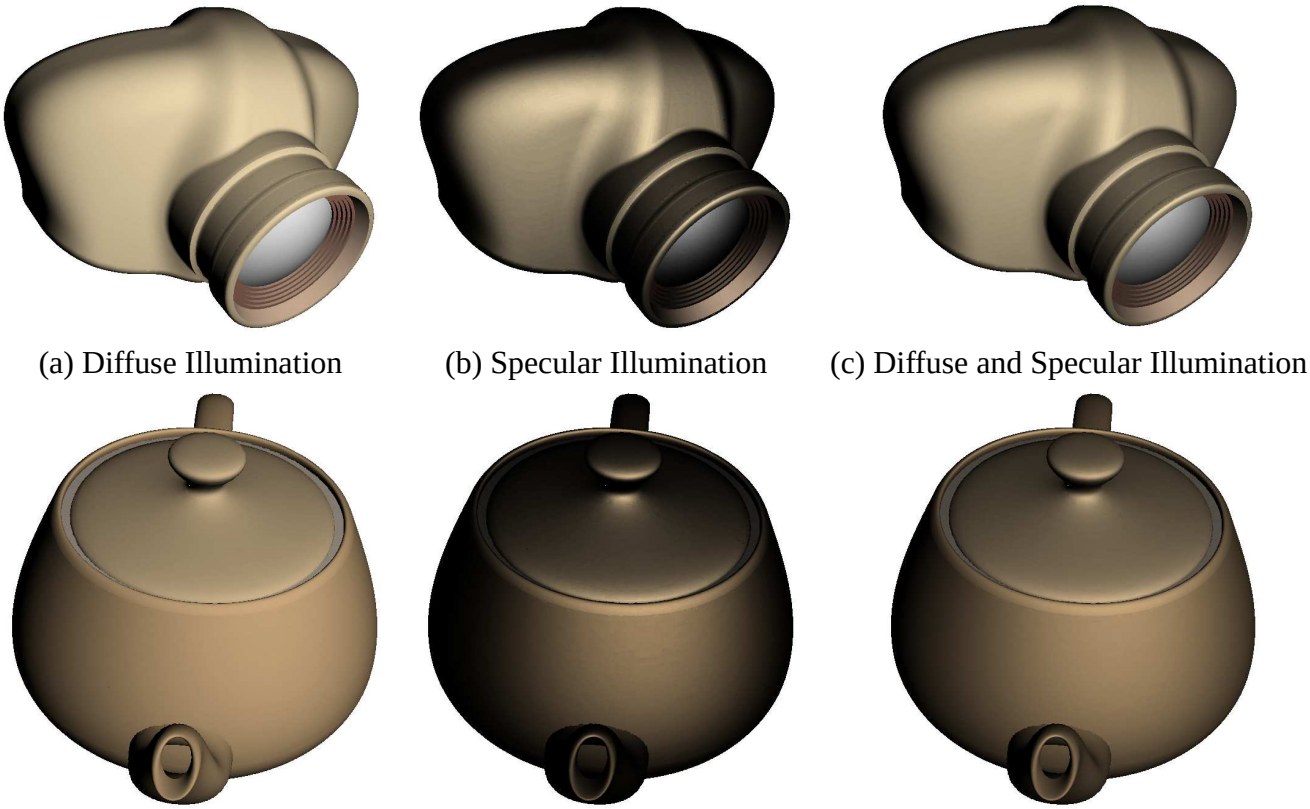
Let $\mathbf{a}$ be the position of the light and $\mathbf{b}$ be the position of the eye. The partial differential term of equation (13) can then be re-written as follows:

