Anisotropic Sampling for Differential Point Rendering of Implicit Surfaces

Florian Levet[1] Julien Hadim[1] Patrick Reuter[1,2] Christophe Schlick[1]

[1] LaBRI - CNRS - INRIA - University of Bordeaux [2] LIPSI - ESTIA
{levet,hadim,preuter,schlick}@labri.fr

ABSTRACT

In this paper, we propose a solution to adapt the *differential point rendering* technique developed by Kalaiah and Varshney to implicit surfaces. Differential point rendering was initially designed for parametric surfaces as a two-stage sampling process that strongly relies on an adjacency relationship for the samples, which does not naturally exist for implicit surfaces. This fact made it particularly challenging to adapt the technique to implicit surfaces. To overcome this difficulty, we extended the *particle sampling* technique developed by Witkin and Heckbert in order to locally account for the principal directions of curvatures of the implicit surface. The final result of our process is a *curvature driven anisotropic sampling* where each sample "rules" a rectangular or elliptical surrounding domain and is oriented according to the directions of maximal and minimal curvatures. As in the differential point rendering technique, these samples can then be efficiently rendered using a specific shader on a programmable GPU.

Keywords: *Geometric Modeling, Implicit Surfaces, Differential Geometry, Point Rendering*

1 Introduction

Implicit surfaces are an elegant surface representation to model 3D surfaces without explicitly having to account for topology issues. Moreover, it is possible to develop a complete modeling-animating-rendering pipeline with almost no topological constraints by using ray-tracing to render the corresponding surfaces. Unfortunately, to be able to provide a decent rendering of implicit surfaces at interactive framerates, there is usually no other choice than to convert them into polygonal meshes, which inherently reintroduces heavy topological constraints.

In 2001, Kalaiah and Varshney [17, 16] proposed an innovative technique to render parametric surfaces in higher quality. The basic idea of their *differential point rendering* technique is to generate a discrete sampling of the parametric surface, where each sample locally defines the differential geometry (i.e. the position, the tangent plane, as well as the minimal and maximal direction of curvature). All the samples are individually rendered without any connectivity information by point rendering using a specific rectangular "splatter" accounting for the local differential geometry. This splatter can then be efficiently rendered by specific shaders on a programmable GPU (see also recent work of Botsch et al. [5]).

The differential point rendering technique is particularly well adapted to parametric surfaces since the samples can be generated explicitly using the local differential geometry, and the technique can be simply extended to triangular meshes by estimating the differential geometry at the mesh vertices.

Since the differential point rendering technique offers high quality rendering of surfaces using less surface samples, it is quite appealing to extend it for implicit surfaces as well. This was the initial motivation of the work we present here. Unfortunately, the extension is not as straightforward as it may appear at first glance. The main concern is that the sampling technique proposed by Kalaiah and Varshney is basically a two-stage process. In the first stage, a relatively dense point sampling of the surface is computed, either by direct sampling of the parameter space (in the case of a parametric surface), or by using the existing vertices (in the case of a triangular mesh). Then, during the second stage, many of the samples are removed using a simplification process that accounts for the local differential geometry, i.e. keeping more samples in the

Conference proceedings ISBN 80-903100-7-9
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

directions of high curvature and less samples in the directions of low curvature. The final result is a *curvature driven anisotropic sampling* where each sample "rules" a rectangular or elliptical surrounding domain, locally oriented according to the directions of maximal and minimal curvatures. The problem is, that this second stage is heavily based on the adjacency relationship between the samples. This relationship naturally exists in a triangular mesh or in a regular sampling of a parametric surface, but it has no natural counterpart for an implicit surface. As a consequence, even if it can theoretically be implemented, the two-stage process proposed by Kalaiah and Varshney is not well adapted to implicit surfaces.

The goal of this paper is to present an alternative solution for implicit surfaces to obtain a curvature driven anisotropic sampling that offers similar properties as the one generated by Kalaiah and Varshney. Our solution is based on a particle system, in the spirit of the one initially proposed by Witkin and Heckbert [31], but we also account for anisotropic sampling using local differential geometry.

The remainder of the paper is organized as follows: Section 2 presents some related previous work mainly dealing with sampling techniques for implicit surfaces. Section 3 details our new anisotropic sampling technique for implicit surfaces. Section 4 presents several experimental results that focus on the visual quality and the convergence speed. Section 5 concludes and presents some directions we are currently studying. Finally, the mathematical background to compute the principal curvature directions of an implicit surface as well as the corresponding curvature amounts is given in the Appendix.

2 Previous work

Since the groundbreaking work done in 1985 by Levoy and Whitted [20] which was revisited by Grossman and Dally in 1998 [13], point rendering has become a popular research domain in recent years. But as our goal is to adapt the differential point rendering technique to implicit surfaces, it is out of the scope of this paper to recall all existing point rendering techniques. We refer the interested reader to a recent overview [18] and tutorial [1].

