

# Hardware-accelerated Extraction and Rendering of Point Set Surfaces

E. Tejada<sup>1†</sup>, J. P. Gois<sup>2‡</sup>, L. G. Nonato<sup>2‡</sup>, A. Castelo<sup>2‡</sup> and T. Ertl<sup>1†</sup>

<sup>1</sup>Institute for Visualization and Interactive Systems, University of Stuttgart, Germany.

<sup>2</sup>Institute of Mathematics and Computer Science, University of São Paulo, Brazil.

---

## Abstract

*Point-based models are gaining lately considerable attention as an alternative to traditional surface meshes. In this context, Point Set Surfaces (PSS) were proposed as a modeling and rendering method with important topological and approximation properties. However, ray-tracing PSS is computationally expensive. Therefore, we propose an interactive ray-tracing algorithm for PSS implemented completely on commodity graphics hardware. We also exploit the advantages of PSS to propose a novel technique for extracting surfaces directly from volumetric data. This technique is based on the well known predictor-corrector principle from the numerical methods for solving ordinary differential equations. Our technique provides good approximations to surfaces defined by a certain property in the volume, such as iso-surfaces or surfaces located at regions of high gradient magnitude. Also, local details of the surfaces could be manipulated by changing the local polynomial approximation and the smoothing parameters used. Furthermore, the surfaces generated are smooth and low frequency noise is naturally handled.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computational Geometry and Object Modeling]: Curve, surface, solid, and object representations G.1.2 [Approximation]: Approximation of surfaces and contours I.3.7 [Three-Dimensional Graphics and Realism]: Ray-tracing

---

## 1 Introduction

Lately, point-based methods are becoming popular within the computer graphics community due to their inherent advantages and due to the advances on both the acquisition methods that provide unorganized point clouds and the modeling tools and techniques available for creating and editing point-based models. Simulation algorithms, e.g. deformable bodies and animation, and rendering methods have also been addressed, which have become the base for the development of new point-based graphics methods [KB04]. In this context, Alexa et al. [ABCO\*01, ABCO\*03] and Zwicker et al. [ZPKG02] proposed re-sampling techniques for generating dense samplings in order to cover the image space consistently by simply projecting the points onto the screen. The surfaces thus obtained are known as *point set surfaces* (PSS).

PSS methods offer a natural mechanism to deal with noisy data sets, which are hard to handle with surface mesh reconstruction techniques from point clouds. In fact, few surface reconstruction methods have been proposed to solve this problem [MAVdf05, DG04].

PSS have been further studied by Amenta and Kil [AK04a, AK04b] who noted important properties about the domain of a PSS and the behavior of the *moving least-squares* (MLS) and *weighted least-squares* (WLS) minimization strategies in the context of PSS. Based on their observations, we propose a novel technique to extract surfaces from volumetric data stored in uniform grids, inspired by the well-known ‘predictor-corrector’ principle. Our method is able to provide good approximations to the surfaces defined by a given feature in the volume, such as iso-surfaces and surfaces located at regions of high gradient magnitude. This last class of surfaces is addressed since, as Kniss et al. [KKH01] pointed out, although there is no mathemati-

---

† e-mail: {eduardo.tejadalthomas.ertl}@vis.uni-stuttgart.de

‡ e-mail: {jngois|gnonato|castelo}@icmc.usp.br

cal prove, regions of interest are assumed to be located at regions of high gradient magnitude.

Mesh surfaces extracted from volumetric data have some inherent disadvantages, such as the need for defining the polygon characteristic size and the need for storing topological information. Also, important details may be omitted or coarse regions might be excessively detailed. Although more sophisticated methods were introduced in order to handle these problems [SP97], the mesh must be locally recomputed and refined, which is computationally expensive. Also, it is necessary to define an initial surface, implying that the user must know *a priori* some characteristics of the object in order to define a good initial approximation. The authors also mention the possibility of handling noisy data if sophisticated strategies of refining and displacement of vertices are used.

On the other hand, our method handles these problems naturally. Since it is based on local polynomial approximations, the precision of the model can be locally defined. Also, low frequency noise in the data is easily handled, due to the fact that the local approximations are computed by means of least-squares (LS) approaches. Our approach generates smooth surfaces avoiding the piece-wise approximation of mesh-based methods.

To achieve interactive frame rates, the implementation was completely realized on commodity graphics hardware, making use of the support for loops and dynamic branches in the shader stage provided by the Pixel Shader 3 model [Mic06]. We also propose an algorithm for ray-tracing PSS on the GPU which is considerably faster than CPU implementations [AA03b]. As stated by Adamson and Alexa, PSS have clear advantages over other representations when ray-tracing is used, namely the locality of the computations, the possibility of defining a minimum feature size and the fact that the surface is smooth and a 2-manifold [ABCO\*03]. Beside the inherent implications of these three characteristics, the second advantage can be exploited when computing the intersection of the ray with the surface, whilst the last one turns CSG operations feasible [AA03b].

