Differential Point Rendering

Aravind Kalaiah Amitabh Varshney
University of Maryland[1]

Abstract. We present a novel point rendering primitive, called *Differential Point* (DP), that captures the local differential geometry in the vicinity of a sampled 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 neighborhoods. The advantages to this representation are manyfold: (1) it delivers a significant reduction in the number of point primitives that represent a surface (2) it achieves robust hardware accelerated per-pixel shading – even with no connectivity information (3) it offers a novel point-based simplification technique that has a convenient and intuitive interface for the user to efficiently resolve the speed versus quality tradeoff. 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.

1 Introduction

Point-based rendering schemes [5, 12, 13, 17, 19, 21] have evolved as a viable alternative to triangle-based representations. They offer many benefits over triangle-based models: (1) efficiency in modeling and rendering complex environments, (2) hierarchical organization to efficiently control frame-rates and visual quality, and (3) zero-connectivity for efficient streaming for remote rendering [20].

Current point primitives store only limited information about their immediate locality, such as normal, bounding ball [19], and tangent plane disk [17]. These primitives are then rasterized with flat shading and hence such representations require very high sampling to obtain a good rendering quality. In other words, the rendering algorithm dictates the sampling frequency of 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 speed due to higher 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 extent of the approximated vicinity is determined by the curvature characteristics of the surface: points in a flat or a low curvature region approximate larger vicinities. Our approach has many benefits to offer:

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

1. **Rendering**: Our approach achieves high rendering quality with per-pixel shading. We reduce the number of rendering primitives and push 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 a focus on the efficiency of per-point rendering computation. It can potentially be used in conjunction with larger point-based organization structures - hierarchical [1, 17, 19] or otherwise [13, 21].

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 differential points. 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 potential extensions.

2 Related Work

Classical differential geometry gives us a mathematical model for understanding the surface variation at a point. Surface curvatures can be accurately computed for parametric surfaces and can also be estimated from discrete sampled representations.Taubin [22] estimates curvature at a mesh vertex by using the eigenvalues of an approximation matrix constructed using the incident edges. Desbrun et al. [3] define discrete operators (normal, curvature, etc.) of differential geometry using Voronoi cells and finite-element/finite-volume methods.

Various hierarchical organization schemes have been used that develop lower frequency versions of the original set of point samples [1, 17, 19]. We use a simplification process that prunes an initial set of points to greatly reduce the redundancy in surface representation. Turk [23] uses a point placement approach with the point density being controlled by the local curvature properties of the surface. Witkin and Heckbert [24] use physical properties of a particle system to place points on an implicit surface.

Image-assisted organization of points [5, 13, 21] are efficient at three-dimensional transformations as they use the implicitness of relationship among pixels to achieve fast incremental computations. They are also efficient at representing complex real-world environments. The multiresolution organizations [1, 17, 19] use their hierarchical structure to achieve block culling, to control depth traversals to respect the image-quality or frame-rate constraints, and for efficient streaming of large datasets across the network [20].

Darsa et al. [2], Mark et al. [14] and Pulli et al. [18] 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 [12] introduced points as a rendering primitive. It has been used by Shade et al. [21] and Grossman and Dally [5] for synthetic environments. However, raw point primitives suffer from aliasing problems and holes. Lischinski and Rappoport [13] raytrace a point dataset. Oliveira et al. [15] use

image space transformations to render point samples. Rusinkiewicz and Levoy [19] use splatting of polygonal primitives for rendering. Pfister et al. [17] follow up the splatting with a screen-space filtering process to handle holes and aliasing problems. Zwicker et al. [25] derive a screen-space formulation of the EWA filter to render high detail textured point samples with a support for transparency.