We will thus focus more precisely on sampling techniques for implicit surfaces. Basically, existing work can be divided into two main families: tessellating techniques and particle system techniques.

2.1 Tessellating implicit surfaces

The tessellating techniques of implicit surfaces can themselves be divided into three categories. First, *spatial sampling techniques* subdivide the 3D space into cells, commonly either cubes or tetrahedra, and search for the cells that intersect the implicit surface. One of the most commonly known spatial sampling techniques is the marching cubes algorithm [32, 21], that divides the 3D space into cubic cells, and triangles are generated according to the sign at the corners of the cells. Unfortunately, there are ambiguous configurations that have to be resolved [12]. Marching tetrahedra algorithms, e.g. by Shirley and Tuchman [23] or Hall and Warren [14], further divide the cubic cells into tetrahedra, and for each tetrahedra there are no ambiguous configurations. Nevertheless, marching tetrahedra algorithms create numerous, often over distorted triangles. In both the marching cubes and the marching tetrahedra algorithms, the cells are of constant size, so all these techniques may miss small features, do not adapt to the local geometry of the implicit surface, and require knowledge about the topology of the implicit surface. When the cell size is too small, an excessive number of polygons may be produced, and when it is too large, details may be obscured. To overcome this drawback, adaptive subdivision techniques that converge to the surface recursively have been developed [3], but with such techniques cracks may occur between triangles of adjacent cells of different size.

The second category are *surface fitting techniques* that create a seed mesh that roughly approximates the implicit surface and progressively adapt and deform it to better fit the implicit surface. For example, Velho [29, 30] starts with a coarse polygonal approximation of the surface and subdivides each polygon recursively according to the local curvature. But still there must be some a priori knowledge about the topology of the surface since the coarse polygonal approximation has to capture the correct topology of the implicit surface. Other surface fitting techniques either assume special classes of implicit surfaces created from skeletal elements [10, 7], or rely on a search for critical points [28, 6] suffering from inefficiency for complex implicit surfaces.

The third category are *surface tracking techniques* (or *continuation techniques*) that start from a seed element on the surface and iteratively grow a polygonal mesh that approximates the implicit surface. Cellular surface tracking techniques [2, 32, 4] start from a cell that intersects the implicit surface and iteratively find all intersecting cells among its neighbors. Since the cells are of constant size, cellular surface tracking techniques suffer from the same drawbacks as non-adaptive spatial sampling techniques, and furthermore, in the general case, it can be difficult to determine a seed cell.

2.2 Particle systems

The second way to sample implicit surfaces is to use so-called *particle systems* that evenly distribute samples over the implicit surface. The first time that particles were used to sample and control a surface was in a complete modeling tool using oriented particle systems by Szeliski and Tonnesen [24], but in their work there was no underlying implicit surface. Turk [26] used a similar process in order to generate textures using a reaction-diffusion method. Like the particle system, he used repulsion radii to simulate this reaction-diffusion, but the distribution of the particles is uniform and does not adapt to the local curvatures. Figueiredo et al. introduced a system to sample implicit surfaces using particles [9]. Even if the force applied to the particles is not the same as the one used by Turk, the authors used the relaxation process of Turk in order to achieve a uniform distribution of the points. Then these points are used to compute a polygonal approximation of the implicit surface. Turk's reaction-diffusion method can also be used to re-tile polygonal surfaces [27] according to the curvature with more points in regions of high curvature. But the method only takes into account isotropic curvature information and does not consider the principal directions of curvature of the surface.

Witkin and Heckbert [31] developed a powerful modeling application by putting together some of these existing ideas. Indeed, they demonstrated that particle systems are useful in order to both display and control implicit surfaces. They used two different types of particles: *floaters* that lie on the surface and that are used for rendering, and *control points* that are used to deform the surface. These two types of particles must resolve a set of constraints so that the floaters follow the implicit surface and that the surface follows the control points. Moreover, the authors defined an adaptive repulsion and a split/death condition so that particles could either split or disappear from the surface. Hart et al. [15] improved upon this particle system for automatic and numerical differentiation of the implicit surfaces. Indeed, in the work of Witkin and Heckbert, the derivatives for complex models can become computation-demanding and error-prone. Moreover, *shape adapters* were introduced that simplify surface deformations. One main limitation of both works is that no information about the curvature of the implicit surface is taken into account, thus only uniform distribution can be generated with spherical particles that all have the same repulsion radius.

