Eurographics Symposium on Rendering 2009
Hendrik Lensch and Peter-Pike Sloan
(Guest Editors)

Volume 28 (2009), Number 4

Dart Throwing on Surfaces

D. Cline[1], S. Jeschke[1], K. White[2], A. Razdan[1] and P. Wonka[1]

[1]Arizona State University, USA [2]Corda, USA

Abstract

In this paper we present dart throwing algorithms to generate maximal Poisson disk point sets directly on 3D surfaces. We optimize dart throwing by efficiently excluding areas of the domain that are already covered by existing darts. In the case of triangle meshes, our algorithm shows dramatic speed improvement over comparable sampling methods. The simplicity of our basic algorithm naturally extends to the sampling of other surface types, including spheres, NURBS, subdivision surfaces, and implicits. We further extend the method to handle variable density points, and the placement of arbitrary ellipsoids without overlap. Finally, we demonstrate how to adapt our algorithm to work with geodesic instead of Euclidean distance. Applications for our method include fur modeling, the placement of mosaic tiles and polygon remeshing.

1. Introduction

A *Poisson disk* point set is a uniformly distributed set of points in which no two points are closer to each other than some minimum distance $2r$. Such point sets are said to be *maximal* if no more points can be added to them without violating the minimum distance constraint. Because of their good statistical properties, Poisson disk point sets are ideal candidates for a number of sampling contexts, and are particularly useful in Monte Carlo integration. Applications of Poisson disk point sets on surfaces include converting a surface to a point-based representation, surface retriangulation, the distribution of data points such as irradiance samples over surfaces and the creation of stipple patterns. Poisson disk point sets are also particularly well suited to applications where the locations of points will be visible, and a number of natural phenomena exhibit Poisson disk-like behavior. For example, the dimples on an orange, the arrangement of pores or hairs on skin, and the coloration of certain kinds of fish and birds, all mimic Poisson disk distributions.

A large body of work exists to generate Poisson disk patterns or similar distributions on the plane. While such a planar distribution can be transferred to a 3D surface by means of a parameterization, in general this introduces distortion. Practically speaking, the distortion causes the point set to lose some of its good spacing characteristics.

The contribution of this paper is a set of optimized dart throwing (ODT) algorithms to *directly* generate Poisson disk point sets on surfaces. The algorithms are fast, between ten and twenty times faster than Turk's method [Tur92]. Our method is also easier to implement as it does not require moving points over the surface. It does not require a parameterization, nor mesh connectivity in the case of Euclidean spacing. A second contribution of the paper is to extend ODT to different surface types, including spheres, Bezier patches, subdivision surfaces and implicits. We also extend our method to handle different kinds of spacing not considered by most point placement algorithms: sphere distributions with randomized density and non-overlapping ellipsoids. A third contribution of this work is to define a fast dart throwing method based on geodesic distance. Taken together, our dart throwing algorithms provide a good solution for many different point placement problems.

2. Previous Work

Planar Methods. Cook [Coo86] was the first to suggest the use of Poisson disk sampling patterns in computer graphics. Noting the exorbitant cost of making such patterns by dart throwing, Cook advocated stratified samples as an alternative. Later, Lloyd [Llo87] described a simple relaxation procedure that could distribute points evenly, and Mitchell et al. [Mit91] removed the minimum distance requirement to produce the best candidate algorithm.

Because of the difficulty of creating very large, well distributed point sets, tile-based approaches were developed that reuse smaller point sets within a larger context. Cohen et al. [CSHD03] showed how Wang tiles can be used to tile the plane seamlessly with well spaced points. Kopf et

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

al. [KCODL06] take this idea even further, tiling the plane with point distributions at multiple resolutions. Other work uses non-rectangular tilings as a scaffolding on which to build a point set [ODJ04, Ost07].

Recently, a number of *optimized dart throwing* (ODT) algorithms have been developed that get around the problems of naive dart throwing. They do this by excluding covered parts of the sampling domain from dart throwing. Dunbar and Humphreys [DH06] encode the empty region around a point set as "scalloped sectors", which can be sampled according to area. Jones [Jon06] uses a Voronoi diagram to keep track of empty spots in the sampling domain. White et al. [WCE07] employ a quadtree structure to exclude covered areas from dart throwing. Bridson [Bri07] describes a similar, but less rigorous method. Wei [Wei08] performs dart throwing in parallel on the GPU, and Feng et al. [FHHJ08] distribute ellipses in the plane using stratified seed points that are relaxed using a variant of Lloyd's algorithm.

Points on Surfaces. Turk [Tur92] uses a variant of Lloyd relaxation defined for polygon meshes. The method starts by placing points randomly on the surface. The points then repel each-other, eventually reaching a uniform distribution. Alliez et al. [AECdVDI03] generate seed points for a similar algorithm by error diffusion on a triangle mesh. Surazhsky et al. [SAG03] produce variable density samples on a mesh by constructing a weighted centroidal Voronoi tesselation of the surface. Witkin and Heckbert [WH05] directly sample implicit surfaces with *floater* particles that roam freely over the surface, repelling each-other.

Nehab and Shilane [NS04] create stratified point sets on triangle meshes by subdividing the space around the mesh with an octree, placing a single sample in each octree leaf node. Rovira et al. [RWCS05] define a set of well distributed "global lines" in space and then intersect those lines with a surface to produce a point set. Li et al. [LLLF08] generalize the concept of Wang tiles to surfaces. Once computed, the tiling can be used to quickly distribute points evenly over the surface, but non-uniform densities are not demonstrated. To overcome distortion in the tiling, Lloyd's method is applied to the initial point distribution made by the tiling.

Geodesic Distance Methods. Some applications, particularly remeshing, prefer points that are well spaced according to *geodesic distance* (shortest path on the surface) rather than Euclidean distance. Coming from the level set literature, Sethian [Set96] cast the geodesic distance problem as a solution to the Eikonal equation, which he solved by fast marching. Later work [KS98] [SV01] extended fast marching to compute geodesics on triangle meshes and parameterized surfaces (geometry images). Peyré and Cohen [PC06] use the fast marching method to sample surfaces, placing points successively at the surface location that is furthest away from peviously placed points.