$$\begin{aligned} \frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} &= \frac{\partial}{\partial u}\left(\frac{\mathbf{L_p}(u,v)}{\|\mathbf{L_p}(u,v)\|} + \frac{\mathbf{W_p}(u,v)}{\|\mathbf{W_p}(u,v)\|}\right)\Big|_{\substack{u=0\\v=0}} \\ &= \frac{\partial}{\partial u}\Big(\frac{(\mathbf{a}-\mathbf{X_p}(u,v))}{\|\mathbf{a}-\mathbf{X_p}(u,v))\|}\Big)\Big|_{\substack{u=0\\v=0}} + \frac{\partial}{\partial u}\Big(\frac{(\mathbf{b}-\mathbf{X_p}(u,v))}{\|\mathbf{b}-\mathbf{X_p}(u,v))\|}\Big)\Big|_{\substack{u=0\\v=0}} \\ &= \frac{\big(\big(\frac{\mathbf{L_p}(u,v)}{\|\mathbf{L_p}(u,v)\|}\cdot\frac{\partial\mathbf{X_p}(u,v)}{\partial u}\big)\frac{\mathbf{L_p}(u,v)}{\|\mathbf{L_p}(u,v)\|} - \frac{\partial\mathbf{X_p}(u,v)}{\partial u}\big)\big|_{\substack{u=0\\v=0}}}{\|\mathbf{a}-\mathbf{x_p}\|} \\ &+ \frac{\big(\big(\frac{\mathbf{W_p}(u,v)}{\|\mathbf{W_p}(u,v)\|}\cdot\frac{\partial\mathbf{X_p}(u,v)}{\partial u}\big)\frac{\mathbf{W_p}(u,v)}{\|\mathbf{W_p}(u,v)\|} - \frac{\partial\mathbf{X_p}(u,v)}{\partial u}\big)\big|_{\substack{u=0\\v=0}}}{\|\mathbf{b}-\mathbf{x_p}\|} \\ &= \frac{((\hat{\mathbf{l}}_\mathbf{p}\cdot\hat{\mathbf{u}}_\mathbf{p})\hat{\mathbf{l}}_\mathbf{p} - \hat{\mathbf{u}}_\mathbf{p})}{\|\mathbf{a}-\mathbf{x_p}\|} + \frac{((\hat{\mathbf{w}}_\mathbf{p}\cdot\hat{\mathbf{u}}_\mathbf{p})\hat{\mathbf{w}}_\mathbf{p} - \hat{\mathbf{u}}_\mathbf{p})}{\|\mathbf{b}-\mathbf{x_p}\|} \end{aligned}$$

When expressed in the local coordinate frame, we get:

$$\frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} = \frac{((\hat{\mathbf{l}}_\mathbf{p}\cdot\hat{\mathbf{e}}_x)\hat{\mathbf{l}}_\mathbf{p} - \hat{\mathbf{e}}_x)}{\|\mathbf{a}-\mathbf{x_p}\|} + \frac{((\hat{\mathbf{w}}_\mathbf{p}\cdot\hat{\mathbf{e}}_x)\hat{\mathbf{w}}_\mathbf{p} - \hat{\mathbf{e}}_x)}{\|\mathbf{b}-\mathbf{x_p}\|} \tag{14}$$

the other partial differential term of equation (13) can be computed similarly to get a tangent plane half-vector distribution. The subtraction and the dot products in equation (14) are simple operations



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

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

Fig. 7. Illumination and per-pixel Shading

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 [41] and in practice, we have found that this approximation causes no compromise in visual quality.

The light vector distribution on $\tau_{\mathbf{p}}$ can be derived similarly. It is given by:

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

So far, we have discussed the tangent plane normal, half vector, and the light vector distribution around $\mathbf{p}$. They are used for shading $\mathbf{p}$. The overall rendering algorithm is given in Figure 8.

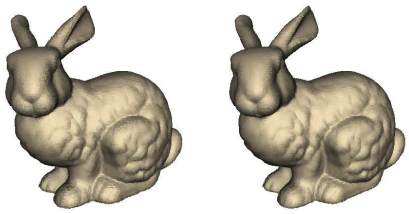
Shading $\mathbf{p}$ essentially involves two kinds of computation: (1) computing the relevant vectors (coordinates) for texture mapping (CPU-end computation) and (2) per-pixel computation (GPU-end computation). The rectangle $\mathbf{r}_{\mathbf{p}}$ is mapped by two textures: the normal map and the 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 $\mathbf{r}_{\mathbf{p}}$. Half-vector mapping involves computing the un-normalized half vectors at the vertices of $\mathbf{r}_{\mathbf{p}}$ using

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

Fig. 8. The Rendering Algorithm

equation (13) 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 liner interpolation between the half vectors specified at the vertices of $\mathbf{r_p}$. Per-pixel shading is achieved at the hardware register combiners level using the (per-pixel) normal and half vectors [40]. 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) 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 light source, with the accumulation buffer being used to buffer intermediate results.



(a) (b)

Fig. 9. Rendering quality with and without encoding. (a) The bunny rendered without encoding, (b) The bunny rendered with encoding.