3 Differential Geometry for Surface Representation

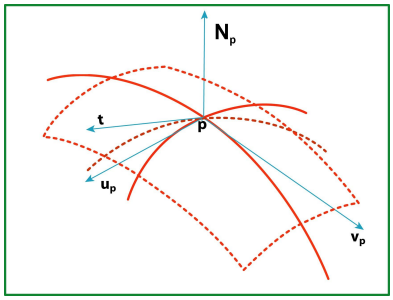


Fig. 1. Neighborhood of a Differential Point

Classical differential geometry is a study of the local properties of curves and surfaces [4]. 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. 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. Every point $\mathbf{p}$ on the regular surface has the following properties: the normal $\mathbf{N_p}$, the *direction of maximum curvature* $\hat{\mathbf{u}}_\mathbf{p}$, the *direction of minimum curvature* $\hat{\mathbf{v}}_\mathbf{p}$, the *maximum normal curvature* $\lambda_{\mathbf{u_p}}$, and the *minimum normal curvature* $\lambda_{\mathbf{v_p}}$. They have the following relation amongst them: (1) $|\lambda_{\mathbf{u_p}}| \geq |\lambda_{\mathbf{v_p}}|$, (2) $\langle \hat{\mathbf{u}}_\mathbf{p}, \hat{\mathbf{v}}_\mathbf{p} \rangle = 0$, and (3) $\hat{\mathbf{u}}_\mathbf{p} \times \hat{\mathbf{v}}_\mathbf{p} = \mathbf{N_p}$ where $\langle \cdot, \cdot \rangle$ denotes the dot product and $\times$ is the cross product operator. The differential of the normal at $\mathbf{p}$ is denoted by $\mathbf{dN_p}$. Given any tangent $\hat{\mathbf{t}}$ (= $t_\mathbf{u}\hat{\mathbf{u}}_\mathbf{p} + t_\mathbf{v}\hat{\mathbf{v}}_\mathbf{p}$), $\mathbf{dN_p}(\hat{\mathbf{t}})$ is the first-order normal variation around $\mathbf{p}$ and is given by [4]:

$$\mathbf{dN_p}(\hat{\mathbf{t}}) \quad = \quad -(\lambda_{\mathbf{u_p}} t_\mathbf{u} \hat{\mathbf{u}}_\mathbf{p} + \lambda_{\mathbf{v_p}} t_\mathbf{v} \hat{\mathbf{v}}_\mathbf{p}) \tag{1}$$

Similarly the normal curvature along $\hat{\mathbf{t}}$, $\lambda_\mathbf{p}(\hat{\mathbf{t}})$, is given by [4]:

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

The normal variation and the normal curvature terms give us second order information about the behaviour of the regular surface around the point $\mathbf{p}$. These properties are illustrated in Figure 1.

A salient feature of the regular surface model is that it gives complete independence to a point to describe its own neighborhood without any reliance, explicit or implicit, on the immediate sampled neighborhood or on any other global property [6, 7, 8]. Even though a surface has to be differentiable to have a regular surface representation, 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 and even important. We sample at the second-order continuous neighborhood of these points. Discontinuities are maintained implicitly by the intersection of the region of influence of the adjacent sampled points.

4 Sampling and Processing

Our fundamental rendering primitive is a point with differential geometry information. These are derived by sampling points on the surface and extracting differential geometry information per sampled point. This is followed by a simplification process that greatly reduces the redundancy in surface representation. This is a pre-process and the output is saved in a render-ready format that will be an input to our rendering algorithm outlined in Section 5.

4.1 Differential Points

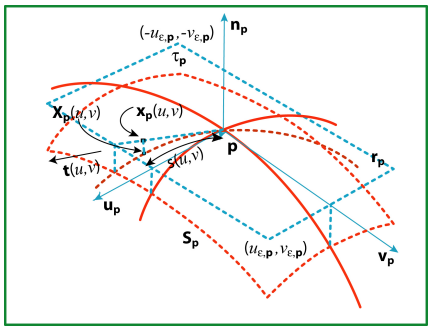


Fig. 2. Tangent plane parameterization of a Differential Point

We call our rendering primitive 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). They derive the unit normal, $\hat{\mathbf{n}}_\mathbf{p}$, and the tangent plane, $\tau_\mathbf{p}$, of $\mathbf{p}$. We extrapolate this information to define a surface, $\mathbf{S_p}$, that will be used to approximate the neighborhood of $\mathbf{x_p}$. The surface $\mathbf{S_p}$ is defined 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.

To aid our 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 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 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 $s(u, v)$.