Surazhsky et al. [SSK*05] and Mitchell et al. [MMP87] present comparatively fast algorithms to compute exact and approximate geodesics on triangle meshes. Fu and Zhou [FZ08] combine this method with boundary sampling [DH06] to calculate Poisson disk point sets directly on triangle meshes. Weber et al. [WDB*08] approximate geodesic distance very quickly on the GPU using a fast sweeping algorithm. However, the method only works on geometry images, and does not distribute points on the surface.

3. Optimized Dart Throwing on Surfaces

This section describes our algorithm for creating Poisson disk point sets on surfaces based on Euclidean distance. Our algorithm draws inspiration from a number of other dart throwing methods, and it is most similar to [WCE07]. However, our technique performs dart throwing in 3D and for a number of different surface types. In order to sample a given surface, our algorithm requires the following:

1. A definition for small pieces or *fragments* of the surface.
2. A method to choose a fragment according to area.
3. A method to uniformly sample points on a fragment.
4. A method to determine if a fragment is covered by a dart.
5. A method to split surface fragments into subfragments.

The algorithm itself works directly with the surface definition, and it does not require moving of points on the surface or a parameterization.

3.1. Overview

Our optimized dart throwing algorithm begins by dividing the surface into fragments that meet the above requirements. For example, the triangles of a triangle mesh serve as the fragments, and a NURBS surface can be decomposed into Bezier patch fragments. These initial surface fragments are placed in an "active" list to prepare for dart throwing.

To throw a dart, the algorithm selects an active fragment F with probability proportional to its area. It then chooses a random point p on the fragment and checks to see if it meets the minimum distance requirement with respect to the current point set. If the minimum distance requirement is met, the algorithm adds p to the point set. In any case, the algorithm then checks to see if F is completely covered by any point from the point set. If F is covered, it is discarded; otherwise, we split it into a number of child fragments and add the uncovered fragments back to the active fragment lists. Dart throwing terminates when there are no more active surface fragments.

4. The Triangle Mesh Case

This section describes optimized dart throwing in the specific case of triangle meshes and Euclidean distance. In this case, the triangles themselves are the surface fragments. A triangle can be split into four children by introducing new vertices at edge midpoints. It is also a simple matter to test whether a triangle is covered by a sphere–check to see if all the vertices are inside the sphere. Furthermore, the area of a

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

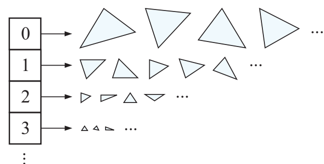

Figure 1: *Logarithmic binning allows the algorithm to choose a surface fragment in constant time on average.*

triangle can easily be computed as one half the length of the cross product of two of its sides: $\|(V_1 - V_0) \times (V_2 - V_0)\|/2$. Finally, sampling a triangle uniformly with respect to area can be done by generating a random barycentric coordinate and returning the corresponding point on the triangle.

Logarithmic binning of active triangles. To make the formulation complete, we must choose an active triangle according to area. A standard way to organize the active triangles would be to place them in a balanced search tree, yielding a $\log n$ search to choose a triangle. In this work, however, we adopt a different strategy based on *logarithmic binning* and rejection sampling that allows us to choose a triangle in constant time on average, given reasonable assumptions on the distribution of triangle areas (i.e. the ratio fo the maximum to minimum triangle area is less than 2^{64}) [WCE07]. The idea behind logarithmic binning is to place all of the active triangles in bins according to area, with each bin spanning a small range of areas, as shown in Figure 1. In our implementation, a bin includes triangles from some minimum area B_{min} to a maximum $B_{max} = 2B_{min}$. A triangle with area A_t would be placed in bin $\lfloor \log_2(A_{max}/A_t) \rfloor$, where A_{max} is the maximum area of triangles from the original mesh.

To choose a triangle, we select a bin with probability proportional to the total area in the bin using a linear search, starting at the first non-empty bin. We then choose a triangle within the bin based on rejection sampling (e.g. pick a triangle at random within the bin and then accept it with probability A_t/B_{max}, where once again A_t is the triangle's area, and B_{max} is the maximum area of triangles assigned to the bin. Repeat until a triangle is accepted.) Triangles in the bin will invariably be at least half as large as B_{max}, so the acceptance rate of the rejection sampling is at least fifty percent, and triangle selection will run in constant time on average.

Overlap testing. An important part of making dart throwing efficient is to optimize minimum distance testing–checking thrown darts and area fragments for overlap against the current point set. We employ a spatial grid hash table to limit the number of overlap tests that must be made. A kd-tree could also be used for this purpose.

Termination in practice. To ensure that the algorithm will terminate, we discard triangles fragments that are too small (< 1/10000th the area of the smallest triangle in the original mesh). This takes care of the hypothetical case in which three dart boundaries meet at one point, as well as small imprecision issues related to floating point numbers.

Performance on triangle meshes. Figure 2 plots the point generation speed of our dart throwing algorithm on three different triangle meshes, compared with the relaxation method described in [Tur92]. Our algorithm is consistently more than an order of magnitude faster than Turk's method with 40 iterations. Note also that we must process the triangles individually, so the algorithm must be at least $O(M+N)$, where M is the number of triangles in the mesh and N is the number of points in the point set. In the timings this shows up as a startup cost related to the mesh size. Our method also fares well against the stratified method in [NS04], with speed between about 0.5 and 2 times as fast, depending on the subdivision settings. This should not be surprising, since similar triangle subdivisions are the bottleneck in both algorithms.