VI. Implementation and Results

All the test cases were run on PC with a 866MHz Pentium 3 processor with 512MB RDRAM having a nVIDIA GeForce2 GTS GPU supported by 32MB of DDR RAM. All the test windows were 800×600 in size. We used 256 normal maps ($|\mathcal{M}| = 256$) corresponding to uniformly sampled values of ρ and we built a linear mip-map on each of these with the highest resolution being 32×32. The resolution of the cube vector map was $512\times512\times6$.

We demonstrate our work 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 patches are sampled uniformly in the parametric domain and simplified independent of each other. ϵ is the main parameter of the sampling process. 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. δ also ensures that the rectangles do not overrun the boundary significantly. We used a simple binary heap for heap operations of 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 I. 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.

Each DP uses 62 bytes of storage without any encoding and about 13 bytes of storage with encoding. Following is the data stored per DP without any encoding: 6 bytes for the diffuse and specular colors,

TABLE I
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

Without Simplification	*Teapot*	*Camera*	*Head*	*Bunny*	*Venus*
Number of Points	156,800	216,712	376,400	34,834	134,359
Storage Space w/o encoding (in MB)	9.19	12.69	22.06	1.77	6.82
Storage 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
Storage Space w/o encoding (in MB)	1.51	2.70	3.75	1.75	4.64
Storage 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

12 floats (48 bytes) for the point location, principal directions and the normal, and 2 floats (8 bytes) for the two curvature values. A simple scheme of encoding is used to represent a DP with just 13 bytes: 3 shorts (6 bytes) for the position $\mathbf{x_p}$, 2 bytes for each of the principal directions $\hat{\mathbf{u}}_\mathbf{p}$ and $\hat{\mathbf{v}}_\mathbf{p}$, 2 bytes for the maximum principal curvature $\lambda_{\mathbf{u_p}}$ and 1 byte for the value $\rho_\mathbf{p}$. The normal $\hat{\mathbf{n}}_\mathbf{p}$ need not be stored explicitly as it can be computed as a cross product of the principal directions. If the DPs were sampled from a triangle mesh then, as explained in section IV-B, some of them would have a size bigger than what is prescribed by the curvatures. This additional information is encoded in one or two bytes depending on the nature of the point. If the size of the rectangle $\mathbf{r_p}$ can be computed solely from the curvature values (as in section IV-A) 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 non-zero). They are written to the file after the first 13 bytes of the DP have been written. 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. Figure (9) shows the bunny rendered with and without encoding.

The results reported in Table I are with dynamic illumination (the light and half vectors are computed for each DP in each frame). Both the specular and diffuse shading are done at the hardware level. However, nVIDIA GeForce2 does not support a hardware implementation for the accumulation buffer. Instead, the accumulation buffer in implemented in software by the OpenGL drivers. So the case of both

diffuse and specular illumination can be slow. Hardware support for the accumulation buffer is available on other GPUs such as Voodoo5 6000 AGP from 3Dfx. However, this should not be a problem with the nVIDIA GeForce3 GPU, as they can allow access to 4 textures at the register combiners making it possible to compute both diffuse and specular shading in one straight pass.

On an average, about $330,000$ DPs can be rendered 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 main focus of this paper is the efficiency of DPs as rendering primitives. Previous works on point sample rendering have orthogonal benefits such as faster transformation [6] and multiresolution [4], [5], [8]. Potentially, DPs can be used in conjunction with these schemes. To demonstrate the benefits of DPs we designed experiments to compare performance against the splatting approach to rendering. A naive OpenGL point rendering was not considered because it is prone to holes and aliasing problems. We compare the rendering performance of an unsimplified differential point representation of a teapot to the splatting of unsimplified and unstructured versions of sampled points. For the splatting test cases, we take the original point samples from which DPs were constructed and associate each of them with a bounding ball whose radius is determined by comparing its distance from its sampled neighbors. This makes sure that there are no holes in surface coverage by the splatting primitives. We consider three kinds of test rendering primitives for splatting:

1. **Square Primitive**: They are squares parallel to the view plane with a width equal to the radius of the bounding sphere [5]. 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 [4]. The rectangle primitives are rendered with Z-buffering but without any blending.