## 2 Related Work

In general, PSS methods make use of classical differential geometry results to ensure consistent local representation through polynomial approximations. In this respect, Zwicker et al. [ZPKG02] use linear minimizations based on WLS to define the local polynomial functions, whilst Alexa et al. [ABCO\*01, ABCO\*03] employ a non-linear strategy based on MLS to define a local coordinate system over which a local polynomial approximation to the surface is calculated using WLS. This approach was based on the work by Levin [Lev98, Lev03].

Amenta and Kil [AK04a] proposed an elegant approach for generating point set surfaces that makes use of computer

vision and computational topology concepts. They also presented a theoretical analysis about the behavior of the minimization strategies adopted by MLS and WLS. Based on these results, a new PSS method that captures sharp corners was introduced by the authors [AK04b]. Some PSS techniques also make use of implicit functions [AA03a, AK04a] whose zero set is guaranteed to generate surface representations. Also, guarantees for homeomorphical approximations to the original object based on this implicit function are presented [DGS05].

Dey and Sun [DS05] propose a hybrid method that uses MLS and surface reconstruction techniques to deal with noisy data. One further hybrid technique that employs MLS together with an incremental surface reconstruction method is described by Scheidegger et al. [SFS05]. Fleishman et al. [FCOS05] propose a PSS method based on robust statistics techniques, which is able to handle very noisy data sets. The work by Reuter et al. [RJT\*05] is also aimed at handling such data sets, based on the *enriched reproducing kernel particle approximation* technique. Both works present equally good results even for models with sharp corners.

Rendering volumetric data using point-based strategies has also been addressed in previous work. Co et al. [CH03] proposed a method named ‘iso-splatting’, that extracts points in the domain defined by the iso-value and uses them as input to compute a third-degree polynomial approximation to the surface. The rendering is performed by means of splatting operations by projecting points onto the iso-surface defined by this interpolation. Livnat and Tricoche [LT04] proposed a hybrid method of iso-contouring based on points and triangles. Nested-grids are employed to traverse the domain and decide whether a triangle or a single point must be rendered.

Recently, Fenchel et al. [FGS05] proposed a technique for animating surfaces extracted from 4D volumetric data. The input of the method is a sequence of volumes changing along time. A surface mesh is extracted using the marching cubes algorithm and the dynamic mesh is obtained by moving the vertices of this mesh by means of WLS using the volume associated with the current time step. The technique is aimed at animating data from medical applications, specifically from the beating heart.

## 3 Least-Squares, Weighted Least-Squares, Moving Least-Squares and Point Set Surfaces

A PSS of a given point cloud  $P$  is defined as the set of points  $p \in \mathbb{R}^3$  for which  $\mathcal{P}(p) = p$ , where  $\mathcal{P}$  is a projection operator defined as follows: given a point  $r \in \mathbb{R}^3$ , its projection  $\mathcal{P}(r)$  is calculated with a two-step process. First, a local plane  $H(n, q)$ , with normal  $n$  and passing through  $q = r + tn$ , where  $t$  is a scalar, is found using either MLS [ABCO\*01] or WLS [ZPKG02], so that  $H(n, q)$  approximates the surface in the neighborhood  $\mathbf{N}(r) \subset P$  of  $r$ . Then, a local orthonormal coordinate system is defined over  $H(n, q)$  with origin in

$q$ , on which a bivariate polynomial approximation  $g(x, y)$  is computed by means of WLS. Then, the projection operator is defined by  $\mathcal{P}(r) = q + g(0, 0)n = r + (t + g(0, 0))n$ .

As stated before, this process is performed using the MLS and WLS methods. WLS differs from the traditional LS scheme in the fact that weights are introduced to force some points to have a greater influence in the solution.  $n$  and  $q = r + tn$  can be computed using WLS by performing the following minimization

$$\min_{n,t} \sum_{p_i \in \mathbf{N}(r)} \langle p_i - (r + tn), n \rangle^2 w(\|p_i - p_w\|), \quad (1)$$

where  $w(x)$  is a non-negative monotonically decreasing function. Frequently, a Gaussian  $w(x) = e^{-\frac{x^2}{h^2}}$  is used, where  $h$  represents the local level of detail of the object. Here, the weighter point  $p_w$  is known *a priori* (usually  $r$ ).