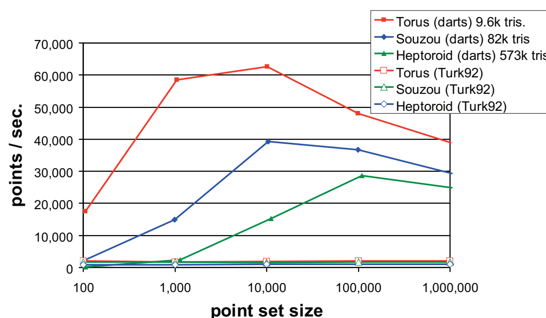

Figure 2: *Points placed per second for different triangle meshes and point set sizes using our optimized dart throwing (darts), and Turk's method with 40 iterations (Turk92).*

5. Other Surface Types

One of the strengths of our algorithm is that its hierarchical structure naturally extends to sampling different surface types. In addition to triangles, we have implemented our dart throwing algorithm on the unit sphere, Bezier patches, Loop subdivision surfaces and implicits (see Figure 3). With the exception of the sphere, we are the first to perform dart throwing directly on these surfaces. Here we describe the modifications to the basic algorithm needed to handle the additional surface types.

5.1. The Unit Sphere

The unit sphere is an important special surface, and it is therefore appropriate that we adapt our ODT algorithm to it. Possible applications of Poisson disk point sets on the sphere include positioning virtual lights for ambient occlusion, and BRDF sampling.

We define the area fragments on a sphere to be *spherical rectangles*, regions bounded by angular extent as shown in Figure 3. The area of a patch can be calculated as an integral over its angular extents, (θ_0, θ_1, ϕ_0 and ϕ_1):

$$\int_{\phi_0}^{\phi_1} (\theta_1 - \theta_0) \sin\phi \, d\phi \;=\; (\theta_1 - \theta_0)(\cos\phi_0 - \cos\phi_1). \quad (1)$$

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

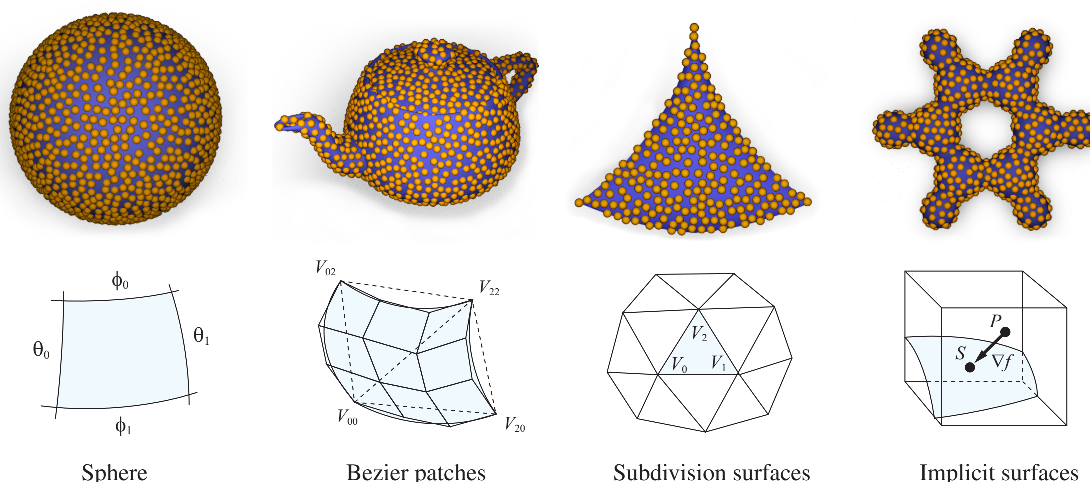

Figure 3: *In addition to triangle meshes, optimized dart throwing can be done directly on a number of other surface types. The figure shows examples of the different surface types (top row), and surface fragment definitions (bottom row).*

As with triangles, spherical patches can be selected proportional to area by logarithmic binning. Choosing a point uniformly on the surface of the patch can be performed by inverting equation 1. We generate two random numbers, r_1 and r_2, uniformly in $[0,1)$ and transform them to angles on the sphere (θ,ϕ) using the equations $\theta = (1-r_1)\theta_0 + r_1\theta_1$, and $\phi = \cos^{-1}((1-r_2)\cos\phi_0 + r_2\cos\phi_1)$.

Testing to see if a spherical patch is covered by a sample point can be done by checking to see if the four corners of the patch lie within the minimum distance sphere around the point. This test is always valid except for patches closer to the poles than the dart radius, where the curvature of the sides of the patch exceeds the curvature of the minimum distance sphere surrounding a sample point.

5.2. Bezier Patches

Optimized dart throwing can also directly sample Bezier patches and NURBS, which can be decomposed into Bezier patches. A Bezier patch is a tensor product surface defined by a control mesh that forms the convex hull of the patch, as shown in Figure 3. A patch can be subdivided into four children whose union is the parent patch by several applications of the de Casteljau algorithm. Furthermore, as a patch is subdivided the control polygons of the subpatches asymptotically approach the surface.

No closed form expression exists for the area of a Bezier patch, but it is still possible to sample its surface according to area by rejection sampling. In this scheme, instead of storing the actual area of the patch, we store a upper bound on the area, $s'_{max}\ t'_{max}$, where s'_{max} and t'_{max} are bounds on the magnitudes of the s and t derivatives over patch, P_s and P_t. The algorithm chooses a patch on which to sample according to the area bounds and then selects a point with a random parametric coordinate (u,v). Rejection sampling then thins out the distribution to be uniform with respect to area. The acceptance rate of the rejection sampling for point (u,v) in parameter space is $\frac{\|P_s(u,v)\times P_t(u,v)\|}{s'_{max}\ t'_{max}}$. Note, however, that in this case the algorithm must choose a new patch on which to place the next point each time a sample is rejected.

The exact algorithm just described is rather slow due to the rejection sampling and the need to calculate surface derivatives. In practice, it is better to approximate the area of the patch as the area of the two triangles defined by the patch corners. Darts can then be thrown uniformly in parameter space, leaving the minimum distance criteria to even out the resulting point distribution.

5.3. Subdivision Surfaces

Subdivision surfaces possess a hierarchical structure that fits nicely into our optimized dart throwing framework. We have implemented our algorithm for Loop subdivision surfaces, although Catmull-Clark subdivision surfaces could be handled in essentially the same manner.