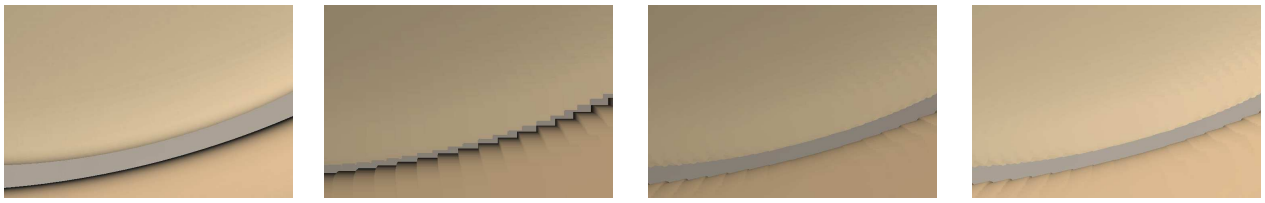
TABLE II
COMPARISON WITH SPLATTING PRIMITIVES: (*Test 1*) SAME NUMBER OF RENDERING PRIMITIVES, (*Test 2*) APPROXIMATELY SIMILAR RENDERING QUALITY, (*Test 3*) SIMILAR FRAME RATES. DP = DIFFERENTIAL POINTS, SP = SQUARE PRIMITIVE, RP = RECTANGLE PRIMITIVE, AND EP = ELLIPTICAL PRIMITIVE.

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

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, the rectangle primitive is texture mapped with a scaled version of the closest approximation of its ellipsoid. The texture-mapped rectangles are then rendered with a small depth offset and blending [5]. This is implemented in hardware using the register combiners.

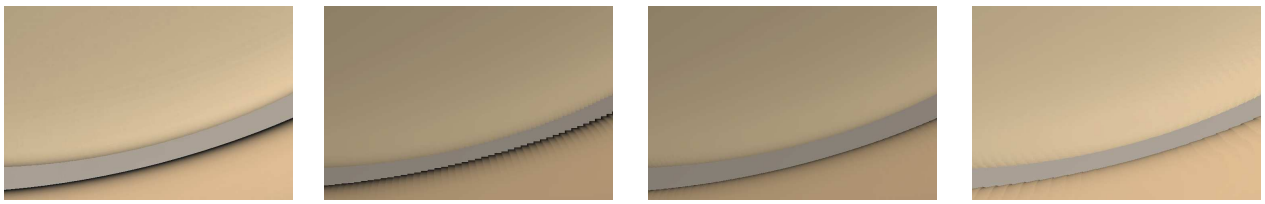
DPs were compared with the splatting primitives for three test cases: (1) same number of rendering primitives, (2) approximately similar visual quality of rendering, and (3) same frame rates. Table II summarizes the results of the first test case. DPs were found to deliver a much better rendering quality for the same number of primitives as seen in Figure 10. 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.

Table II also summarizes the results of the second test case. Sample renderings for this test are shown in Figure 10. For this case the number of square, rectangle, and elliptical primitives were increased by increasing the sampling frequency of the uniformly sampled model used for DPs. In this test case, DPs clearly out-performed the splatting primitives both in frame rates and in the storage space requirements. The third test case shows that for the same frame rates DPs produced better rendering quality using



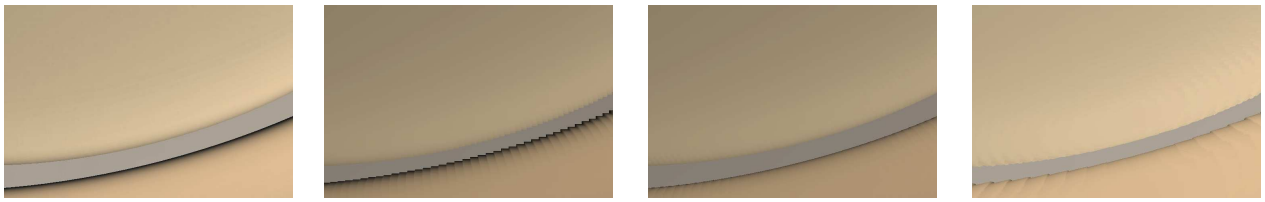
(a) Differential Points (2.13 fps) (b) Square Primitive (11.76 fps) (c) Rectangle Primitive (10.52 fps) (d) Elliptical Primitive (2.35 fps)

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



(a) Differential Points (157K pts., 2.13fps) (b) Square Primitive (1411K pts., 1.61 fps) (c) Rectangle Primitive (1155K pts., 1.49 fps) (d) Elliptical Primitive (320K pts., 1.16 fps)

Test 2: Comparison of primitives for similar rendering quality



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

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

Fig. 10. Selected areas of rendering of the teapot model for the three test cases

fewer rendering primitives.