Crossno and Angel [8] derived another extension based on the work of Witkin and Heckbert that can be used to sample an isosurface extracted from a 3D density image. A trilinear interpolation between the eight vertices of the voxel surrounding the particle location is used to approximate the implicit function. They use the same repulsion forces and movement calculation as in [31], but they estimate the repulsion radius of each particle depending on the curvature at the sample. Consequently, particles in regions of higher curvature have a smaller repulsion radius. Nevertheless, Crossno and Angel only account for isotropic curvature information and thus do not consider principal directions of curvature.

Finally, Pauly et al. [22] used a particle system in order to simplify point-sampled surfaces. The same linear force as in [27] is used and the distribution of points depends on the curvature of a moving least squares (MLS) surface that approximates the points. Using a death condition combined with the repulsion forces enables them to simplify the number of points of the surface. Again, they only account for isotropic curvature information and do not consider principal directions and curvatures.

Some particle systems have also been used to polygonize implicit surfaces [9] via a Delaunay triangulation. Again, care must be taken that the particles are dense enough to create a topologically correct polygonal mesh.

3 Anisotropic sampling

Recall that our goal is to generate an anisotropic sampling technique for implicit surfaces that offers similar properties as the one generated by Kalaiah and Varshney for parametric surfaces. The density of the sampling should be related to the local curvatures as well as to account for the directions of maximal and minimal curvatures. In other words, each sample should be the center of an elliptical domain oriented along these directions and sized according to the maximal and minimal radius of curvature. To reach this goal, we propose to adapt the particle sampling technique developed by Witkin and Heckbert [31].

The basic idea of the sampling algorithm is outlined in Algorithm 1. In this section, we detail the choice we made for every step involved in this algorithm.

Algorithm 1 The basic idea of our algorithm.

Require: An implicit surface
  Create a set of particles lying on the surface
  while Convergence is not reached **do**
    Compute the repulsion radii of the set of particles
    Compute the repulsion forces of the set of particles
    Update the position of the particles
    Split particles when necessary
  end while

3.1 Initial set of particles

Actually, thanks to the particle splitting step, the algorithm converges even when starting with one sin-

gle initial particle, but it is more efficient to have an initial set of particles covering the surface. The easiest way to reach this is to use a scheme similar to the *shrink-wrap* technique [28]: first regularly sample either the bounding sphere or the bounding box of the implicit surface and then migrate the resulting particles by following the gradient of the implicit function until the surface is reached. Note that the diameter of this bounding volume is used as a normalization scaling factor during the entire process, so that every distance (radius, curvature, migration) can be computed in a scale-independent manner.

3.2 Repulsion radii

At each step of the particle migration loop, the repulsion radius for each particle has to be calculated. As stated above, we want an anisotropic repulsion process where particle are repelling more in directions of low curvature and less in directions of high curvature. So we actually compute two repulsion radii per particle (for the directions of maximal and minimal curvatures, respectively) by adapting the computation given in [17] to implicit surfaces. The mathematical background to compute the principal curvature directions of an implicit surface as well as the corresponding curvature amounts is given in the Appendix. Note that this calculation allows us to determine the variation of the implicit field of the implicit surface. The values that are found are not some distance measurements of the curvature that could be used directly in the application. Indeed, one has to multiply these values by a coefficient in order to scale them and to be able to use them as distance measurements that will define a repulsion domain around each particle. More specifically, $minCurv$ and $maxCurv$ will yield the curvature amounts in the two principal directions $minCurvDir$ and $maxCurvDir$.

3.3 Repulsion forces

This step of the algorithm differs significantly from the rest of the literature. In [31, 8, 15], the authors compute the repulsion forces between the particles by using an energy measure. Even if it works well for isotropic repulsion, this process cannot be generalized for anisotropic repulsion. Another way to compute repulsion forces between particles has been proposed in [27, 22]. Again, the computation was proposed for isotropic repulsion between circular particles, but in contrary to the previous one, it is possible to extend this scheme for anisotropic repulsion between elliptical particles.

More precisely, an elliptical particle can be defined by combining the two repulsion radii $minRadius$ and $maxRadius$ and the two orthogonal directions of curvature. By adding a radius in the third direction, we actually define ellipsoidal particles instead of elliptical ones. We have done this modification because in 3D space, it is much easier to compute repulsion forces between ellipsoids than between 2D ellipses defined on two different planar domains. Note that the radius given for this third direction (let us call it the *height* of the particle) does not really matter: we have tested either by using a small constant value for each particle, to get almost flat ellipsoids, or by using $minRadius$ again, to get particles with a circular section, but the final sampling obtained after the particle migration process is very similar. The main reason is that the centroids of neighboring ellipsoids lie almost on the same plane during the last iteration steps, therefore the repulsion forces are more or less orthogonal to the height direction canceling the influence of the particle's height.