The natural area fragment in a Loop subdivision surface is the 1-ring of a triangle in the control mesh (the triangle plus all vertices connected to it). This 1-ring contains all of the information needed to reconstruct the limit surface for the triangle. In order to make optimized dart throwing work with a subdivision surface, the algorithm must be able to subdivide all of the triangles in the control mesh individually. Consequently, we store the 1-ring of each triangle separately, replicating vertices as needed.

Like the control mesh of a Bezier patch, the 1-ring of a triangle forms a convex hull of the limit surface corresponding to the triangle. However, this bound is not particularly

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

tight, so in practice we create an approximate bound composed of the three vertices of the triangle along with their corresponding limit points. [WP04] define an exact bound that could also be used.

The limit surface area for a mesh triangle can be approximated using the area of the triangle itself, and we can approximate a unifiorm distribution on the limit surface with a uniform distribution over the current control mesh. To throw a dart, we choose a mesh triangle with probability proportional to area, then we select a random point on the triangle and project it to the limit surface. This dart location can then be tested against the current point set. Finally, the triangle 1-ring is tested against the point set for coverage, and is either discarded or subdivided based on the results of this test.

5.4. Implicit Surfaces

Implicit surfaces do not possess a natural hierarchical structure, which makes placing points on them challenging. Our solution is to use a spatial hierarchy instead of a hierarchy of surface elements. Here the active surface fragments are replaced by octree cells that surround the level set. To begin with, we subdivide space into a coarse grid and place those voxels that contain the surface on the active list of fragments. To throw a dart, the algorithm selects a voxel with probability proportional to its surface area, picks a random point within the voxel, and projects it to the surface along the gradient direction, as shown in Figure 3. Voxels are split by octree subdivision, discarding those children that do not contain the surface. Testing a voxel for enclosure within a sphere can be done by either testing the corners of the voxel against the sphere, or by directly determining the distance from the center of the sphere to the corner of the voxel that is farthest away, d_{fc}:

$$d_{fc}^2 = (|x_p - x_c| + h)^2 + (|y_p - y_c| + h)^2 + (|z_p - z_c| + h)^2. \quad (2)$$

In the above expression $P = (x_p, y_p, z_p)$ is the center of the sphere, $C = (x_c, y_c, z_c)$ is the center of the voxel, and h is half of the voxel width [WCE07]. Note that in order to completely sample the surface, the algorithm must be able to reliably determine if the surface passes through a given voxel, for example by maintaining min/max isovalues within each voxel cell.

Poisson Sphere Distributions Although not part of our main topic, we mention here that ODT using octree cells can be adapted to create Poisson sphere distributions that fill space with non-overlapping spheres.

6. Variable Distributions

In order for a point generation routine to be an effective modeling primitive, it must be able to generate non-uniform distributions. In this section we describe two variations of our ODT algorithm that handle the placement of (1) spheres of varying size and (2) arbitary ellipsoids on surfaces.

6.1. Placing Spheres of Variable Size

A useful extension to our algorithm is to place spheres of variable size without overlap, enabling the algorithm to be used in a number of additional applications, such as variable density remeshing.

When a dart is thrown in a variable distribution, the algorithm must choose a dart radius as well as a position. To control the distribution of dart sizes, we store a minimum and a maximum dart radius, r_{min} and r_{max} at the triangle vertices. We then throw darts as follows:

1. Choose a triangle and dart location P uniformly.
2. Determine radius r_b of the largest sphere that will fit at P.
3. If the maximum sphere radius is greater than the minimum dart radius assign the dart a random radius between r_{min} and $\min\{r_{max}, r_b\}$. Otherwise discard the dart.
4. Discard the triangle if covered, or split it otherwise. To determine triangle coverage, expand nearby spheres by the minimum radius over the triangle, and check to see if they cover the triangle.

Figure 4 gives two examples of variable distributions produced by our system. In the left example, r_{min} and r_{max} vary over the surface, but take on the same value at any given point. In the right example, they are both constant over the surface, but have different values.

6.2. Placing Ellipsoids Without Overlap

This section extends the ideas presented in Section 6.1 to handle the placement of arbitrary ellipsoids on surfaces without overlap. To make ellipsoid placement work, we must define how the parameters of the ellipsoids will be set, find a suitable overlap test between ellipsoids, and estimate whether a given triangle can have any more ellipsoids placed on it without overlapping the current set.

Ellipsoid intersection testing. A number of methods exist to determine if two ellipsoids intersect (see for instance [WWK01, WCC*04]). These tests could be used, but they are rather slow for our application since we may require dozens of overlap tests per accepted dart. To handle the high

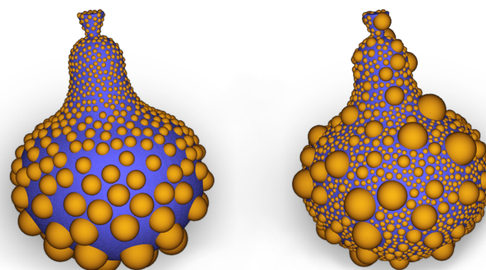

Figure 4: *Spatially varying and random radius Poisson disk point sets.*

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

number of overlap tests required by our method, we have designed an approximate ellipsoid overlap test which in the worst case is 10 times faster than the algebraic test described by [WWK01] (2.14 million vs. 196 thousand tests/sec.). In addition, our test has a number of early outs that make it even faster for most cases. The test is exact when both ellipsoids are spheres, or when they have common orientation and aspect ratio, and it is fairly accurate otherwise.