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 an arc-length of $s(u, v)$, the normal at $\mathbf{X_p}(u, v)$ can be obtained by using a Taylor's expansion on each dimension to get:

$$\mathbf{N_p}(u, v) \approx \mathbf{N_p}(0, 0) + s(u, v)\ \mathbf{dN_p}(\hat{\mathbf{t}}(u, v)) \tag{3}$$

$\mathbf{S_p}$ and $\mathbf{N_p}(u, v)$ give an approximation of the spatial and the normal distribution around $\mathbf{x_p}$. (Note that $\mathbf{N_p}(u, v)$ is not the approximation of the normal of $\mathbf{S_p}$.) We use two criteria to determine the extent of the approximation:

1. *Maximum Principal Error* (ϵ): This specifies a curvature-scaled maximum orthographic deviation of $\mathbf{S_p}$ along the principal directions. We lay down this constraint as: $|\lambda_{\mathbf{u_p}}(\mathbf{X_p}(u,0) - \mathbf{x_p}(u,0))| \leq \epsilon \leq 1$ and $|\lambda_{\mathbf{v_p}}(\mathbf{X_p}(0,v) - \mathbf{x_p}(0,v))| \leq \epsilon \leq 1$. Note that since $\mathbf{S_p}$ is defined by semi-circles, we have that $\|\mathbf{X_p}(u,0) - \mathbf{x_p}(u,0)\| \leq \frac{1}{|\lambda_{\mathbf{u_p}}|}$. 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. 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.

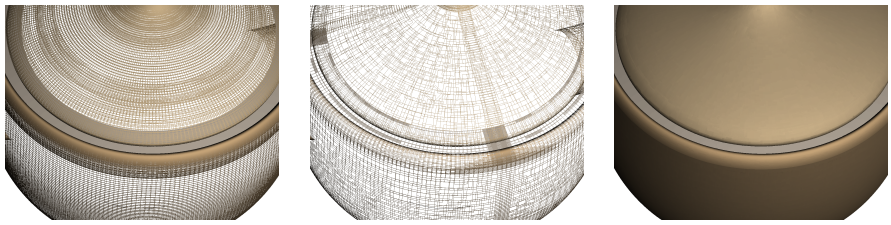
4.2 Sampling and Simplification

Given a 3D model, it is first sampled by points. Currently, we sample uniformly in the parametric domain of a NURBS surface and standard techniques outlined in the differential geometry literature [4] are used to extract the differential geometry information at each sampled point. This work can be extended to triangle meshes using ideas of Taubin [22] and Desbrun et al. [3]. Alternately, a NURBS surface can be fit into the triangle mesh [11] and the points can be sampled using this representation.

Initially the surface is super-sampled so that the rectangle of each differential point overlaps sufficiently with its neighbors without leaving holes in the surface coverage. This is followed by the simplification process which prunes the redundant points that are adequately represented by its neighbors. Simplification has two main stages: (1) ordering and (2) pruning. In the first stage, we compute the *redundancy factor* of each DP: the closer it resembles its neighbors in curvature values and directions, the higher the redundancy factor. All the DPs are then ordered in a priority heap with their redundancy factor as the key. In the second stage, we iteratively pop the top of the heap (the most redundant DP of the lot) and check if the rectangles $\mathbf{r_p}$ of its neighbors cover up the surface if it is pruned: if so we prune it and re-order the heap, otherwise we mark the DP to be saved. Figure 3 illustrates the effectiveness of the simplification process. Due to page limit constraints we are unable to discuss the details of our simplification algorithm here. Full details can be found in [9].

5 Rendering

Since graphics hardware do not support $\mathbf{S_p}$ as a rendering primitive, $\mathbf{r_p}$ is used as an approximation to $\mathbf{S_p}$ when rasterizing $\mathbf{p}$. However, the shading artifacts are more readily discernible to the human eye and 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) (b) (c)

Fig. 3. Effectiveness of Simplification: (a) Rectangles of the initial set of DPs representing the teapot. (b) Rectangles of the differential points that are retained by the simplification algorithm. Simplification is currently 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 model.

5.1 Normal Distribution

Consider the projection of $\mathbf{N_p}(u,v)$ onto $\tau_\mathbf{p}$ using the projection $\mathcal{P}_\mathbf{p}$ discussed in Section 4.1. The area of projection on the tangent plane is limited by the nature of the surface $\mathbf{S_p}$ and includes all (u,v) such that $\sqrt{u^2+v^2} \leq \frac{1}{|\lambda_\mathbf{p}(\hat{\mathbf{t}}(u,v))|}$. The (un-normalized) normal distribution, $\mathbf{n_p}(u,v)$, on $\tau_\mathbf{p}$ can then be expressed using equation (3) as:

$$\mathbf{n_p}(u,v) \quad \approx \quad \mathbf{N_p}(0,0) + s(u,v)\ \mathbf{dN_p}(\hat{\mathbf{t}}(u,v)) \tag{4}$$

