Silhouette Enhanced Point-Based Rendering

José Luiz Luz, Luiz Velho, Paulo Cezar P. Carvalho

IMPA–Instituto Nacional de Matemática Pura e Aplicada
Estrada Dona Castorina, 110, 22460
Rio de Janeiro, RJ, Brasil
{josell, lvelho, pcezar}@visgraf.impa.br

ABSTRACT

With the recent advances in the 3D scanning field, the size of datasets to be displayed has increased up to billions of points. Typically, we have a dense, unstructured set of points without connectivity information. Most researchers have proposed the point-surfel association to represent the surface's geometry and to render it using a planar approximation for each point. This paper proposes an alternative approximation, where curved surface elements (c-surfels) are employed, in order to get better adaptation to the surface to be rendered. We also use texture mapping and blending, to produce a perceptually better visualization. Improvements caused by using curved surfels instead of planar ones are especially noticeable at the object's silhouette.

Keywords: Point-based Rendering, Graphics Data Structures, Texture Mapping.

1 INTRODUCTION

The problem of handling 3D datasets obtained from real-world objects has drawn the attention of the research community. Typically, we have a dense, unstructured set of points (sometimes, billions of them) without connectivity information. The techniques to treat these datasets have evolved, especially due to research on triangle meshes, since triangles are the most popular modeling primitives. Nevertheless, with the growing use of complex geometries the overhead associated with polygonal meshes is reaching prohibitive levels. As a consequence, other representations become more attractive.

More recently, there has been a trend to use point-based representations. Given the simplicity of points, they seem natural for modeling and rendering. We can obtain them from parametric representations (polygonal meshes, splines patches, subdivision surfaces) and non-parametric ones (implicit surfaces, fractals). Other representations use directly the point samples, such as particle systems, volumetric data in medical images, and image-based rendering.

WSCG 2005, January 31-February 4, 2005
Plzen, Czech Republic.

Point-based representations can compensate for their lack of connectivity information, by spatial proximity between the points in a sufficiently dense sample, without causing loss of quality in the final image. With the texture mapping technique introduced by Catmull [Cat74] we can improve the visualization while keeping the object fundamental geometry, getting better results for planar surfaces or slightly curved surfaces. Moreover, blending operations can be used to reduce discontinuities in the texture mapping of overlapping surfaces.

This paper proposes an alternative approximation where curved surface elements (c-surfels) are employed, in order to get better adaptation to the surface to be rendered. We also use texture mapping and blending, to produce a perceptually better visualization. Improvements caused by using curved surfels instead of planar ones are especially noticeable at the object's silhouette.

We discuss related work in Sec. 2 and then describe the steps to build our primitives in Sec. 3. In Sec. 4, point out some factors that contribute to the use of c-surfels at the object's silhouette and show some results and applications, and in Sec. 5 we present some conclusions, and discuss limitations and future work.

2 RELATED WORK

Levoy and Whitted [LT85] in 1985 proposed the use of points as universal rendering primitives. The conceptual idea was to have a single element good enough

to model and render any kind of object. The surface could be represented by points, considering it differentiable, and estimating the tangent plane and the normal from a small set of neighboring points.

About a decade later, in 1998, Grossman and Dayle [GD98] addressed object sampling from a set of orthographic views. They used a hierarchy of depth buffers to determine when a pixel is considered a hole or not.

Various researchers have published their ideas relative to point rendering and modeling. In 2000 three papers introduced the ground ideas for our work.

Pfister et al. [PZBG00] extended Grossman and Dayle's work by adding hierarchical level of detail (LOD) control and hierarchical visibility culling. They proposed the paradigm of surface elements (surfels) to efficiently render complex geometric objects. Surfels are primitives without explicit connectivity, with attributes such as depth, texture color, and normal. The objects are sampled from three orthogonal views and the sampling is stored in a octree. When rendering, the visible surfels and the holes are detected; the surface attributes are interpolated at the pixels that have samples.