Conceptually, the test detects an intersection by expanding each ellipsoid in turn, checking to see if it encloses the center of the other ellipsoid (see Figure 5). Let A be an axis of one of the ellipsoids. The amount by which we expand A is found by projecting the axes of the other ellipsoid onto A and taking the L_2 norm of the projections. For example, suppose that we want to test ellipsoid $\mathbf{E_1}$ with center P and axes $\{A_p, B_p, C_p\}$ against ellipsoid $\mathbf{E_2}$ with center Q and axes $\{A_q, B_q, C_q\}$. We first expand $\mathbf{E_1}$ to create the ellipsoid $\mathbf{E}'_1$ with center P and axes:

$$A'_p = \left(1 + \frac{\sqrt{(A_p \cdot A_q)^2 + (A_p \cdot B_q)^2 + (A_p \cdot C_q)^2}}{A_p \cdot A_p}\right) A_p \tag{3a}$$

$$B'_p = \left(1 + \frac{\sqrt{(B_p \cdot A_q)^2 + (B_p \cdot B_q)^2 + (B_p \cdot C_q)^2}}{B_p \cdot B_p}\right) B_p \tag{3b}$$

$$C'_p = \left(1 + \frac{\sqrt{(C_p \cdot A_q)^2 + (C_p \cdot B_q)^2 + (C_p \cdot C_q)^2}}{C_p \cdot C_p}\right) C_p. \tag{3c}$$

Next we test to see if $\mathbf{E}'_1$ encloses point Q. This can be done by finding the distance between Q and P in a stretched space in which $\mathbf{E}'_1$ is a unit sphere:

$$d_e^2 = \frac{((Q-P)\cdot A_p)^2}{(A_p \cdot A_p)^2} + \frac{((Q-P)\cdot B_p)^2}{(B_p \cdot B_p)^2} + \frac{((Q-P)\cdot C_p)^2}{(C_p \cdot C_p)^2}. \tag{4}$$

In the stretched space, if $d_e < 1$ then $\mathbf{E}'_1$ contains Q; otherwise it does not. The two ellipsoids are considered to intersect if both of the expanded ellipsoids enclose the center of the other ellipsoid.

We also add a number of early outs that make our test faster and more accurate. First, if the center of $\mathbf{E_1}$, or the endpoints of any of its axes lie within $\mathbf{E_2}$ (or vice-versa) the ellipsoids must overlap. Also, if the centers of $\mathbf{E_1}$ and $\mathbf{E_2}$ are further apart than the sum of their semimajor axes, then the ellipsoids cannot intersect.

Figure 6 demonstrates the accuracy of our ellipsoid overlap test. The right figure shows the worst case scenario for our test–when a thin ellipsoid intersects another one at an angle. Even in this case, the test is accurate enough for placement without visual overlap. Of course, if total precision is required an exact test (using our early outs) can be used.

Determining triangle coverage. With the ellipsoid intersection test in place, we need to define a test for triangle coverage. This test must determine whether additional ellipsoids can be placed on a given triangle without overlap. In the case of spheres we determined triangle coverage by expanding neighboring points by a minimum radius, checking the expanded spheres to see if they cover the triangle. We adopt a similar approach for ellipsoids, constructing a minimum ellipsoid within the triangle, and expanding the neighboring ellipsoids by it to determine triangle coverage.

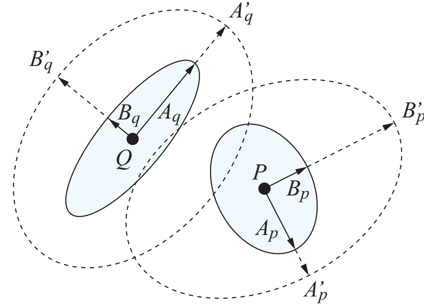

Figure 5: *Geometry for the ellipsoid overlap test.*

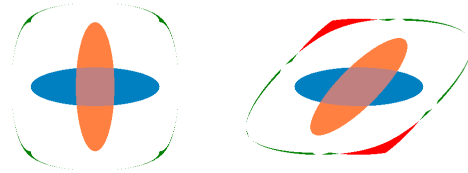

Figure 6: *Ellipsoid overlap test examples. The orange ellipsoid is translated relative to the blue one and overlap is tested. Green areas show false negatives, where our approximate method returns no overlap when overlap exists. Red areas show false positives.*

Given that $\mathbf{E}_{min}$ is the minimum ellipsoid allowed on the triangle, the triangle coverage test proceeds as follows: For each ellipsoid in the neighborhood of the triangle, $\mathbf{E}$,

1. Expand $\mathbf{E}$ by $\mathbf{E}_{min}$ and test to see if the expanded ellipsoid encloses the triangle vertices.
2. Expand $\mathbf{E}_{min}$ by $\mathbf{E}$, place the center of the expanded ellipsoid at the center of $\mathbf{E}$ and check the triangle vertices for enclosure once again.

The triangle is deemed to be covered if any of the expanded ellipsoids enclose all three of its vertices. Dart throwing is guaranteed to terminate as long as the minimum ellipsoid provided to the test can in fact be placed somewhere on the triangle. Figure 7 shows examples of ellipsoid distributions. Note that ellipsoids offer a number of interesting object placements that would not be possible with spheres alone.

Dart throwing in the ellipsoid case. Dart throwing in the ellipsoid case proceeds similar to the variable sphere case. The algorithm generates a desired ellipsoid, $\mathbf{E}$ along with a minimum allowed scale, s_{min}. It attempts to place $\mathbf{E}$ or scale it to fit in its current location. If the ellipsoid cannot be placed, the triangle on which the dart was thrown is tested for coverage using $\mathbf{E} * s_{min}$ as the minimal ellipsoid. Finally, we subdivide or discard the triangle as dictated by the coverage test.