The MLS method, proposed by Lancaster and Salkauskas [LS81] for smoothing and interpolating data, is a weighted least-squares scheme, that finds not only the best approximation to the set of weighted points, but also the best weighter at the same time. Thus, the plane that best approximates a set of points is obtained by computing

$$\min_{n,t} \sum_{p_i \in \mathbf{N}(r)} \langle p_i - (r + tn), n \rangle^2 w(\|p_i - (r + tn)\|). \quad (2)$$

Since  $q$  is also the weighter in  $w$ , Equation 2 is a non-linear function. The results of the WLS and MLS minimizations are slightly different. Since the MLS minimization finds the best plane that approximates the weighted points and the weighter simultaneously, the minimum can approximate a tangent plane to the surface at a point, if the interval of search is close enough to the set of points. This is not the case for the WLS strategy, which only finds the best plane that approximates the weighted points. Therefore, the MLS is assumed to produce smoother PSS and more accurate results.

#### 4 Extracting Point Set Surfaces from Volumetric Data

As stated before, in our approach we use a strategy inspired by the predictor-corrector methods which make use of two numerical approaches to solve ordinary differential equations. The first approach is a ‘predictor’ which provides a first rough solution but requires only limited information. This solution is the input to the ‘corrector’ which then finds a final more accurate solution. These processes can be iterated in order to improve the solution obtained [PTVF95].

In the study by Amenta and Kil [AK04b] it is shown that the energy function based on WLS strategies is able to project points far from the surface, but the solution is not as accurate as the one obtained with MLS. On the other hand, points far from the surface are not properly handled by MLS. Thus, we based our method on the belief that a combination of both schemes in a predictor-corrector sense would provide better results.

The first step of our PSS method is performed using WLS to find an initial approximation for the projection of a given point ‘on the surface’. This allows us to deal with points that are relatively far from the surface. As discussed above, the weights traditionally used in PSS minimization schemes are given by some monotone decreasing function of the distance from the point  $r$  to be projected to the point (voxel)  $p_i$  in its neighborhood  $\mathbf{N}(r)$ . However, in this step we use information on ‘how close a voxel is to a feature in the volume’ in order to weigh the voxels  $p_i \in \mathbf{N}(r)$ . The function  $\theta_j$ ;  $j = \{1, 2\}$  used to weigh the voxels  $p_i$  determines the feature used to identify the surface and therefore the surface generated. Iso-surfaces can be generated with our approach by using  $\theta_1(p_i) = 1 - e^{-\frac{|v-f(p_i)|^2}{k^2}}$ , where  $v$  is the iso-value defining the iso-surface,  $f(p_i)$  is the scalar value at  $p_i$  and  $k$  a smoothing factor. Also, assuming that regions of interest are located at regions of high gradient magnitude, a class of surfaces that depicts changes in the material properties can be obtained by using  $\theta_2(p_i) = \left(\frac{\|\nabla f(p_i)\|}{\max\{\|\nabla f(p_i)\|\}}\right)^2$ . With these weighting functions the following minimization is performed

$$\min_{n,t} \sum_{p_i \in \mathbf{N}(r)} \langle n, p_i - (r + tn) \rangle^2 \theta_j(p_i); j = 1, 2. \quad (3)$$

After the minimization, a local coordinate system is defined by the plane  $H(n, q)$ , where  $q = r + tn$ . On this local coordinate system we use WLS to find a bivariate polynomial  $g(x, y)$  that locally approximates the surface using as weighting function  $\theta_j$ ;  $j = \{1, 2\}$ . Defining  $\mathcal{P}(p)$  as the orthogonal projection of a point  $p$  on the fitted polynomial  $g(x, y)$ , the corrector scheme begins by performing

$$\min_{n,t} \sum_{p_i \in \mathbf{N}(\mathcal{P}(r))} \langle n, p_i - (\mathcal{P}(r) + tn) \rangle^2 \Theta(p_i) \quad (4)$$

where  $\Theta(p_i) = [(\alpha_1 \theta_{1,2}(p_i) + \alpha_2 \theta_3(\|p_i - (\mathcal{P}(r) + tn)\|))]$ ,  $\alpha_1 + \alpha_2 = 1$ ,  $\theta_3(d) = e^{-\frac{d^2}{h^2}}$ , and  $h$  is a smoothing factor. Then, as in the predictor step, we find the projection  $\gamma$  of  $\mathcal{P}(r)$  onto the polynomial that approximates the surface on the local coordinate system obtained with the new  $n$  and  $\mathcal{P}(r)$ . This polynomial is also calculated with the WLS method using the weighting function  $\Theta$ . Thus,  $\gamma$  will be the final projection of  $r$  on the point set surface.

#### 5 Computing and Ray-tracing PSS on the GPU

With the advent of the Pixel Shader 3 model [Mic06], support for dynamic loops and conditional branching is available in the shader stage. We exploit this new functionality to implement numerical methods on the GPU to compute and render PSS from volumetric data and point clouds.