Once the ellipsoidal shapes of the particles have been set up, the repulsion forces can be computed according to the algorithm given below. It is important to note that we do not have to compute the repulsion force between any pair of particles. We rather use a space partitioning scheme that avoids computing the force between two particles that belong to distant areas. We use the same scheme as in [31], but any other hierarchical partitioning should work fine. Space partitioning reduces the overall complexity from $O(n^2)$ to $O(n \ln n)$ and thus significantly speeds up the computation involved in each step of the migration process.

3.3.1 Computing repulsion forces by using spherical coordinates

We use Algorithm 2 in order to compute the repulsion force.

Algorithm 2 Calculating the repulsion force between two particles.

Require: Two particles with their respective curvature information
Compute vector $\mathbf{r}_{ij} = \mathbf{p}_j - \mathbf{p}_i$ between the centers of the particles i and j
Comute the intersection $\mathbf{m}_i$ of $\mathbf{r}_{ij}$ and the ellipsoid of i
Compute the intersection $\mathbf{m}_j$ of $\mathbf{r}_{ij}$ and the ellipsoid of j
Determine whether the two ellipsoids intersect themselves
Compute their repulsion force.

This algorithm is really efficient as it only requires computing two line/ellipsoid intersections.

The intersection of the two ellipsoids can then be calculated. Starting from the two intersection points $\mathbf{m}_i$ and $\mathbf{m}_j$:

$$res = \|\mathbf{m}_i\| + \|\mathbf{m}_j\| - \|\mathbf{r_{ij}}\| \qquad (1)$$

A negative res means that the two particles i and j do not intersect and thus do not apply forces to each other.

Otherwise, the repulsion force of the particle j applied on the particle i is defined by:

$$F_{ij}(i) = res * (\mathbf{p}_i - \mathbf{p}_j) \quad (2)$$

We use the same linear repulsion force as in [27, 22] because of its compact radius of support. The total force exerted on i is then given by

$$F(i) = \sum_{j \in N_p} F_{ij}(i), \quad (3)$$

where N_p is the neighborhood of i that can be retrieved by the spatial partitioning.

3.4 Migration of the particles

After we have computed the repulsion forces for all the particles, the next step of the algorithm is to move the particle according to its repulsion force. Since the repulsion force $F(i)$ of a particle is a vector, we just have to add this vector to the position of the particle in order to find its new position: $\mathbf{p}_i$ += $kF(i)$. The constant k defines the rigidity of the particle's reaction with respect to the forces that are applied on it. In our implementation we use a constant value, but ideally k should be proportional to the average distance from a particle to its neighbors. Note that after this movement the particle does not exactly lie on the surface anymore, and we have to apply a step of the Newton-Raphson method to glue the particle on the surface: $\mathbf{p}_i$ -= $f(\mathbf{p}_i)\mathbf{n}_i$, where $f(\mathbf{p}_i)$ is the value of the implicit function f at the particle i and $\mathbf{n}_i$ is the normal at the new position of particle i.
Once we have the final position of the particle, we have to compute its normal one last time for rendering.

3.5 Determining the fate of the particles

The final step of the algorithm is to determine whether the particle has to be split. We have seen that a good condition is to subdivide a particle when the sum of the norms of the forces that are applied on it becomes lower than a predefined threshold. Indeed, as the forces of repulsion are applied on a finite radius around the particles, the fact of having a particle with few forces applied means that it is in an under-sampled region. It is then natural to divide it in order to increase the local sampling density.