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

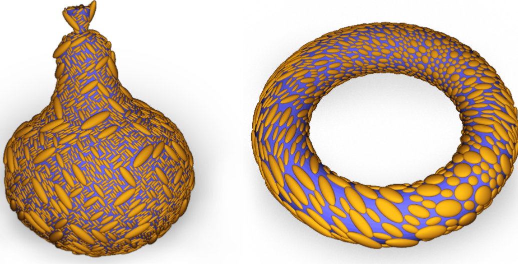

Figure 7: *Ellipsoid distributions generated by our system.*

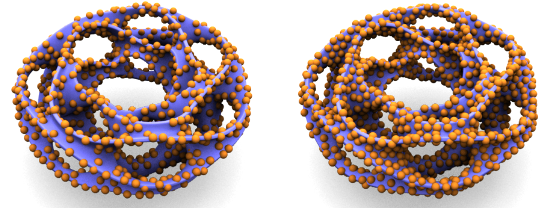

Figure 8: *Point sets created using Euclidean (left) and Geodesic (right) distance metrics.*

7. Geodesic Dart Throwing

While the Euclidean distance metric is often acceptable, some applications and problem setups require points to be distributed according to Geodesic distance (shortest distance on the surface). Figure 8 illustrates such a case. The test model contains a number of thin vanes, and samples on either side of a vane interfere with each-other in the Euclidean case, causing undersampling. On the other hand, geodesic dart throwing has no difficulties, and properly samples the surface.

In this section we present an optimized dart throwing algorithm for geodesic distance-based dart throwing. It has the characteristics that (1) every dart is accepted since all of the free surface area is explicitly tracked, (2) spatially varying density is supported, and (3) the algorithm allows for a time/quality tradeoff.

Distance propagation. We base the geodesic distance calculation in our algorithm on fast marching [Set96] with a spherical wavefront approximation to the geodesic distance. While not perfect, this approximation is generally accurate to within a few tenths of a percent of the exact distance [WDB*08]. Briefly, in fast marching, distance travels outward from a source point (dart) moving from vertex to vertex in the mesh. The spherical distance propagation rule that we use works as follows: Given a triangle with vertices X_0, X_1 and X_2, and distance values t_1 and t_2 at X_1 and X_2, we want to find the distance t_0 at X_0. This can be solved by projecting the triangle onto a plane, yielding vertices $\widehat{X}_0$, $\widehat{X}_1$ and $\widehat{X}_2$, as shown in figure 9. A virtual source point $\widehat{S}$ is then produced in the same plane, and t_0 is evaluated as $\|\widehat{X}_0-\widehat{S}\|$.

Projection onto the plane begins by defining three vectors, N, U and V, that serve as the triangle's coordinate system:

$$N=\frac{(X_2-X_1)\times(X_0-X_1)}{\|(X_2-X_1)\times(X_0-X_1)\|},\quad U=\frac{X_2-X_1}{\|X_2-X_1\|},\quad V=N\times U.$$

Letting $d_{12}=\|X_2-X_1\|$ and setting $\widehat{X}_1$ as the origin of the coordinate system yields the values

$$\widehat{X}_1=(0,0),\ \widehat{X}_2=(d_{12},0),\ \widehat{X}_0=((X_0-X_1)\cdot U,\ |(X_0-X_1)\cdot V|)$$

for the projected vertices. The location of the virtual source $\widehat{S}$ (a hypothetical point in the projected plane at distance t_1

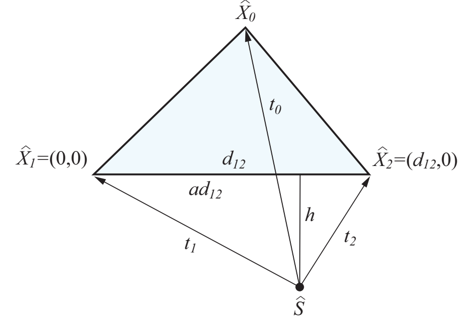

Figure 9: *Spherical wavefront propagation geometry.*

from $\widehat{X}_1$ and t_2 from $\widehat{X}_2$) can be computed as

$$\widehat{S}=(ad_{12},-h),$$

where

$$a=\frac{1}{2}+\frac{t_1^2-t_2^2}{2d_{12}^2}\quad\text{and}\quad h=\sqrt{t_1^2-a^2d_{12}^2}.$$

At this point t_0 can be tentatively set to $\|\widehat{X}_0-\widehat{S}\|$. However, the update is only valid if the computed value for t_0 is greater than both t_1 and t_2, and the x intercept of the segment between $\widehat{S}$ and $\widehat{X}_0$ lies between $\widehat{X}_1$ and $\widehat{X}_2$, or in other words

$$0\ \le\ x_s+\frac{h}{y_0+h}(x_0-x_s)\ \le\ d_{12},$$

where x_0, y_0 and x_s are x and y coordinates of $\widehat{X}_0$ and $\widehat{S}$. If either of these tests fails, or if $t_1^2-a^2d_{12}^2$ is negative, the update reverts to Dijkstra's approximation:

$$t_0=min(\ t_1+\|X_0-X_1\|,\ t_2+\|X_0-X_2\|\ ).$$

Triangle connectivity and initial subdivision. Unlike the algorithms presented so far, geodesic dart throwing requires the mesh connectivity. To avoid updating connectivity during dart throwing, we perform most of the mesh subdivision required by the algorithm as a preprocess. Sudivisions that take place during dart throwing are internal to individual triangles, and we do not update the connectivity for them.

Let s_{max} be the longest edge of a triangle, and r_{min} be the minimum dart radius specified at the vertices. In the initial subdivision, mesh triangles are split recursively on their longest edge until $r_{min}>\alpha s_{max}$, where $\alpha\in(0,2]$ is a user specified value that controls the subdivision rate. We then

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

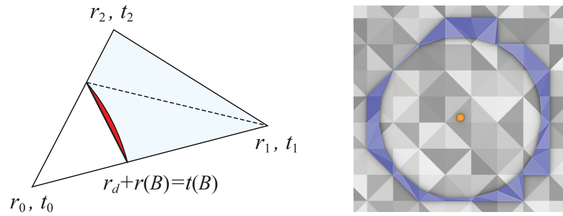

Figure 10: *(Left) Close-up of a triangle on the dart boundary. The triangle is clipped on the approximate dart boundary by a straight segment, producing 1 or 2 surviving subtriangles (shown in blue). The red region shows the error compared to the ideal boundary. (Right) The boundary for a dart thrown in our system (inner edge of the blue region).*

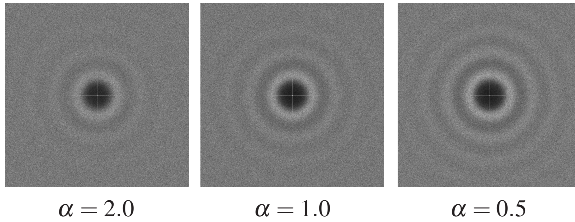

Figure 11: *Averaged Fourier transforms for geodesic dart throwing with different α values.*

remove T junctions in the mesh by further subdivision as needed and recalculate the mesh connectivity.