Rusinkiewicz and Levoy [RL00] devised a rendering system called Qsplat. It allows real-time viewing of models consisting of hundred of millions of points samples. They used a bounding sphere hierarchy for hierarchical LOD control and culling and they employed splatting for surface reconstruction. The splats are oriented along the view plane and rendered in a back-to-front order.

Schauffer and Jensen [JS00] used small surfels to render point-based representations. They considered these surfels as tangent plane approximations and employed ray tracing to interpolate per-point attributes.

3 POINT RENDERING

There are many approaches to render objects from its point-based representation. We can distinguish two different proposals. The first renders the primitives as 0-dimensional points, while the second renders the primitives considering an area for each point.

We can also classify the algorithms to render point-based surfaces in two groups: those that do *forward mapping* and those that do *backward mapping*. The methods in the first group send points directly to rendering pipeline and compute their contributions to the pixels; thus we have projection from object-space to image-space. Splatting [Räs02] is an example of this type of algorithm. The methods in the second group compute for each pixel in the image the object that projects on it; therefore, we have projection from image-space to object-space. Ray tracing and polygon texture mapping are examples of this type of algorithm. Some algorithms use a combination of both techniques.

Point rendering requires information about point attributes such as position, normal, color, texture coordinates, etc. We may also associate an element of surface to a point, i.e. a surfel. The surface area at each point can be considered circular and characterized by a radius, that must be sufficiently large to ensure a hole-free reconstruction. We can store other attributes for a surfel, such as transparency and material properties.

In this paper we have as input a set of point samples on a smooth surface, which are assumed to be sufficiently dense so that the distribution of the points over the surface can be considered approximately uniform. We also assume that a normal is available at each point, and, in some cases, textures coordinates are also available. We do forward mapping and texture mapping, and regard each point as either a surfel or curved surfel (c-surfel), with the same fixed radius for all surfels which is computed before sending them to the rendering pipeline.

3.1 Building our primitives

A topological surface is a subset S of an Euclidean space $\mathbb{R}^3$, which is locally homeomorphic to the Euclidean space $\mathbb{R}^2$, that is, for each point $p \in S$ there is a spherical neighborhood $B^3_\varepsilon \subset \mathbb{R}^3$ with center p, in such a way that the subset $B^3_\varepsilon \cap S$ is homeomorphic to the open unit disk in the Euclidean plane (Figure 1).

$$B^2_1 = \left\{(x, y) \in \mathbb{R}^2; x^2 + y^2 < 1\right\}$$

Intuitively this definition says that a surface is obtained by overlapping several deformed pieces of the plane [VG03].

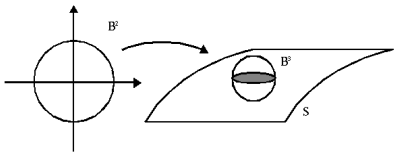


Figure 1: Homeomorphism

Let us represent these pieces by means of the function $\phi(r)$ ($r \in [0, \varepsilon]$), which is given by:

$$\phi(r) = \begin{cases} 0 & \text{, planar approximation} \\ 1 - e^{-\left(\frac{r^2}{h^2}\right)^2} & \text{, almost planar approximation} \end{cases}$$

The expression "almost planar" is used to represent our c-surfel, and h is a constant which defines the surfel curvature. From this function we can obtain a

plateau-like surface (gaussian approximation) or a planar one, defined by the points $(r.\cos\theta, r.\sin\theta, \phi(r))$ $(\theta \in [0, 2\pi])$, which allows one to use it as a local approximation to a point, and allowing a perceptually better adaptation to the surface (Fig. 2).

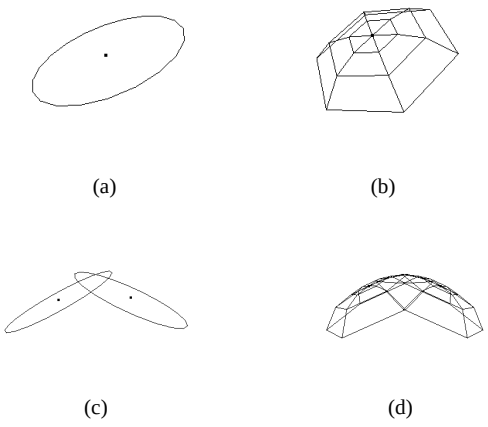


Figure 2: (a) planar surfel. (b) c-surfel. (c),(d) overlapping surfels

When we put these approximations on the surface we obtain overlapping surfels (c-surfels). Therefore, to give an appearance of continuity to the surface we use texture mapping and blending operations.