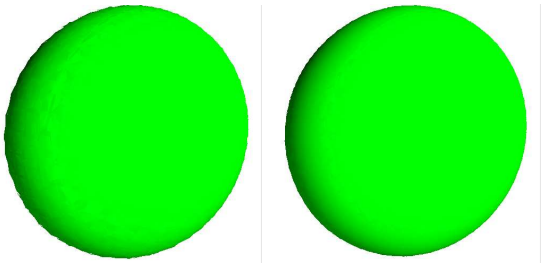
3.6 Rendering

After that the characteristics of the ellipsoids for all the particles i have been created, we use them in order to create the differential points that are rendered as fragment-shaded rectangles. Since current graphics hardware does not support curved primitives as differential points, we make use of the possibility of directly programming new primitives in the GPU of these graphics cards.
More precisely, we use a rectangular primitive to represent a particle according to the local differential geometry. Similar to [17], the rectangle is defined in the tangent plane of the particle where the normal and the two perpendicular curvature directions define a local coordinate system.
The rectangle's extent is computed according to the maximum and minimum curvature amounts. Consequently, the higher the surface curvature, the smaller is the rectangle. In order to render the rectangle as a piece-wise smooth surface, an adequate normal distribution of the rectangle is required. In contrast to Kalaiah and Varshney, who select the best fitting normal out of 256 precomputed normals according to the principal curvatures, we interpolate the normals at the corners of the differential point's rectangle using *vertex shaders* and *fragment shaders*. Since we know the underlying implicit surface, we assign the normal to each of the four vertices of the rectangle. In other words, we define a normal field over the rectangle that locally approximates the appearance of the smooth implicit surface. The fragment shader interpolates the normals for each fragment and normalizes them by using a *cube map texture*.
The rectangles are fragment-shaded using the programmable graphics pipeline with vertex and fragment programs. In the vertex program, we do not compute the shading since we want per-pixel lighting. We write the normal, light and half vector in texture registers in order to interpolate them and define a normal distribution in screen-space. Then, in the fragment program, we normalize the interpolated normal, light and half vector and shade the fragment with both diffuse and specular components according to the Phong shading model. Finally, the set of these fragment-shaded rectangles gives a visual impression of a smooth surface.

4 Experimental results

All the images of this section were produced on a Pentium IV at 3.0 GHz with 1 GB of main memory and an NVidia GeForce Quadro FX. No code optimization effort has been done, the only acceleration technique is the spatial subdivision in order determine the particles that are in the compact support radius of another particle. We implemented the differential point rendering using Cg's vertex and fragment shaders on a NV30 chipset graphics board from NVidia.
The only parameter that a user can modify is the scaling factor of the curvature amounts. Note that this factor must be chosen carefully. Indeed, the differential

point rendering allows having a smooth rendering of an implicit surface with a reduced number of points. Nevertheless, when the repulsion radii are too large, each differential point covers a too large amount of the surface resulting in a lower rendering quality. Figure 1 underlines this problem. Indeed, Figure 1(a) shows an ellipsoid rendered with a large repulsion radii. Artefacts appear at the silhouette of the surface as well as in the shaded regions compared to Figure 1(b), where the ellipsoids are rendered with a smaller repulsion radii and thus reducing the artefacts.



(a) Large repulsion radii (b) Small repulsion radii

Figure 1: Differential point rendering

As explained in Section 3.6, differential point rendering renders the surface as a collection of overlapping fragment-shaded rectangles. In Figure 2, a random color has been used for each rectangle in order to outline the underlying structure. Figure 3(a) presents the rabbit rendered with a constant diffuse material but it should be noted that once the size and orientation of the rectangles have been defined, any fragment-shader can be used. As an example, Figure 3(b) shows a non-photorealistic rendering that is obtained by simply changing the shaders.

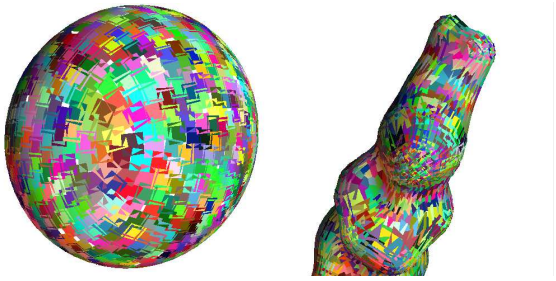
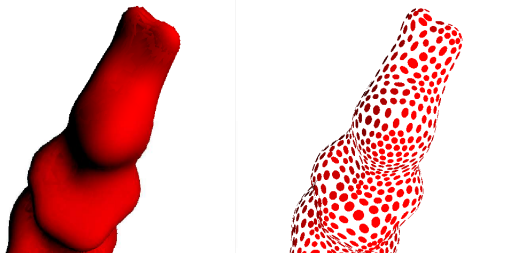


Figure 2: The differential points are rendered with random colors to outline the overlapping.

The principal problem of particles systems concerns the convergence detection: even if the surface seems to be well sampled, particles can still be created due to the splitting criterion of the particle system. Indeed, a slightly moving particle can lead to a change in the splitting criterion and thus to the generation of a new particle. This does not have a high impact on the rendering of the surface, but it shows that the convergence



(a) Constant diffuse material (b) Non-photorealistic rendering

Figure 3: Different renderings of the rabbit

of particle systems is critical. A simple example of this problem is given in Figure 4.

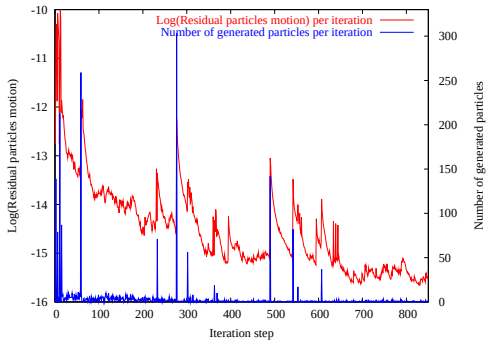


Figure 4: Residual particles motion and number of particles created per iteration.