where tangent $\hat{\mathbf{t}}(u,v) = \frac{(u\ \hat{\mathbf{u}}_\mathbf{p} + v\ \hat{\mathbf{v}}_\mathbf{p})}{\sqrt{u^2+v^2}}$ and arc-length $s(u,v) = \frac{\sin^{-1}(\lambda_\mathbf{p}(\hat{\mathbf{t}}(u,v))\sqrt{u^2+v^2})}{\lambda_\mathbf{p}(\hat{\mathbf{t}}(u,v))}$ (the range of the arcsin function being $[-\frac{\pi}{2}, \frac{\pi}{2}]$). Using these terms, and equations (1) and (2) as well, equation (4) can be re-written as:

$$\mathbf{n_p}(u,v) \approx \mathbf{N_p}(0,0) - \left[\frac{(\lambda_{\mathbf{u_p}}\ u\ \hat{\mathbf{u}}_\mathbf{p} + \lambda_{\mathbf{v_p}}\ v\ \hat{\mathbf{v}}_\mathbf{p})}{(\lambda_{\mathbf{u_p}}u^2 + \lambda_{\mathbf{v_p}}v^2)/\sqrt{u^2+v^2}} \sin^{-1}\left(\frac{\lambda_{\mathbf{u_p}}u^2 + \lambda_{\mathbf{v_p}}v^2}{\sqrt{u^2+v^2}}\right)\right]$$

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:

$$\mathbf{n_p}(u,v) \approx \hat{\mathbf{e}}_z - \left[\frac{(\lambda_{\mathbf{u_p}}\ u\ \hat{\mathbf{e}}_x + \lambda_{\mathbf{v_p}}\ v\ \hat{\mathbf{e}}_y)}{(\lambda_{\mathbf{u_p}}u^2 + \lambda_{\mathbf{v_p}}v^2)/\sqrt{u^2+v^2}} \sin^{-1}\left(\frac{\lambda_{\mathbf{u_p}}u^2 + \lambda_{\mathbf{v_p}}v^2}{\sqrt{u^2+v^2}}\right)\right] \tag{5}$$

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 $\mathbb{R}^3$. Note that $\mathbf{n_p}(u,v)$ is independent of $\hat{\mathbf{u}}_\mathbf{p}$, $\hat{\mathbf{v}}_\mathbf{p}$, and $\mathbf{N_p}(0,0)$ in the local coordinate frame. Hence DP is shaded in its local coordinate frame so that the normal distribution is computed for each combination of $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$ and reused for all DPs with that combination.

The only hardware support to specify such a normal distribution is normal mapping. We pre-computed normal-maps in the local coordinate frame for various quantized values of $\lambda_\mathbf{u}$ and $\lambda_\mathbf{v}$. However, 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 $|\lambda_{\mathbf{u_p}}| \geq |\lambda_{\mathbf{v_p}}|$. Equation (5) can be rewritten using $\rho_\mathbf{p}$ as follows:

$$\mathbf{n_p}(u,v) \quad \approx \quad \hat{\mathbf{e}}_z - (u\ \hat{\mathbf{e}}_x + \rho_\mathbf{p}\ v\ \hat{\mathbf{e}}_y)\frac{\sin^{-1}(\lambda_{\mathbf{u_p}}\psi_\mathbf{p}(u,v))}{\psi_\mathbf{p}(u,v)} \tag{6}$$

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 $\lambda_{\mathbf{u_m}} = 1$: 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 (6)) becomes:

$$\begin{aligned}
\mathbf{n_p}(u,v) &\approx \hat{\mathbf{e}}_z - \left[((\lambda_{\mathbf{u_p}} u)\hat{\mathbf{e}}_x + \rho_{\mathbf{p}}(\lambda_{\mathbf{u_p}} v)\hat{\mathbf{e}}_y) \, \frac{\sin^{-1}(\psi_{\mathbf{p}}(\lambda_{\mathbf{u_p}} u, \lambda_{\mathbf{u_p}} v))}{\psi_{\mathbf{p}}(\lambda_{\mathbf{u_p}} u, \lambda_{\mathbf{u_p}} v)} \right] \\
&= \hat{\mathbf{e}}_z - \left[((\lambda_{\mathbf{u_p}} u)\hat{\mathbf{e}}_x + \rho_{\mathbf{m}}(\lambda_{\mathbf{u_p}} v)\hat{\mathbf{e}}_y) \, \frac{\sin^{-1}(\psi_{\mathbf{m}}(\lambda_{\mathbf{u_p}} u, \lambda_{\mathbf{u_p}} v))}{\psi_{\mathbf{m}}(\lambda_{\mathbf{u_p}} u, \lambda_{\mathbf{u_p}} v)} \right] \\
&\approx \mathbf{n_m}(\lambda_{\mathbf{u_p}} u, \lambda_{\mathbf{u_p}} v) \qquad \square
\end{aligned}$$

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 $\lambda_{\mathbf{u_p}}$.