VII. Conclusions and Future Work

The results and the test comparisons clearly demonstrate the efficiency of DPs as rendering primitives. The ease of simplification gives DPs an added advantage to get a significant speed up. The high quality of rendering is attributed to the inherent ability of DPs to well approximate the local surface normal distribution. The rendering efficiency of DPs is attributed to the sparse surface representation that reduces bus bandwidth and its amenability to per-pixel shading.

One shortcoming of DPs is that the complexity of the borders limit the maximum width of the interior DPs through the δ constraint. This leads to increased sampling in the interior even though these DPs have enough room to expand within the bounds laid down by the ϵ constraint. A width-determination approach that uses third-order differential information (such as the variation of the surface curvature)

should be able to deal with this more efficiently.

A multiresolution scheme of DPs can be explored that will efficiently render lower frequency versions of the original surface. Under this scheme, a rendering algorithm that blends the DPs is a promising prospect. Another line of future work is with regards to simplification. Currently, we using a simple heuristic with some approximation algorithms which do not guarantee a hole-free representation. There is a lot of scope for improvement here. Compression of point samples is also a promising prospect.

VIII. Acknowledgements

We would like to thank our colleagues at the Graphics and Visual Informatics Laboratory at the University of Maryland, College Park and at the Center for Visual Computing at SUNY, Stony Brook. We would like to thank Robert McNeel & Associates for the head model, the camera model, and for the openNURBS code. Also thanks to the Stanford Graphics Laboratory for the bunny model, and Cyberware Inc. for the venus model. Last, but not the least, we would like to acknowledge NSF funding grants IIS00-81847, ACR-98-12572 and DMI-98-00690.

References

[1] J. P. Grossman and William J. Dally, "Point sample rendering," in *Rendering Techniques '98*, G. Drettakis and N. Max, Eds. 1998, Eurographics, pp. 181–192, Springer-Verlag Wien New York.

[2] M. Levoy and T. Whitted, "The use of points as a display primitive," in *Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill*, Jan. 1985.

[3] D. Lischinski and A. Rappoport, "Image-based rendering for non-diffuse synthetic scenes," in *Rendering Techniques '98*, G. Drettakis and N. Max, Eds. 1998, Eurographics, pp. 301–314, Springer-Verlag Wien New York.

[4] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proceedings of SIGGRAPH 2000*, July 2000, pp. 335–342.

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

[6] J. Shade, S. Gortler, L. He, and R. Szeliski, "Layered depth images," in *Proceedings of SIGGRAPH 98*, Aug. 1998, pp. 231–242.

[7] S. Rusinkiewicz and M. Levoy, "Streaming QSplat: A viewer for networked visualization of large, dense models," in *Symposium of Interactive 3D Graphics*, Mar. 2001, pp. 63–68.

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

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

[10] A. Hilton, J. Illingworth, and T. Windeatt, "Statistics of surface curvature estimates," *Pattern Recognition*, vol. 28, no. 8, pp. 1201–1222, 1995.

[11] 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*, vol. 8, pp. 833–840, August 1992.

[12] G. Taubin, "Estimating the tensor of curvature of a surface from a polyhedral approximation," in *Fifth International Conference on Computer Vision*, 1995, pp. 902–907.

[13] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Discrete differential-geometry operations in *n*D," *http://www.multires.caltech.edu/pubs/DiffGeoOperators.pdf*, 2000.

[14] 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*, 1999, pp. 34–43.

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

[16] 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*, July 2000, pp. 131–144.

[17] P. Rademacher and G. Bishop, "Multiple-center-of-projection images," in *Proceedings of SIGGRAPH 98*, Aug. 1998, pp. 199–206.

[18] H. Rushmeier, G. Taubin, and A. Guéziec, "Applying shape from lighting variation to bump map capture," in *Rendering Techniques'97*, Julie Dorsey and Philipp Slusallek, Eds. June 1997, pp. 35–44, Springer-Verlag Wien New York.

[19] C. L. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3D scans," in *Proceedings of SIGGRAPH 95*, Aug. 1995, pp. 109–118.