Figure 4 underlines the fact that some particles are created even when the surface is well sampled. For example, Figure 4 shows that between iteration number 600 and iteration number 800, only 5 to 10 particles are created. So a quasi-equilibrium state of the system has been reached as soon as iteration number 200, but, because of the particle system behaviour, a small number of particles continues to split. Another evidence that lead us to believe that the equilibrium state has been reached is that, even if some particles have been created, the residual motion the particles decreases after a higher number of iterations. This means that the surface is well sampled and that the new particles only have a small influence on the motion of other particles. In order to resolve this problem, we are currently working on a more robust convergence criterion based on the residual motion of the particles instead of thresholding the forces.

5 Conclusion

In this paper, we presented an adaptation of the differential point rendering to implicit surfaces by anisotropically sampling the implicit surfaces using a particle system. Linking differential point rendering with particle systems provides an elegant way to sample and

render implicit surfaces. By pushing the information of the local differential geometry into each sample, we can describe the surface using fewer particles. This is particularly beneficial for remote rendering applications with limited bandwith.
Another contribution of our work is the mathematical background that we provide to compute the principal curvature directions of an implicit surface as well as the corresponding curvature amounts (cf. Appendix).
One drawback of the current implementation concerns the definition of a robust convergence criterion: in the case of spherical particle systems, an energy criterion can be used to identify the convergence. Unfortunately, energy criteria work well for isotropic particle systems (using spheres), but cannot be easily adapted for anisotropic systems (using ellipsoids).
We believe that the convergence time and the number of iterations of the relaxation process can be significantly reduced by first doing a global approximation step of the sampling and then running the particle system in a second step. More precisely, the global sampling step should determine the number of particles to sample the surface, and the second step determines the position of the particles on the surface.
Finally, we believe that particles systems are perfectly suited for interactive implicit surface modeling. As a consequence, the next step of our application will be to allow the user to deform the surface while the particles are following the surface deformation.

References

[1] M. Alexa, M. Gross, M. Pauly, H. Pfister, M. Zwicker, and M. Stamminger. Point based computer graphics. In *SIGGRAPH Course Notes*, 2004.

[2] E. Allgower and S. Gnutzmann. Simplicial pivoting for mesh generation of implicitly defined surfaces. *CAGD*, 8(4):305 – 325, 1991.

[3] J. Bloomenthal. Polygonization of implicit surfaces. *CAGD*, 5(4):341–355, 1988.

[4] J. Bloomenthal. An implicit surface polygonizer. *Graphics Gems IV*, pages 324–349, 1994.

[5] M. Botsch, M. Spernat, and L. Kobbelt. Phong splatting. In *Symposium on Point-based Graphics 2004*, pages 25–32, 2004.

[6] A. Bottino, W. Nuij, and K. van Overveld. How to shrinkwrap through a critical point. In *Proc. of Implicit Surfaces '96*, pages 53–73, 1996.

[7] B. Crespin, P. Guitton, and C. Schlick. Efficient and accurate tessellation of implicit sweep objects. In *Proc. of Constructive Solid Geometry '98*, 1998.

[8] P. Crossno and E. Angel. Isosurface extraction using particle systems. In *IEEE Visualization '97*, pages 495–498, 1997.

[9] L. de Figueiredo, J. Gomes, D. Terzopoulos, and L. Velho. Physically-based methods for polygonization of implicit surfaces. In *Proc. of Graphics Interface '92*, pages 250–257, 1992.

[10] M. Desbrun, N. Tsingos, and M.P. Cani. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Computer Graphics Forum*, 15(5), 1996.

[11] M. DoCarmo. *Differential Geometry of curves and surfaces*. Prentice-Hall, 1976.

[12] A. Van Gelder and J. Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, 1994.

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

[14] M. Hall and J. Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics & Applications*, 10(6):33–42, 1990.

[15] J. Hart, E. Bachta, W. Jarosz, and T. Fleury. Using particles to sample and control more complex implicit surfaces. In *Proc. of Shape Modeling International 2002*, pages 129–136, 2002.

[16] A. Kalaiah and A. Varshney. Modeling and rendering of points with local geometry. *Trans. on Visualization and Computer Graphics*, 9(1):30–42, 2003.

[17] Aravind Kalaiah and Amitabh Varshney. Differential point rendering. In *Proc. of Eurographics Workshop on Rendering 2001*, pages 139–150, 2001.

[18] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. *Computer & Graphics*, 2004.

[19] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proc. of ACM SIGGRAPH 96*, pages 313–324, 1996.

[20] M. Levoy and T. Whitted. The use of points as display primitive. Technical Report TR 85–022, University of North Carolina at Chapel Hill, 1985.