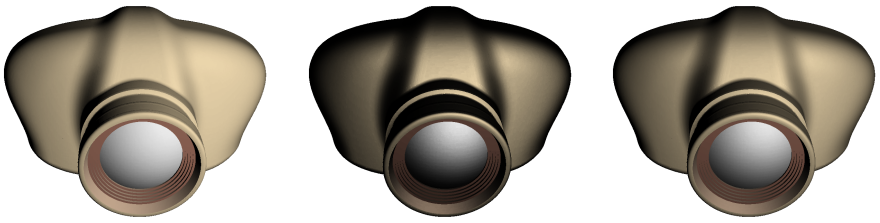
5.2 Shading

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 [10] functionality offered in the nVIDIA GeForce 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. Using this feature we obtain normalized half-vectors on a per-pixel basis by specifying un-normalized half vectors at the vertices of $\mathbf{r_p}$. Per-pixel shading is achieved by using the per-pixel normal (by normal map) and half or light vector (by cube vector map) for illumination computations in the register combiners.

Let $\hat{\mathbf{h}}_{\mathbf{p}}$ denote the (normalized) half vector at $\mathbf{x_p}$ and let $\mathbf{H_p}(u,v)$ denote the (un-normalized) half vector at $\mathbf{X_p}(u,v)$ where $\mathbf{H_p}(0,0) = \hat{\mathbf{h}}_{\mathbf{p}}$. Let $\mathbf{h_p}(u,v)$ be the (un-normalized) half vector at $\mathbf{x_p}(u,v)$ obtained by 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 $\mathbf{W_p}(0,0) = \hat{\mathbf{w}}_{\mathbf{p}}$. Similar to equation (4), $\mathbf{h_p}(u,v)$ can be written as:

$$\begin{aligned}
\mathbf{h_p}(u,v) &\approx \mathbf{H_p}(0,0) + s(u,v)\, \mathbf{dH_p}(\hat{\mathbf{t}}(u,v)) \\
&\approx \mathbf{H_p}(0,0) + \sqrt{u^2+v^2}\, \mathbf{dH_p}(\hat{\mathbf{t}}(u,v)) \\
&= \mathbf{H_p}(0,0) + u \frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} + v \frac{\partial}{\partial v}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} \qquad (7)
\end{aligned}$$

Let $\mathbf{a}$ be the position of the light and $\mathbf{b}$ be the position of the eye. The partial



(a) (b) (c)

Fig. 4. Illumination and per-pixel Shading: (a) Diffuse Illumination. (b) Specular Illumination. (c) Diffuse and Specular Illumination

differential of equation (7) can then be re-written as follows:

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

When expressed in the local coordinate frame, we get:

$$\frac{\partial}{\partial u}\mathbf{H_p}(u,v)\Big|_{\substack{u=0\\v=0}} = \frac{((\hat{\mathbf{l}}_\mathbf{p}\cdot\hat{\mathbf{e}}_x)\hat{\mathbf{l}}_\mathbf{p}-\hat{\mathbf{e}}_x)}{\|\mathbf{a}-\mathbf{x_p}\|}+\frac{((\hat{\mathbf{w}}_\mathbf{p}\cdot\hat{\mathbf{e}}_x)\hat{\mathbf{w}}_\mathbf{p}-\hat{\mathbf{e}}_x)}{\|\mathbf{b}-\mathbf{x_p}\|} \tag{8}$$

the other partial differential of equation (7) can be computed similarly. The subtraction and the dot products in equation (8) are simple and fast operations. The square root and the division operations are combined together by the fast inverse square root approximation [16] which in practice causes no compromise in the visual quality. The light vector distribution on $\tau_\mathbf{p}$ can be derived similarly and is given by $\mathbf{l_p}(u,v) \approx \hat{\mathbf{l}}_\mathbf{p} - u\hat{\mathbf{e}}_x - v\hat{\mathbf{e}}_y$.