[20] 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*, vol. 5, no. 4, pp. 349–359, Oct./Dec. 1999.

[21] I. Guskov, W. Sweldens, and P. Schröder, "Multiresolution signal processing for meshes," in *Proceedings of SIGGRAPH 99*, 1999, pp. 325–334.

[22] M. Pauly and M. Gross, "Spectral processing of point-sampled geometry," in *Proceedings of SIGGRAPH'01*, Aug. 2001, pp. 379–386.

[23] M. Eck and H. Hoppe, "Automatic reconstruction of B-spline surfaces of arbitrary topological type," in *Proceedings of SIGGRAPH'96*, Aug. 1996, pp. 325–334.

[24] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," in *Proceedings of SIGGRAPH 96*, Aug. 1996, pp. 313–324.

[25] G. Turk, "Re-tiling polygonal surfaces," in *Proceedings of SIGGRAPH 92*, July 1992, pp. 55–64.

[26] A. P. Witkin and P. S. Heckbert, "Using particles to sample and control implicit surfaces," in *Proceedings of SIGGRAPH 94*, July 1994, pp. 269–278.

[27] P. Cignoni, C. Montani, and R. Scopigno, "A comparison of mesh simplification algorithms," *Computers & Graphics*, vol. 22, no. 1, pp. 37–54, 1998.

[28] 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.

[29] P. Lindstrom, "Out-of-core simplification of large polygonal models," in *Proceedings of SIGGRAPH 2000*, July 2000, pp. 259–262.

[30] L. Darsa, B. C. Silva, and A. Varshney, "Navigating static environments using image-space simplification and morphing," in *1997 Symposium on Interactive 3D Graphics*, Michael Cohen and David Zeltzer, Eds., Apr. 1997, pp. 25–34.

[31] W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3D warping," in *1997 Symposium on Interactive 3D Graphics*, Apr. 1997, pp. 7–16.

[32] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle, "View-based rendering: Visualizing real objects from scanned range and color data," in *Rendering Techniques'97*, Julie Dorsey and Philipp Slusallek, Eds. June 1997, pp. 23–34, Springer-Verlag Wien New York.

[33] M. M. Oliveira, G. Bishop, and D. McAllister, "Relief texture mapping," in *Proceedings of SIGGRAPH 2000*, July 2000, pp. 359–368.

[34] B. Chen and M. X. Nguyen, "Pop: A polygon hybrid point and polygon rendering system for large data," in *IEEE Visualization'01*, Oct. 2001.

[35] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, C. Silva, and D. Levin, "Point set surfaces," in *IEEE Visualization 2001*, Oct. 2001.

[36] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Surface splatting," in *Proceedings of SIGGRAPH 2001*, Aug. 2001, pp. 371–378.

[37] Klaus Mueller, Torsten Mller, and Roger Crawfis, "Splatting without the blur," in *IEEE Visualization'99*, Oct. 1999, pp. 363–370.

[38] V. L. Interrante, "Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution," in *Proceedings of SIGGRAPH 97*, Aug. 1997, pp. 109–116.

[39] P. S. Heckbert and M. Garland, "Optimal triangulation and quadric-based surface simplification," *Computational Geometry*, vol. 14, pp. 49–65, 1999.

[40] 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).

[41] K. Turkowski, "Computing the inverse square root," in *Graphics Gems*, A. Paeth, Ed. 1995, vol. 5, pp. 16–21, Academic Press.

Aravind Kalaiah is a Doctoral Candidate at the Computer Science Department of the University of Maryland, College Park. He received a B.Tech. in Computer Science from the Indian Institute of Technology, Bombay in 1998 and a M.S. in Computer Science from the State University of New York at Stony Brook in 2000. His primary research interests are in real-time visualization, geometric modeling and photorealism.

Amitabh Varshney is currently an Associate Professor of Computer Science at the University of Maryland at College Park. He received his B.Tech. in Computer Science from the Indian Institute of Technology, Delhi in 1989 and his M.S. and Ph.D. degrees from the University of North Carolina at Chapel Hill in 1991 and 1994. He was an Assistant Professor of Computer Science at the State University of New York at Stony Brook from 1994 to 2000. His current research interests are in interactive 3D graphics, scientific visualization, geometric modeling, and molecular graphics. He received the National Science Foundation's CAREER award in 1995.