##### 5.1 Computing PSS from volumetric data

To extract a PSS directly from the volumetric data, we render viewport-aligned slices clipped with the bounding box of the

volume, separated from each other by a distance of  $b = kh$  (in the direction of the view vector), where  $0.5 < k < 1$  and  $h$  is the smoothing factor to be used in  $\theta_3$  during the corrector step. The idea behind this operation is that, since the minimal feature size of the point set surface must be greater than  $h$ , by taking steps smaller than  $h$  we ensure that the intersection between each ray and the surface will be found [AA03b].

In order to reduce the computation time, we pre-compute the per-voxel information to be used and discard those fragments for which this information is smaller than a pre-defined threshold. For the case of iso-surfaces this data is  $|v - f(p_i)|$  and for the surfaces located in regions of high gradient magnitude is  $\|\nabla f(p_i)\|$ . This threshold must be low enough to ensure that a sufficient number of fragments is used for the rest of the process. To interpolate the data in the shader we use 8-bits per-channel textures. Therefore, the gradient must be coded to 8 bits when the texture is created and reconstructed in the shader.

For each fragment generated that passes the above mentioned test, we perform the following computations completely in a single rendering pass. In the predictor step we start by minimizing Equation 3 with respect to  $t$  by taking  $n = \frac{\nabla f}{\|\nabla f\|}$  (pre-computed using central differences filtered with the Sobel gradients filter) and defining the point  $r$  to be projected as the position of the fragment in space. This represents a straightforward computation since a simple linear minimization with respect to a scalar value must be performed. The neighbors  $p_i \in \mathbf{N}(q)$  are the neighboring voxels of  $q = r + tn$ . Thus, no spatial search is required during the whole projection process. After  $t$  is found, we use covariance analysis to find the new  $n$ . For that, we build the  $3 \times 3$  covariance matrix  $C_1(q) = \sum_{p_i \in \mathbf{N}(q)} (p_i - q)(p_i - q)^T \theta_{1,2}(p_i)$  and find the eigenvector associated with the smallest eigenvalue of the matrix, which is the new  $n$ . To find this eigenvector we use the *inverse power* method [PTVF95].

These two steps are iterated until the change in  $t$  is smaller than a threshold, using the  $n$  computed in the second step for minimizing with respect to  $t$  in the next iteration. Once  $n$  and  $t$  are defined, a polynomial approximation to the surface is calculated in a local coordinate system defined over the plane  $H(n, q)$  using WLS and weighting the points  $p_i$  with  $\theta_{1,2}(p_i)$ . To exploit the capabilities of the GPU to handle vector operations for vectors of size 4, we used the polynomial  $g(x, y) = Ax^2 + By^2 + Cxy + D$  for the local approximation (note that  $(x, y)$  is in the local coordinate system). Therefore, the matrix of the linear system to be solved is of size  $4 \times 4$  and thus easily handled in the shader. The projection of  $q$  on the local approximation is  $\mathcal{P}(r)$ , which we use as input to the corrector step.

In the corrector step we perform the minimization of Equation 4, first with respect to  $t$ . Since this is a non-linear minimization, we implemented the *Brent with derivative* algorithm [PTVF95] to perform it. Then,  $t$  is fixed to the value obtained in order to find the new normal  $n$ . We chose to

follow the approach used by Amenta and Kil [AK04a] and compute  $n$  as the eigenvector corresponding to the smallest eigenvalue of the matrix  $C_2(\mathcal{P}(r)) = \sum_{p_i \in \mathbf{N}(\mathcal{P}(r))} (p_i - \mathcal{P}(r))(p_i - \mathcal{P}(r))^T \Theta(p_i)$ . Note that in this case we use the neighbors of  $\mathcal{P}(r)$  instead of the neighbors of  $q$  as was the case for the predictor step.

These two steps are iterated until the change in  $t$  is smaller than a threshold. Then we let  $q = \mathcal{P}(r)$  and use it and  $n$  to define a local coordinate system on which a local polynomial approximation is calculated with WLS using the weighting function  $\Theta$ . Thus, the projection  $\gamma$  of  $q$  on the polynomial defines the projection of  $r$  on the point set surface. Then, we find the distance between  $\gamma$  and  $r$ , and if it is less or equal a pre-defined error,  $r$  is taken as the resulting intersection of the ray with the PSS. Otherwise, as in the work by Adamson and Alexa [AA03b], we find the intersection between the polynomial and the ray. If the intersection is within a region of confidence defined by a ball with radius  $b$  and center in  $r$ , the projection process is started again taking the intersection found as the new  $r$ . This process is repeated until the distance between the projection and  $r$  is less than the error, or the intersection is outside the ball. In the last case the fragment is killed, which, since we use depth tests, will simulate the jump to the next ball used by Adamson and Alexa.