The tangent plane normal, half vector, and the light vector distribution around $\mathbf{p}$ are used for shading $\mathbf{p}$ which essentially involves two kinds of computation: (1) mapping of the relevant vectors and (2) per-pixel computation. The overall rendering algorithm is given in Figure 5. The rectangle $\mathbf{r_p}$ is mapped by 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 $\mathcal{M}$. Half-vector mapping involves computing un-normalized half vectors at the vertices of $\mathbf{r_p}$ using equation (7) and using them as the texture coordinates of the cube vector map that is mapped onto $\mathbf{r_p}$. Per-pixel shading is done in the hardware register combiners using the (per-pixel) normal and half vectors [10]. 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. The accumulation buffer can be bypassed by disabling depth writes in the second pass, but it leads to multiple writes to a pixel due to rectangle overlaps and depth quantization which results in bright artifacts. If three textures are accessible at the combiners then both illuminations can be combined into one pass.

6 Implementation and Results

All the test cases were run on a 866MHz Pentium 3 PC with 512MB RDRAM and having a nVIDIA GeForce2 card supported by 32MB of DDR RAM. All the test win-

Display( )

(Compute $\mathcal{M}$ and load them into texture memory at the program start time)

1 Clear the depth and the color buffers
2 Configure the register combiners for diffuse shading
3 $\forall$ DP $\mathbf{p}$
4 $\quad\mathcal{M}_{\mathbf{p}}$ = normal-map $\in \mathcal{M}$ whose ρ is closest to $\rho_{\mathbf{p}}$
5 $\quad$Map $\mathcal{M}_{\mathbf{p}}$ onto $\mathbf{r_p}$
6 $\quad$Compute the light vector, $\mathbf{l_p}(\cdot,\cdot)$, at the vertices of $\mathbf{r_p}$
7 $\quad$Use the light vectors to map a cube vector map onto $\mathbf{r_p}$
8 $\quad$Render $\mathbf{r_p}$
9 Clear the color buffer after loading it into the accumulation buffer
10 Clear the depth buffer
11 Configure the register combiners for specular shading
12 $\forall$ DP $\mathbf{p}$
13 $\quad\mathcal{M}_{\mathbf{p}}$ = normal-map $\in \mathcal{M}$ whose ρ is closest to $\rho_{\mathbf{p}}$
14 $\quad$Map $\mathcal{M}_{\mathbf{p}}$ onto $\mathbf{r_p}$ (The details from the last pass can be cached)
15 $\quad$Compute the half vector, $\mathbf{h_p}(\cdot,\cdot)$, at the vertices of $\mathbf{r_p}$
16 $\quad$Use the half vectors to map a cube vector map onto $\mathbf{r_p}$
17 $\quad$Render $\mathbf{r_p}$
18 Add the accumulation buffer to the color buffer
19 Swap the front and the back color buffers

Fig. 5. The Rendering Algorithm

dows were 800×600 in size. We used 256 normal maps ($|\mathcal{M}|$=256) corresponding to uniformly sampled values of $\rho_{\mathbf{p}}$ and we built a linear mip-map on each of these with a highest resolution of 32×32. The resolution of the cube vector map was $512\times512\times6$.

We demonstrate our work on three models: the Utah Teapot, a Human Head model, and a Camera prototype. The models are in the NURBS representation. The component patches are sampled uniformly in the parametric domain and simplified independent of each other. The error threshold ϵ is the main parameter of the sampling process. δ ensures that the rectangles from the low curvature region do not block the nearby rectangles in the higher curvature regions and also ensures that the rectangles do not overrun the boundary significantly. Simplification can lead to an order-of-magnitude speed-up in rendering and can save substantial storage space as reported in Table 1. In our current representation, each DP uses 62 bytes of storage: 6 bytes for the diffuse and specular colors, 12 floats (48 bytes) for the point location, principal directions and the normal, and 2 floats (8 bytes) for the two curvature values. We anticipate over 80% compression by using quantized values, delta differences, and index colors. Both the specular and diffuse shading are done at the hardware level. However, accumulation buffer is not supported in hardware by nVIDIA and is implemented in software by the OpenGL drivers. So the case of both diffuse and specular illumination can be slow.

On an average, about $330,000$ DPs can be rendered per second with diffuse illumination. The diffuse and specular illumination passes take around the same time. The main bottleneck in rendering is the bus bandwidth. 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 per-pixel computation is higher for specular illumination. The pixel-fill rate was not a bottleneck as the frame rates did not vary with the size of the window.