Subdivision during dart throwing. During dart throwing, we must track the uncovered area on each triangle. To do this, we create a list of *subtriangles* for triangles that are partially covered. Referring to figure 10, the dart boundary on a partially covered triangle is defined by the set of points B satisfying the equation $r_d + r(B) = t(B)$, where r_d is the radius of the originating dart, and $t(B)$ and $r(B)$ are the t value at B and the radius of a dart that would be thrown at B.

[FZ08] showed that in the case of variable density, the ideal dart boundaries are general conics. However, for efficiency reasons we approximate them by line segments, linearly interpolating the r and t values on the triangle edges to find the approximate dart border. The maximum error caused by this clipping procedure is $(2-\sqrt{4-\alpha^2/4})r$, or about 27% of the dart radius at $\alpha = 2.0$, 6.4% at $\alpha = 1.0$, and 1.6% at $\alpha = 0.5$. Figure 11 shows Fourier transforms for point sets created using different α values. The error caused by the dart boundary approximation manifests itself as a slight loss of power in the "blue noise rings", with $\alpha = 0.5$ being virtually indistinguishable from a standard dart throwing periodogram.

Dart throwing in the geodesic case. Conceptually, our geodesic dart throwing algorithm proceeds by placing a point on a free location, marching the geodesic distance (t) out to the dart boundary, and repeating. As in the Euclidean case, dart radius values are stored at the triangle vertices.

In practice, a dart is thrown by first choosing a triangle

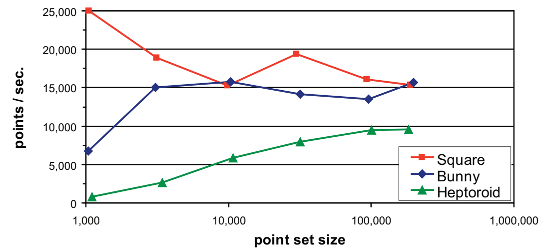

Figure 12: *Timings for the geodesic dart throwing algorithm ($\alpha = 1.25$). The graph shows points per second for different models and point set size.*

T uniformly with respect to its *uncovered* area. Once again, we use logarthmic binning for this purpose. Next, we pick a point uniformly on T, or one of its subtriangles if it is partially covered. The distances to the vertices of T are then computed, and the vertices of T are placed in a *vertex queue*. The fast marching routine then propagates t outwards to the dart boundary using the spherical wavefront update rule. Finally, completely covered triangles are flagged as such, and partially covered triangles and subtriangles are clipped. At no time during this process do we alter the connectivity of the mesh. Instead, each partially covered triangle stores a list of active subtriangles. All distance calculations are done on the vertices of the original mesh, and distances are transferred to the subtriangles by barycentric interpolation.

Geodesic dart throwing results. Figure 12 shows timings for the geodesic dart throwing algorithm. In general, runtime is linear in the total number of triangles processed after the initial subdivision rather than the number of points produced (about 300k to 450k triangles per second on our test machine). One competing method to our geodesic dart throwing is the direct placement algorithm of Fu and Zhao [FZ08]. They report sample generation times of less than 50 points per second, several orders of magnitude slower than our method. (These times include point relaxation and remeshing, but they state that the majority of the time is taken in the initial sample placement.) Our geodesic dart throwing method also compares favorably to the tile based method of Li et al. [LLLF08]. A direct comparison is difficult, but our algorithm is about three times faster than their fastest stated time for the the bunny model (15k vs. 5k points per second). This does not including the time they require to make the PolyCube map for the surface tiling.

8. Applications and Examples

In this section we demonstrate three applications of point sets made using our system: remeshing, hair placement and the modeling of mosaic tiles.

Remeshing. Remeshing polygonal objects allows a user to create models with more or less polygons than an input model, or regularize its triangulation. Because of its ability to quickly generate high quality point sets with a specified

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

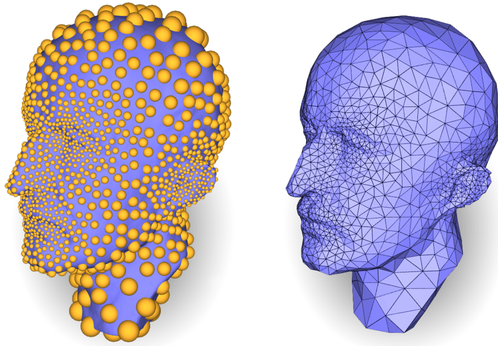

Figure 13: *(Left) Optimized dart throwing creates variable density points. (Right) A new mesh created from the points.*

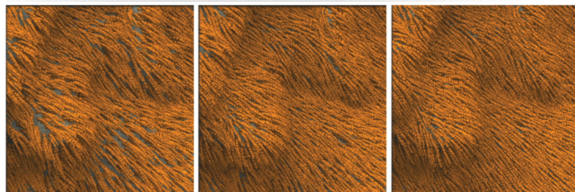

Figure 14: *(Top) A lion with 230k hairs placed by optimized dart throwing. (Bottom) From left to right, random, stratified, and ODT hair placement.*

density, optimized dart throwing is an good way to generate the vertices of a revised mesh. Figure 13 shows a remeshing example using ODT. For this example, we employed a user-specified density function to determine the vertex placements, but other density functions based on surface curvature [Tur92], mesh saliency [LVJ05] or edge feature extraction [FZ08] could easily be incorporated into the point generation process.