## 5.2 Ray-tracing PSS from point clouds

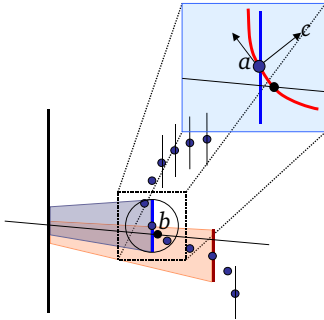
A similar approach as the one used for volumes is taken for ray-tracing point set surfaces extracted from point clouds on the GPU, with the difference that the computation is spread along multiple render passes due to the need for spatial queries. Also, since no interpolation is needed, 32-bits per-channel float textures are used to store the data.

The process is described below where each step represents a rendering pass. As in the work by Adamson and Alexa [AA03b] we define balls around the sample points, which will be rendered during the steps ‘intersection’, ‘form covariance matrix’, and ‘form system for polynomial fitting’. During the steps ‘find normals’ and ‘solve linear system and find projection’ we render a quad covering the entire viewport to generate a fragment per ray. The step ‘initial approximation’ is performed only once as a pre-processing step.

**Initial approximation.** In this render pass we calculate local polynomial approximations for each sample point  $p_i$ . For this, we render a single quad to generate a fragment per sample point. Each fragment calculates the corresponding local polynomial following similar steps as the described in the previous subsection, but using Equation 2 for the minimization. Since the point to be projected is the sample point itself, and is thus near the PSS, we assume  $t = 0$  and calculate the normal  $n$  using covariance analysis. Once  $n$  is defined, the polynomial is approximated in the local coordinate system (built using  $p_i$  as the origin and  $n$  as one of the vectors of the orthonormal basis), with  $p_i$  as  $r$  and using neighborhood information pre-stored in a 3D texture to obtain the

neighbors  $p_i$  of the sample point  $p_i$ . The result (coefficients) is rendered to a 32-bits per-channel float texture for further use.

**Intersection.** The nearest intersection of each ray with the local polynomials stored at the sample points defines our first approximation of the intersection of the ray with the point set surface. To find this intersection, we render viewport aligned discs with radius  $b$  as shown in Figure 1 (as a 2D example). Each fragment belonging to a disc calculates the intersection of the ray that passes through it with the polynomial stored at the respective sample point (top-right zoom in the figure). For this, we transform the ray into the local coordinate system, defined by  $a$  and  $c$  in our 2D example, where  $c$  is the normal  $n$  calculated in the first step of the algorithm. If there is no intersection or if the intersection is outside the ball, the fragment is killed. Thus, using depth tests we obtain the nearest intersection  $r$  of the ray with the local polynomials. Once  $r$  is determined and stored in a float texture, we find its projection on the point set surface. This is done in the four next steps.



**Figure 1:** Calculating the intersection of the ray with the local approximation stored in each sample point.

**Form covariance matrix.** Since the point  $r$  found in the last step is assumed to be reasonably close to the PSS, we set  $t = 0$  and find  $n$  using covariance analysis. For that, we must form the covariance matrix using the nearest neighbors of the point  $r$ . Since performing  $k$ -nearest neighbors spatial searches is expensive we opted for performing a range query by rendering discs with a radius  $\rho$  sufficiently large to influence the points in the neighborhood of each point, being  $\rho = 2h$  a good estimate for homogeneously sampled point clouds.

Each fragment generated this way calculates its distance to the intersection point on the ray passing through it in order to ensure that it is in the neighborhood of the intersection. In Figure 1 the zoomed disc influences the intersection point on the ray since it is within a distance  $\rho$  ( $b$  in the figure), whilst the influence of the disc in the back is discarded by means of a kill instruction for the fragment through which the ray passes. Each fragment belonging to

the disc corresponding to  $p_i$  that passes the proximity test calculates  $(p_i - r)(p_i - r)^T w(p_i - r)$ . The results of the fragments in the neighborhood of  $r$  are accumulated using one to one blending to three 16-bit-per-channel float textures that hold the  $3 \times 3$  matrix (since blending to a 32-bit-per-channel texture is prohibitively slow).

**Find normals.** In this step we render a single quad covering the viewport to generate a fragment per ray. Each fragment calculates the eigenvector associated to the smallest eigenvalue of the matrix obtained in the previous step, using a GPU implementation of the inverse power method as in the last subsection. The result is written to a float texture.

**Form system for polynomial fitting.** Once the normal at each intersection point is found, we must calculate the polynomial approximation using WLS. For that, we form a linear system which solution will give us the coefficients of the polynomial. The process is similar to the one of the step ‘form covariance matrix’ with the difference that the value calculated by each fragment belonging to the disc corresponding to  $p_i$  is twofold, a  $4 \times 4$  matrix  $w(p_i - r)aa^T$  and a vector  $w(p_i - r)a$  of size 4, where  $a = [(p_i - r)_x^2 \ (p_i - r)_y^2 \ (p_i - r)_x(p_i - r)_y \ 1]^T$ . These results are accumulated by means of blending to four float textures to be used as input for the next step.