[21] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (ACM SIGGRAPH 87 Proc.)*, 21(4):163–169, 1987.

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

[23] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, 1990.

[24] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (Proc. of ACM SIGGRAPH 92)*, 26(2):185–194, 1992.

[25] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proc. of ICCV'95*, pages 902–907, 1995.

[26] G. Turk. Generating textures for arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proc. of ACM SIGGRAPH 91)*, 25(4):289–298, 1991.

[27] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics*, 26(2):55–64, 1992.

[28] K. van Overveld and B. Wyvill. Shrinkwrap: an adaptive algorithm for polygonizing an implicit surface. Technical Report 93/514/19, University of Calgary, 1993.

[29] L. Velho. Adaptive polygonization made simple. In *Proc. of SIBGRAPI '95*, pages 111–118, 1995.

[30] L. Velho. Simple and efficient polygonization of implicit surfaces. *Journal of Graphics Tools*, 1(2):5–25, 1996.

[31] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proc. of ACM SIGGRAPH 94*, pages 269–278, 1994.

[32] B. Wyvill, Craig McPheeters, and Geoff Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

Appendix: Principal curvature directions of an implicit surface

In the initial differential point rendering technique [17, 16], Kalaiah and Varshney proposed to extract the principal directions of curvature from parametric surfaces [11], triangular meshes [25], or NURBS surfaces that are fit to triangular meshes [19]. In this appendix, we show how to extract the principal directions of curvature for a point $\mathbf{p} = [x, y, z]^T$ on the implicit surface $\mathcal{S}$, i.e. $\mathbf{p} \in \mathcal{S}$. To this end, consider the defining function $f : \Re^3 \rightarrow \Re$ (that has second order partial derivatives) of the implicit surface $\mathcal{S} = \{\mathbf{x} \in \Re^3 : f(\mathbf{x}) = 0\}$. Recall, that the normal $\mathbf{n}$ of a point $\mathbf{p}$ is defined by the non-zero gradient of the defining function

$$\mathbf{n} = \nabla f(\mathbf{p}) = \frac{\partial f}{\partial x}(\mathbf{p}), \frac{\partial f}{\partial y}(\mathbf{p}), \frac{\partial f}{\partial z}(\mathbf{p}) \quad .$$

In order to derive second-order local geometry for the determination of the principal directions of curvature, we require the *Hessian matrix* $\mathbf{H}$ of the second derivatives of the defining function f:

$$\mathbf{H} = \begin{pmatrix} \frac{\partial \mathbf{n}}{\partial x}(\mathbf{p}) \\ \frac{\partial \mathbf{n}}{\partial y}(\mathbf{p}) \\ \frac{\partial \mathbf{n}}{\partial z}(\mathbf{p}) \end{pmatrix} = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(\mathbf{p}) & \frac{\partial^2 f}{\partial x \partial y}(\mathbf{p}) & \frac{\partial^2 f}{\partial x \partial z}(\mathbf{p}) \\ \frac{\partial^2 f}{\partial x \partial y}(\mathbf{p}) & \frac{\partial^2 f}{\partial y^2}(\mathbf{p}) & \frac{\partial^2 f}{\partial y \partial z}(\mathbf{p}) \\ \frac{\partial^2 f}{\partial x \partial z}(\mathbf{p}) & \frac{\partial^2 f}{\partial y \partial z}(\mathbf{p}) & \frac{\partial^2 f}{\partial z^2}(\mathbf{p}) \end{pmatrix}$$

Note that $\mathbf{H}$ is symmetric because of the equality of mixed partials. We use a local parameterization to extract the principal directions and curvatures on a point $\mathbf{p} \in \mathcal{S}$ with an associated normal $\mathbf{n}$ and a Hessian matrix $\mathbf{H}$. For the illustration of the following calculations, consider Figure 5.
Let us now approximate the defining function f of the implicit surface $\mathcal{S}$ in a small vicinity of $\mathbf{p}$ by using a small vector $\mathbf{w}$ for a second degree Taylor expansion with an approximation error $o(\|\mathbf{w}\|^2)$:

$$f(\mathbf{p} + \mathbf{w}) = f(\mathbf{p}) + \mathbf{n}^T \bullet \mathbf{w} + \frac{1}{2}\mathbf{w} \bullet (\mathbf{H}\mathbf{w}) + (\|\mathbf{w}\|^2)$$

Since $\mathbf{p} \in \mathcal{S}$, the defining function of the implicit surface $\mathcal{S}$ in $\mathbf{p}$ is $f(\mathbf{p}) = 0$, and we find

$$f(\mathbf{p} + \mathbf{w}) = \mathbf{n}^T \bullet \mathbf{w} + \frac{1}{2}\mathbf{w} \bullet (\mathbf{H}\mathbf{w}) + o(\|\mathbf{w}\|^2).$$