3.2 Texture mapping and blending

We map to our surfels a single texture with only one color, and opacity falling off radially according to a gaussian approximation. The alpha-value of pixel (i, j) (initially set to 1.0) is multiplied by a factor $f(i, j)$ given by:

$$f(i,j) = e^{-((i-x_0)^2+(j-y_0)^2)/d^2},$$

where

(i, j)	- position at texture;
(x_0, y_0)	- texture center;
d	- radial fall-off factor.

We map the texture considering initially $r \in (0, 1]$, and using the function:

$$(r.\cos(\tfrac{2k\pi}{n}), r.\sin(\tfrac{2k\pi}{n}), \phi(r))$$

$$\downarrow$$

$$(\tfrac{1}{2}\cos(\tfrac{2k\pi}{n}) + \tfrac{1}{2}, \tfrac{1}{2}\sin(\tfrac{2k\pi}{n}) + \tfrac{1}{2}).$$

Where n is the number of sides of the surfel and k is in the interval $(0, n]$. Thus, the surfel (c-surfel) center corresponds exactly to the texture center in a parametric space uv defined in $[0, 1]$x$[0, 1]$, and transparency is observed at the surfel border, as shown in Figure 3. Further the surfel is scaled in according to the computed radius.

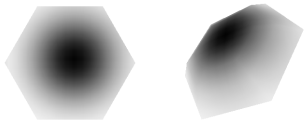
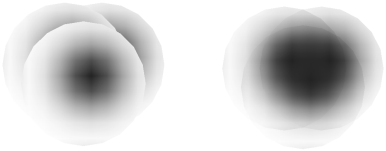


Figure 3: mapped texture surfel.

To handle overlapping surfels we need to know how to use alpha values to combine the currently processed color and the one previously stored at color-buffer.

The color c at position (x, y) in the final image is computed as a normalized weighted mean of contributions from mapped texture colors (surfels). The normalization is necessary since the weights (alpha values) do not necessarily constitute a partition of the unity at screen-space, due to irregular surfel position and the truncation of the ideal alpha mask (Figure 4).



(a) no blending. (b) blending.

Figure 4:

We have:

$$c(x,y) = \frac{\sum_i c_i.w_i(x,y)}{\sum_i w_i(x,y)}$$

c_i	- i^{th} polygon color
$w_i(x, y)$	- weight at position (x, y)

We sort the points before rendering their corresponding surfels, since ordering affects smoothness at the final image, and we use a multipass rendering due to the interaction between blending and Z-buffering.

Figure 5: incorrect occlusion and blending

3.3 Visibility

Rusinkiewicz and Levoy [RL00] proposed a multipass rendering in OpenGL to ensure that both occlusion and blending happen correctly (Fig. 5). For the first pass, we render the surfel with an offset z_o away from the viewer. We do this only into the depth buffer. For the second pass we turn off the depth offset allowing depth comparison, without updating the depth buffer and writing to color-buffer. This steps blend together with the correct occlusion all surfels within a depth range z_o of the surface.

3.4 Surfel size

We need to compute the correct size of the surfels, which will be the same for all of them. To compute the size of the surfels we use eigenanalysis of the covariance matrix of a local neighborhood (Principal Component Analysis - PCA) to estimate local surface properties, as proposed by Pauly [MPK02]. Given a point cloud $P = \left\{p_i \in \mathbb{R}^3\right\}$, the covariance matrix C for a sample point p is given by

$$\mathbf{C} = \begin{bmatrix} p_{i_1} - \overline{p} \\ \cdots \\ p_{i_k} - \overline{p} \end{bmatrix}^T \cdot \begin{bmatrix} p_{i_1} - \overline{p} \\ \cdots \\ p_{i_k} - \overline{p} \end{bmatrix} \quad , i_j \in N_p$$