**Solve linear system and find projection.** The linear system formed in the last step is solved in a further render pass by rendering a quad covering the viewport. Each fragment (ray) solves the respective linear system using *conjugate gradient* [PTVF95]. Then, the intersection  $r$  is projected onto the polynomial. If the distance between the projection  $\gamma$  and  $r$  is smaller than a threshold, the intersection of the ray with the surface has been found to be  $r$ . Otherwise, the intersection of the ray and the local approximation is calculated. If the intersection is inside the ball with its center in the original sample point  $p_i$  and with radius  $b$ , we write this intersection into a float texture in order to use it in the next iteration.

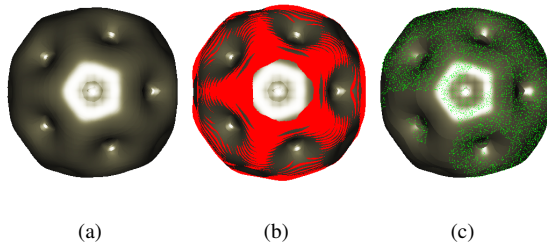
The next iteration starts in the step ‘intersection’ where, from the second iteration on, we check if the result of the last iteration (i.e. the result of ‘solve linear system and find projection’) is already the intersection with the PSS, in which case no further processing is done in any of the following steps. Otherwise, if the result of the last iteration is a valid intersection point within the ball defined by  $p_i$  and  $b$ , the following steps are performed using this intersection as the new point  $r$ . If not, we find the nearest intersection, after the current  $r$ , between the local approximations and the ray by means of depth tests as in the first iteration, killing all fragments with depth less or equal the depth of the current  $r$ .

As reported by Adamson and Alexa [AA03b], two to three iterations are needed to find all intersecting points between the primary rays and the PSS. It is important to note that in the case of calculating the intersection between the PSS and a set of rays that are not consistently oriented as the primary

rays (e.g. secondary rays) the search for the intersection in step ‘intersection’ must follow another strategy. We used a naive brute force scheme, checking all local polynomial approximations for each ray (fragment). This can be performed using nested loops and/or multiple rendering passes. Since in this case it is not possible to perform the range queries as described above, we propose to use this first intersection as a rough approximation of the intersection between the ray and the PSS for secondary reflected, refracted, and shadow rays. Thus, for these secondary rays none of the four remaining steps is performed.

## 6 Results

In this section we will present rendering and performance results of the methods we proposed. All tests were carried out on a standard PC with a 3.4 GHz processor, 2GB of RAM and an NVidia 6800 GT graphics card. The size of the viewport used for the performance measurements was  $512^2$ .



**Figure 2:** The Bucky Ball dataset. (a) The final result of applying the predictor-corrector method. (b) The points projected by the predictor at a distance greater than a pre-defined error are shown in red (see colorplate). (c) The output points from the predictor projected by the corrector at a distance greater than the error are shown in green.

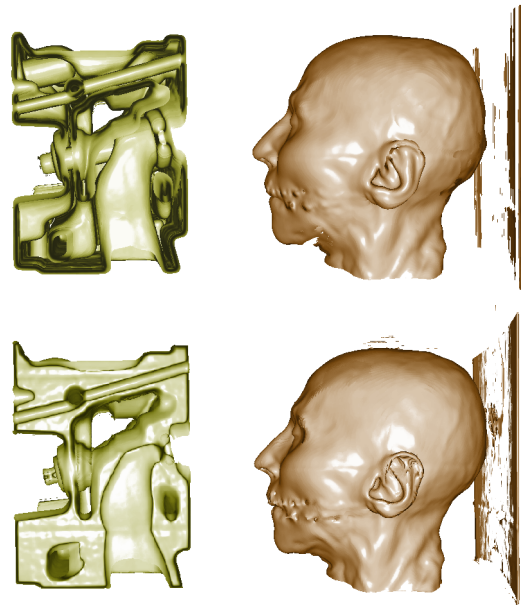
In Table 1 we present the results obtained for the extraction of surfaces from volumetric data performed using only the predictor step and the extraction performed using one iteration of the predictor and one iteration of the corrector steps. As can be noticed in the table, the corrector step adds a significant overhead in the processing time. Although this step improves the accuracy of the result, for interactive applications where precision is not important, the predictor suffices to generate an already good approximation to the PSS. This fact is depicted in Figure 2 where the effect of the predictor and the corrector steps on the input points (fragments) is shown. The predictor step projects a significant percentage of the points at a distance greater than a pre-defined error we set to test this effect. On the other hand, although the effect of the corrector step over the points already projected by the predictor is reduced to a small amount of points, this further projection could be important for applications where precision is the main concern.