Table 1. Summary of results: NP = Number of points, SS = Storage Space, PT = Pre-processing time, FPS = Frames per second. (Illumination is done with a moving light source)

Statistical Highlights	**Without Simplification**			**With Simplification**		
	Teapot	*Head*	*Camera*	*Teapot*	*Head*	*Camera*
NP	156,800	376,400	216,712	25,713	64,042	46,077
SS (in MB)	9.19	22.06	12.69	1.51	3.75	2.70
PT (in sec)	22.5	22.2	15.25	146.5	485.5	178.17
FPS (Diffuse)	2.13	0.89	1.59	12.51	5.26	6.89
FPS (Specular)	2.04	0.88	1.52	11.76	5.13	6.67

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

Statistical Highlights		**Rendering Primitive**			
		DP	*SP*	*RP*	*EP*
Test 1	NP	156,800	156,800	156,800	156,800
	SS (in MB)	9.19	4.90	4.90	4.90
	FPS (Diffuse)	2.13	11.76	10.52	2.35
Test 2	NP	156,800	1,411,200	1,155,200	320,000
	SS (in MB)	9.19	44.10	36.10	10.01
	FPS (Diffuse)	2.13	1.61	1.49	1.16

The main focus of this paper is the rendering quality and efficiency delivered by DPs as rendering primitives. Previous works on point sample rendering have orthogonal benefits such as faster transformation [21] and multiresolution [1, 17, 19] which can potentially be extended to DPs. To demonstrate the benefits of DPs 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. From this 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 ball [19]. They are rendered without 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. An orthogonal projection on a plane parallel to the view plane and located at the position of the point 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 [17]. 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 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 [19]. This is implemented in hardware using the register combiners.

DPs were compared with the splatting primitives for two test cases: (*Test 1*) same number of rendering primitives and (*Test 2*) approximately similar visual quality of rendering. For *Test 1*, DPs were found to deliver a much better rendering quality for the same number of primitives as seen in Figure 6 and summarized in Table 2. 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 elliptical primitives. Sample renderings of *Test 2* are shown in Figure 6 and the results are summarized in Table 2. For this test the number of square, rectangle, and elliptical primitives were increased by increasing the sampling frequency of the uniformly sampled model used for DPs. In *Test 2*, DPs clearly out-performed the splatting primitives in both criteria.

7 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. High rendering quality is achieved because the normal distribution is fairly accurate. The rendering efficiency of DPs is attributed to the sparse surface representation that reduces bus bandwidth.

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. DPs are currently implemented for a NURBS representation. The main challenge in extending them to polygonal models would be the accurate computation of curvature properties and handling of discontinuities.

In its current form DPs are not efficient when a large set of points fall onto the same pixel while rendering. We plan to explore a multiresolution scheme of DPs that will efficiently render lower frequency versions of the original surface under such instances. Texturing can be achieved by texturing the rectangles $\mathbf{r_p}$ and having a separate texture pass while rendering. The texture coordinates of the vertices of $\mathbf{r_p}$ can be computed with the aid of the object space parameterization of $\mathbf{S_p}$.

8 Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments that were exceptionally detailed and insightful. 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 also like to thank Robert McNeel & Associates for the Head and the Camera models and for the openNURBS C++ code. Last, but not the least, we would like to acknowledge NSF funding grants IIS00-81847, ACR-98-12572 and DMI-98-00690.

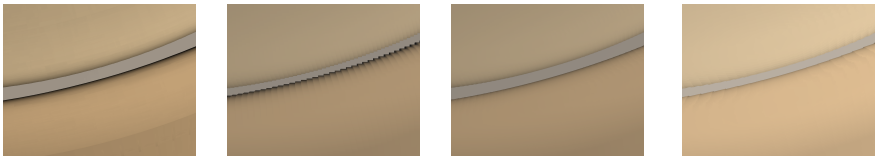
References

1. 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.
2. L. Darsa, B. C. Silva, and A. Varshney. Navigating static environments using image-space simplification and morphing. In *1997 Symp. on Interactive 3D Graphics*, pages 25–34, 1997.