where $\overline{p}$ is the centroid of the neighbours p_{i_j} of p, and N_p is the index set of the k-nearest neighbours of the sample p. The principal components are the solutions to the following eigenvector problem:

$$C \cdot v_l = \lambda_l \cdot v_l \quad , l \in \{0, 1, 2\}$$

We use the eigenvalues and their corresponding eigenvectors to do a space partition. Since eigenvalues give a measure to the variation of the points in N_p, we take the eigenvector corresponding to the greatest eigenvalue, and define a splitting plane in a BSP-tree, that we use to perform hierarchical clustering.

Assuming that $\lambda_0 \leq \lambda_1 \leq \lambda_2$, the eigenvalue λ_0 describes the variation along the surface normal. We define

$$\sigma_n(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2},$$

as the surface variation at point p in a neighborhood of size n. If $\sigma_n(p) = 0$ all points lie in the plane. Then we use that as a subdivision criterion to locate clusters with exactly three non-collinear points (Figure 6). The size of the surfel corresponding to a cluster is the diameter of the smallest circle circumscribed to its associated triangle. Since we assume that our sample is approximately uniformly distributed, we expect that the triangles have approximately the same area, and use the average of all surfel sizes as a common size used in the rendering process. But if we have regions with uneven distribution, holes can appear on the surface.

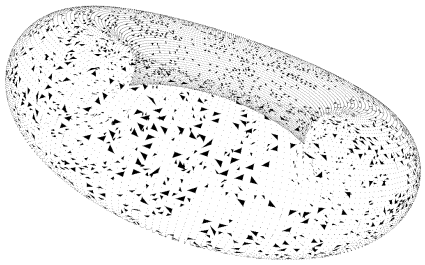


Figure 6: triangles on the surface obtained by clustering

4 VISUALIZING WITH OUR SURFELS

We associate to each point either a planar surfel or a c-surfel, which are constructed, and stored. Then they are oriented, translated and scaled accordingly to the point attributes. Some factors contribute to obtain good results when using c-surfels, Among them we can highlight the following ones:

- Overlapping c-surfels provides a better local approximation to the points on the surface, since they have an associated mesh, which provides more details;
- All normals in the c-surfel have the same orientation as the normal at the point; thus shading and blending operations give an appearance of continuity to the rendered surface.

Figure 7 shows a result using only c-surfels. However, as the c-surfel have a mesh, the computational cost due to rendering them can become high, depending on the number of polygons of the mesh. The table 1 shows the performance of our unoptimized C implementation for different c-surfels (Pentium IV 1.4 GHz 512 RAM).

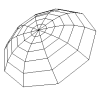
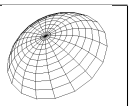
c-surfel	 24 polygons	 60 polygons	 200 polygons
Igea (134.345 points)	1.06 *fps*	1.00 *fps*	0.41 *fps*

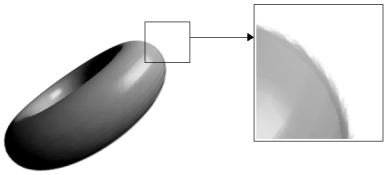
Table 1: different c-surfel resolutions

Instead of using only c-surfels, we propose to use both planar and curved surfels, and since using flat surfels at the silhouettes of the object may result in less

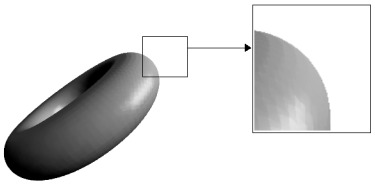
(a) igea (134.345 points).

Figure 7:

precise rendering (Figure 8), we use c-surfels at the silhouette points (and close to them) and flat ones for the rest of the surface as illustrated in Figure 9.



(a) flat surfels at the silhouette.



(b) c-surfels at the silhouette.