Dataset	Size (voxels)	Predictor [fps]	Predictor-Corrector [fps]
Cadaver Head	$256^2 \times 154$	0.95	0.02
Engine	$256^2 \times 110$	1.16	0.03
Fuel	$64^3$	3.57	0.08
Bucky	$32^3$	7.16	0.17

**Table 1:** Performance in frames per second for the PSS extraction from volumetric data method.

Dataset	Size (points)	Ray-tracing PSS	
		(2 it.) [fps]	(3 it.) [fps]
Stanford Bunny	35947	6.62	5.30
Horse	48485	5.40	4.00
Skeleton Hand	109108	2.38	1.17

**Table 2:** Performance in frames per second for the PSS ray-tracing (2 iterations and 3 iterations) method.



**Figure 3:** PSS extracted for the Engine and the Cadaver Head datasets using the gradient magnitude (top) and iso-values (bottom).

For the tests performed with the ray-tracing algorithm we used a single reflection secondary ray and a depth of 2. The results of these tests are shown in Table 2. The use of secondary rays in our implementation is currently limited by the lack of a proper data structure for performing range queries efficiently on the GPU. Therefore, the inclusion of such a data structure is of major importance.

Nevertheless, the results obtained for both approaches are

promising considering the complexity of the computations involved. Although the implementation for extracting PSS from volumetric data is not interactive for the predictor-corrector case, the processing time is considerably low in relation to the large amount of fragments projected. Also, the renderings are of considerable quality as shown in Figures 3, 4 and 5.

Figure 3 shows renderings of surfaces extracted with our method from the Engine and Cadaver Head volumes. For generating the surfaces located at regions with high gradient magnitude, we projected only those fragments for which the square of the gradient, normalized between 0 and 1, was greater than 0.5. For the iso-surfaces, we discarded the fragments for which  $|v - f(p_i)|$  did not surpass the threshold.

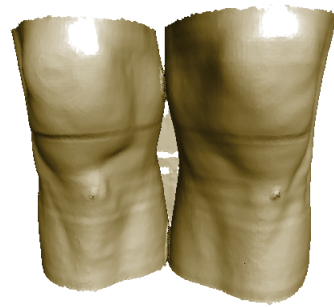


**Figure 4:** GPU-based ray-tracing of PSS for the Horse point cloud.

For the case of point clouds, ray-tracing a PSS on the CPU could be prohibitively slow, whilst we achieved up to 6.62 fps for the Stanford Bunny using our GPU implementation with 2 iterations. In the case of 3 iterations the frame rate dropped to 5.3 fps for the Bunny. However, 2 iterations suffice to generate a high-quality rendering as shown in Figure 4. This advantage could be exploited for accurate rendering of surfaces modeled as point clouds extracted from different sources, like medical data. For instance, the vertices of a surface mesh generated from MRI data by means of traditional methods, such as marching cubes or surface reconstruction techniques applied over the result of a segmentation algorithm, could be used as input to our ray-tracing algorithm. This will provide a smooth noise-free rendering of the iso-surface extracted. Figure 5 shows an example of this process for the Knee dataset, where we dropped the topology of an iso-surface mesh generated with marching and ray-traced the PSS defined by its vertices.

## 7 Conclusion and Future Works

The advantages of PSS over other surface representations for point clouds are clear, namely the fact that the surface



**Figure 5:** PSS for the vertices of an iso-surface mesh extracted from the Knee dataset with marching cubes.

is 2-manifold, the ability to deal with low-frequency noise and the locality of the computations. However, PSS strategies are not limited to cloud of points as proven in this work, where we employ PSS strategies to model three-dimensional surfaces directly from volumetric data based on a predictor-corrector strategy. Our method was completely implemented on the GPU, achieving relatively low computation times for the volumes tested with the predictor step. When the predictor-corrector approach is used, although the computation time increases, more accurate results are obtained. The decision of whether using the faster predictor or the more accurate predictor-corrector approach will depend of the application.

There are at least three possible improvements to this method. The first is related to the volumetric data classification, where we intend to test classification strategies based on *convolutions* and *wavelets* schemes. The second improvement involves the minimization strategies to handle very noisy data and sharp corners. The last aspect is the development of more efficient weighting functions, capable of handling non-homogeneous sampling densities.

We also developed a GPU-based ray-tracing algorithm for rendering PSS from point clouds, which achieves interactive frame rates for up to 100000 points. The most important problem found in this case was the cost of the range queries. The lack for an efficient GPU-based data structure to store the points in our implementation forced us to span the computation along multiple rendering passes for the primary rays and limit the use of secondary rays. We plan to include such a structure in our implementation in the near future in order to accelerate the computations and in order to be able handle secondary and shadow rays efficiently.