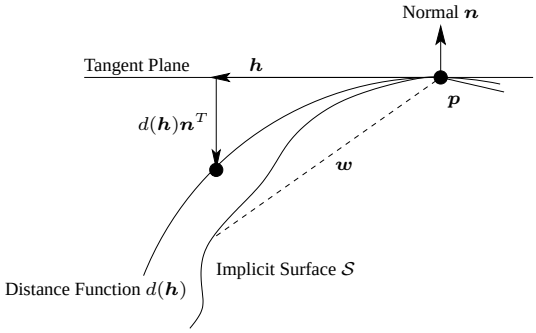


Figure 5: Curvature of an implicit surface $\mathcal{S}$.

Now, we split vector $\mathbf{w}$ into a vector $\mathbf{h}$ that is orthogonal to $\mathbf{n}$, i.e. $\mathbf{n}^T \bullet \mathbf{h} = 0$, and a distance function d to the tangent plane in $\mathbf{p}$: $\mathbf{w} = \mathbf{h} + d(\mathbf{h})\mathbf{n}^T$. We want to determine the distance function d so that $f(\mathbf{p} + \mathbf{h} + d(\mathbf{h})\mathbf{n}^T) = 0$. By combining the two previous equations and setting $d(\mathbf{h}) = o(\mathbf{h})$ since $d'(\mathbf{0}) = 0$ by definition, we find

$$\mathbf{n}^T \bullet (\mathbf{h} + d(\mathbf{h})\mathbf{n}^T) + \frac{1}{2}\ (\mathbf{h} + d(\mathbf{h})\mathbf{n}^T) \bullet\ \mathbf{H}(\mathbf{h} + d(\mathbf{h})\mathbf{n}^T) = o(\|\mathbf{h}\|^2),$$

that we develop to

$$\mathbf{n}^T \bullet (\mathbf{h} + d(\mathbf{h})\|\mathbf{n}\|^2 + \frac{1}{2}\ \mathbf{h} \bullet (\mathbf{H}\mathbf{h})\ + \frac{1}{2}\ \mathbf{h} \bullet\ \mathbf{H}d(\mathbf{h})\mathbf{n}^T$$
$$+\frac{1}{2}\ d(\mathbf{h})\mathbf{n}^T \bullet (\mathbf{H}\mathbf{h})\ + \frac{1}{2}\ d(\mathbf{h})\mathbf{n}^T \bullet (\mathbf{H}d(\mathbf{h})\mathbf{n}^T)\ = o(\|\mathbf{h}\|^2).$$

Since $\mathbf{n}^T \bullet \mathbf{h} = 0$, and since we can neglect the constant term with respect to $\mathbf{h}$, we find

$$d(\mathbf{h})\|\mathbf{n}\|^2 + \frac{1}{2}\ \mathbf{h} \bullet (\mathbf{H}\mathbf{h})\ + d(\mathbf{h})\ \mathbf{h} \bullet (\mathbf{H}\mathbf{n}^T)\ = o(\|\mathbf{h}\|^2),$$

and the second order approximation of d is

$$d(\mathbf{h}) = -\frac{\mathbf{h} \bullet (\mathbf{H}\mathbf{h})}{2\|\mathbf{n}\|^2}.$$

Now, we want to find the maximum and minimum of $\mathbf{h} \bullet (\mathbf{H}\mathbf{h})$ for $\mathbf{h}$ orthogonal to $\mathbf{n}$. This implies that the derivative of $\mathbf{h} \bullet (\mathbf{H}\mathbf{h})$ has a component orthogonal to $\mathbf{n}$ with the value 0, and hence $\mathbf{H}\mathbf{h} = \mu\mathbf{h}^T + \lambda\mathbf{n}^T$. To determine the principal directions and curvatures, we have to find the eigenvectors with associated non-zero eigenvalues of

$$\mathbf{H}\mathbf{h} - \frac{(\mathbf{H}\mathbf{h}) \bullet \mathbf{n}^T}{\|\mathbf{n}\|^2}\mathbf{n}^T = \mathbf{I} - \mathbf{n}^T \bullet \mathbf{n}\ \ \mathbf{H}\ \ \mathbf{I} - \frac{\mathbf{n}^T \bullet \mathbf{n}}{\|\mathbf{n}\|^2} \quad .$$

Summing up, the principal curvature amounts $u_\mathbf{p}$ and $v_\mathbf{p}$ are the non-zero eigenvalues of this latter matrix, and the principal directions $\mathbf{u_p}$ and $\mathbf{v_p}$ are given by the corresponding eigenvectors.