Figure 8:

Then if the angle between the vector from a point p to the observer's eye and the normal at this point is in the interval $[90^o - \epsilon, 90^o + \epsilon] \quad (\epsilon \in [0^o, 20^o])$, we use a c-surfel for this point, otherwise we employ a flat surfel. The table 2 shows some time measures using planar surfels with 6 sides, and c-surfels formed by 24 polygons.

We can see that using c-surfels at the object's silhouette does not increase the cost significantly, when comparing to the all planar case. Therefore, their use seems a good option to enhance the level of details at these regions. Some rendering results are shown in the figure 10.

When we have texture coordinates available for the points, we can do texture mapping by blending the texture image colors: given the texture coordinates we verify the corresponding color in the texture-space, and map it to the surfel (Figure 11).

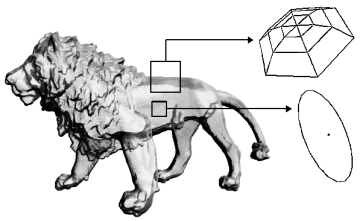


Figure 9: surfel and c-surfels

Model	planar surfel	c-surfel	planar surfel + c-surfel
Igea (134.345 points)	1.22 *fps*	1.06 *fps*	1.18 *fps*
Ball joint (137.059 points)	1.08 *fps*	0.96 *fps*	1.03 *fps*
Budha (389.347 points)	0.35 *fps*	0.3 *fps*	0.33 *fps*

Table 2: using planar and almost planar approximations

5 CONCLUSIONS

We proposed the use of curved surfels and texture blending to visualize a set of point samples, by exploring the adaptation to the surface when the c-surfels overlapping each other, which provides more details because of their mesh. Since the associated computational cost can be high, we used these c-surfels only at the models silhouette, without increasing the overhead very much.

In future work we intend to improve the technique to handle difficult regions, such as those with high-curvature, perhaps storing a curvature information for each point, estimated in terms of its local neighborhood, and use that to adapt the surfel curvature.

6 ACKNOWLEDGMENTS

The authors are partially supported by CNPq research grants. This research has been developed in the VISGRAF Laboratory at IMPA. VISGRAF is sponsored by CNPq, FAPERJ, FINEP and IBM Brasil.

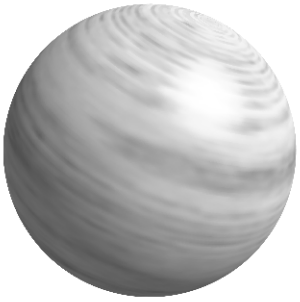
(a) budha (389.347 points).

(b) hand (327.323 points).

(c) ball joint (137.059 points).

(d) rabbit (44.691 points).

Figure 10:



(a) texture-mapped sphere.

(b) texture-mapped torus.

Figure 11:

7 REFERENCES

[Cat74] E. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Department of Computer Science, University of Utah, 1974.

[GD98] J. P. Grossman and William J. Dally. Point sample rendering. *Eurographics Rendering Workshop 1998*, pages 181–192, 1998.

[JS00] Henrik Wann Jensen and Gernot Schauffer. Ray tracing point sampled geometry. In Springer-Verlag, editor, *Rendering Techniques 2000*, pages 319–328. Eds. Peroche and Rushmeier, 2000.

[LT85] Marc Levoy and Whitted Turner. The use of points as a display primitive. Technical Report 85-022, University of North Carolina at Chapel Hill, 1985.

[MPK02] Markus Gross Mark Pauly and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. *In Proceedings IEEE Visualization 2002*, pages 163–170, Computer Society Press, 2002.

[PZBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Barr, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000*, pages 335–342. ACM Press/ ACM SIGGRAPH/ Addison Wesley Longman, 2000.

[Räs02] Jussi Räsänen. Surface splatting: Theory, extensions and implementation. Master's thesis, Dept. of Computer Science, Helsinki University of Technology, 2002.

[RL00] Szymon Rusinkiewicz and Mark Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, pages 343 – 352. ACM Press/Addison-Wesley Publishing Co, 2000.

[VG03] L. Velho and J. Gomes. *Fundamentos de Computação Gráfica*. Impa, Rio de janeiro, 2003.