## Acknowledgments

This work was partially supported by the German International Exchange Service (DAAD), with grant number A/04/08711, and the State of São Paulo Research Foundation (FAPESP), with grant number 04/10947-6.

## References

- [AA03a] ADAMSON A., ALEXA M.: Approximating and intersecting surfaces from points. In *Proc. of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2003), Eurographics Assoc., pp. 230–239. 2
- [AA03b] ADAMSON A., ALEXA M.: Ray tracing point set surface. In *Shape Modeling International* (2003), IEEE Computer Society, pp. 272–282, 299. 2, 4, 5
- [ABCO\*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proc. of IEEE Visualization* (2001), IEEE Computer Society, pp. 21–28. 1, 2
- [ABCO\*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics* 9, 1 (2003), 3–15. 1, 2
- [AK04a] AMENTA N., KIL Y. J.: Defining point-set surfaces. *ACM Transactions on Graphics* 23, 3 (2004), 264–270. 1, 2, 4
- [AK04b] AMENTA N., KIL Y. J.: The domain of a point set surfaces. *Eurographics Symposium on Point-based Graphics 1*, 1 (2004), 139–147. 1, 2, 3
- [CHJ03] CO C. S., HAMANN B., JOY K. I.: Iso-splatting: A point-based alternative to isosurface visualization. In *Proceedings of the Eleventh Pacific Conference on Computer Graphics and Applications - Pacific Graphics 2003* (2003), pp. 325–334. 2
- [DG04] DEY T. K., GOSWAMI S.: Provable surface reconstruction from noisy samples. In *Proceedings of the twentieth annual symposium on Computational Geometry* (2004), ACM Press, pp. 330–339. 1
- [DGS05] DEY T. K., GOSWAMI S., SUN J.: *Extremal Surface Based Projections Converge and Reconstruct with Isotopy*. Tech. Rep. OSU-CISRC-05-TR25, Ohio State University, 2005. 2
- [DS05] DEY T. K., SUN J.: An adaptive MLS surface for reconstruction with guarantees. In *Proc. of Eurographics Symposium on Geometry Processing* (2005), pp. 43–52. 2
- [FCOS05] FLEISHMAN S., COHEN-OR D., SILVA C. T.: Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics* 24, 3 (2005), 544–552. 2
- [FGS05] FENCHEL M., GUMBOLD S., SIEDEL H.-P.: Dynamic surface reconstruction from 4D-MR images. In *Proc. of Vision, Modeling and Visualization* (2005), p. electronic version. 2
- [KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics* 28, 6 (2004), 801–814. 1
- [KKH01] KNISS J., KINDLMANN G. L., HANSEN C. D.: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *IEEE Visualization* (2001), pp. 255–262. 1
- [Lev98] LEVIN D.: The approximation power of moving least-squares. *Mathematics of Computation* 67, 224 (1998), 1517–1531. 2
- [Lev03] LEVIN D.: Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization* (2003), 37–49. 2
- [LS81] LANCASTER P., SALKAUSKAS K.: Surfaces generated by moving least squares methods. *Mathematics of Computation* 37, 155 (1981), 141–158. 3
- [LT04] LIVNAT Y., TRICOCHÉ X.: Interactive point-based isosurface extraction. In *VIS 04: Proceedings of the conference on Visualization 04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 457–464. 2
- [MAVdF05] MEDEROS B., AMENTA N., VELHO L., DE FIGUEIREDO L. H.: Surface reconstruction from noisy point clouds. In *Proc. of Eurographics Symposium on Geometry Processing* (2005), pp. 53–62. 1
- [Mic06] MICROSOFT CORPORATION: DirectX 9 SDK. <http://www.microsoft.com/directx>, 2006. 2, 3
- [PTVF95] PRESS W., TEUKOLSKY S., VETTERLING W., FLANNERY B.: *Numerical Recipes in C*, second edition ed. Cambridge University Press, 1995. 3, 4, 5
- [RJT\*05] REUTER P., JOYOT P., TRUNZLER J., BOUBEKEUR T., SCHLICK C.: Surface reconstruction with enriched reproducing kernel particle approximation. In *Eurographics Symposium on Point-Based Graphics* (2005), Eurographics Assoc., p. electronic version. 2
- [SFS05] SCHEIDEGGER C. E., FLEISHMAN S., SILVA C. T.: Triangulating point set surfaces with bounded error. In *Proc. of Eurographics Symposium on Geometry Processing* (2005), Eurographics Assoc., pp. 63–72. 2
- [SP97] SADARJOEN I. A., POST F. H.: Deformable surface techniques for field visualization. *Eurographics Computer Graphics Forum* 16, 3 (1997), C109–C116. 2
- [ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3D: an interactive system for point-based surface editing. In *SIGGRAPH : Proc. of Computer Graphics and Interactive Techniques* (2002), ACM Press, pp. 322–329. 1, 2