3. 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.
4. M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall. Inc., Englewood Cliffs, New Jersey, 1976.
5. J. P. Grossman and William J. Dally. Point sample rendering. In *Rendering Techniques '98*, Eurographics, pages 181–192. Springer-Verlag Wien New York, 1998.
6. I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proceedings of SIGGRAPH 99*, pages 325–334, 1999.
7. P. S. Heckbert and M. Garland. Optimal triangulation and quadric-based surface simplification. *Computational Geometry*, 14:49–65, 1999.
8. V. L. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Proceedings of SIGGRAPH 97*, pages 109–116, August 1997.
9. A. Kalaiah and A. Varshney. Modeling and rendering of points with local neighborhood. In *Technical Report CS-TR-4224, Computer Science Department, University of Maryland, College Park*, March 2001 (http://www.cs.umd.edu/∼ark/dpr.pdf).
10. 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).
11. V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of SIGGRAPH 96*, pages 313–324, August 1996.
12. M. Levoy and T. Whitted. The use of points as a display primitive. In *TR 85-022, Computer Science Department, University of North Carolina at Chapel Hill*, January 1985.
13. D. Lischinski and A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98*, Eurographics, pages 301–314. Springer-Verlag New York, 1998.
14. W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. In *1997 Symposium on Interactive 3D Graphics*, pages 7–16, April 1997.
15. M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *Proceedings of SIGGRAPH 2000*, pages 359–368, July 2000.
16. A. Paeth. *Graphics Gems*, volume 5. Academic Press, 1995.
17. 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.
18. 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*, pages 23–34. Springer-Verlag New York, June 1997.
19. S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH 2000*, pages 343–352, July 2000.
20. S. Rusinkiewicz and M. Levoy. Streaming QSplat: A viewer for networked visualization of large, dense models. In *Symposium of Interactive 3D Graphics*, pages 63–68, March 2001.
21. J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *Proceedings of SIGGRAPH 98*, pages 231–242, August 1998.
22. 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.
23. G. Turk. Re-tiling polygonal surfaces. In *Proc. of SIGGRAPH 92*, pages 55–64, July 1992.
24. 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.
25. M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proceedings of SIGGRAPH 2001 (to appear)*, 2001.

(a)DP (b)SP (c)RP (d)EP

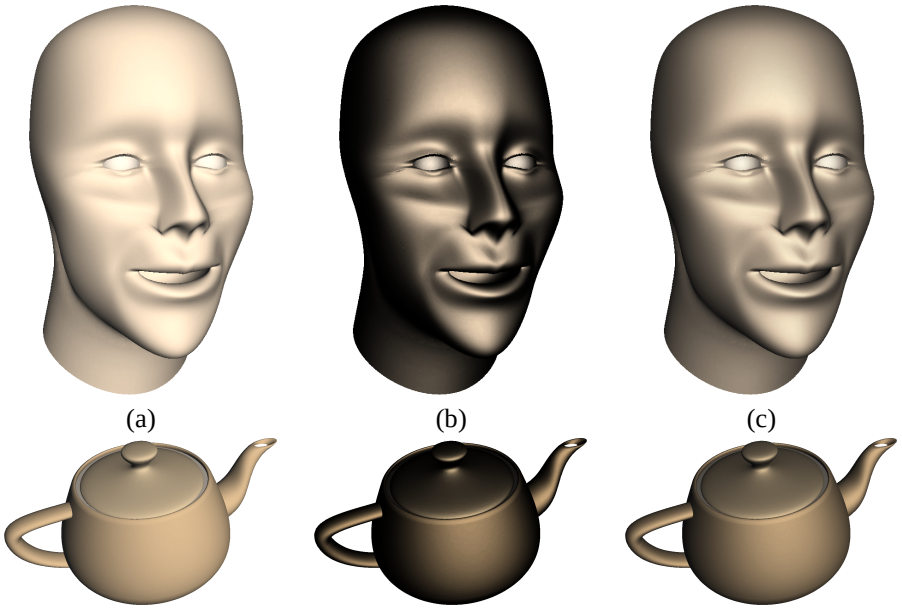
Test 1: Comparison of rendering quality for the same number of rendering primitives



(a)DP (b)SP (c)RP (d)EP

Test 2: Approximately similar rendering quality achieved with different sampling frequency

Fig. 6. Selected areas of rendering of the teapot model for the two test cases: (a) Differential Points. (b) Square Primitive. (c) Rectangle Primitive. (d) Elliptical Primitive



(a) (b) (c)

(a) (b) (c)

Fig. 7. Illumination and per-pixel Shading: (a) Diffuse Illumination. (b) Specular Illumination. (c) Diffuse and Specular Illumination