Hair and fur rendering. The current trend in hair and fur rendering for close shots is to instantiate individual hair fibers, which may require placing hundreds of thousands of hairs on an object. Our algorithm can quickly place a large number of well spaced hairs that provide even surface coverage and a smooth rendered appearance. Figure 14 shows

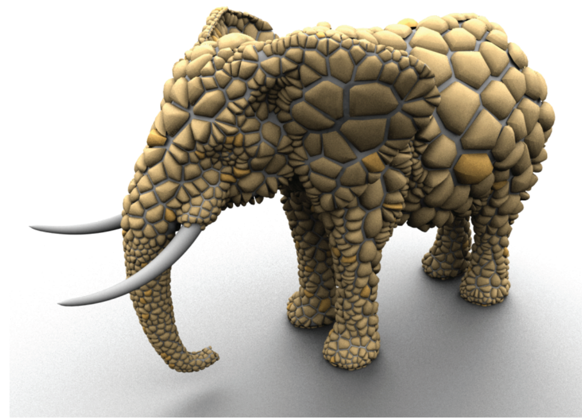

Figure 15: *An elephant decorated with mosaic tiles, with tile locations determined by ODT.*

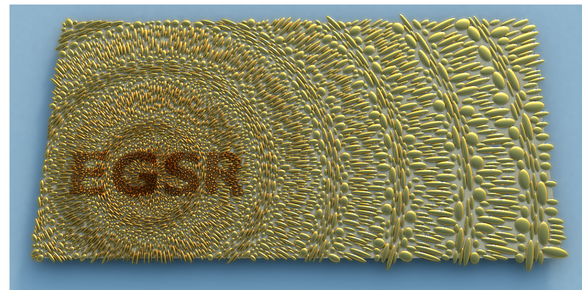

Figure 16: *A mosaic of ellipsoid tiles placed with our ellipsoid dart throwing algorithm.*

a rendering with nearly a quarter million hairs placed using our algorithm. As shown in the close-ups, our algorithm produces a smoother hair distribution with fewer gaps than either random or stratified placement.

Mosaic tile placement. The placement of mosaic tiles is another application of optimized dart throwing. Figure 15 shows an elephant figurine decorated with a mosaic of tiles. In this case, each point becomes a mosaic tile in the model and the Voronoi diagram of the point set acts as a template for the tile shapes. Figure 16 shows a different mosaic built with non-overlapping ellipsoids. The subtle coloration effects in this example result from the different sizes, aspect ratios and amounts of shadowing on the ellipsoids.

9. Conclusion

This paper presented optimized dart throwing (ODT) algorithms to generate Poisson disk point sets on surfaces. We demonstrated the ability to directly handle several surface types, including triangle meshes, spheres, Bezier patches, subdivision surfaces and implicits. We extended the algorithm to handle non-uniform densities, ellipsoid placement, and point sets spaced according to geodesic distance. Finally we demonstrated ODT in the context of remeshing, hair modeling, and mosaic tile placement.

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.

Limitations and Future Work. Optimized dart throwing is quite fast in the uniform case, but it can run much more slowly when producing non-uniform distributions because of the acceleration grid. One area for future work would be to find a more efficient acceleration structure for finding neighborhoods of points with different sizes. A solution to this issue would be useful in other problem settings, such as collision detection and scattered data interpolation.

Ideas for extending the modeling capabilities of ODT include placing arbitrary objects without overlap, adding constraints to the dart placement such as lines that the points may not cross, and coherently animating the point sets.

Acknowledgements. We would like to thank the makers of the *tight cocone* [DG03] remeshing application, which we used for figure 13, as well as Robert Nelson for creating the tiled elephant in Figure 15. This work was supported by the NSF and Austrian FWF Grant P20768-N13.

References

[AECdVDI03] ALLIEZ P., ÉRIC COLIN DE VERDIÈRE, DEVILLERS O., ISENBURG M.: Isotropic surface remeshing. In *Shape Modeling Intl. 2003* (2003), p. 49. 2

[Bri07] BRIDSON R.: Fast Poisson disk sampling in arbitrary dimensions. *ACM SIGGRAPH 2007 sketches* (2007). 2

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1 (1986), 51–72. 1

[CSHD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. *ACM Trans. Graph. (SIGGRAPH 2003) 22*, 3 (2003), 287–294. 1

[DG03] DEY T. K., GOSWAMI S.: Tight cocone: a water-tight surface reconstructor. In *ACM symposium on Solid modeling and applications* (2003), pp. 127–134. 10

[DH06] DUNBAR D., HUMPHREYS G.: A spatial data structure for fast poisson-disk sample generation. *ACM Trans. Graph. (SIGGRAPH 2006) 25*, 3 (2006), 503–508. 2

[FHHJ08] FENG L., HOTZ I., HAMANN B., JOY K.: Anisotropic noise samples. *IEEE Trans. on Visualization and Computer Graphics 14*, 2 (2008), 342–354. 2

[FZ08] FU Y., ZHOU B.: Direct sampling on surfaces for high quality remeshing. In *ACM symposium on solid and physical modeling* (2008), pp. 115–124. 2, 8, 9

[Jon06] JONES T. R.: Efficient generation of poisson-disk sampling patterns. *J. of Graphics Tools 11*, 2 (2006), 27–36. 2

[KCODL06] KOPF J., COHEN-OR D., DEUSSEN O., LISCHINSKI D.: Recursive wang tiles for real-time blue noise. *ACM Trans. Graph. (SIGGRAPH 2006) 25*, 3 (2006), 509–518. 2

[KS98] KIMMEL R., SETHIAN J.: Computing geodesic paths on manifolds. *Proceedings of National Academy of Sciences, USA, 95(15): 8431-8435.* (1998). 2

[LLLF08] LI H., LO K.-Y., LEUNG M.-K., FU C.-W.: Dual poisson-disk tiling: An efficient method for distributing features on arbitrary surfaces. *IEEE Trans. on Visualization and Computer Graphics 14*, 5 (2008), 982–998. 2, 8

[Llo87] LLOYD S. P.: Least squares quantization in pcm. *IEEE Trans. on Information Theory*, 2 (1987), 129–137. 1

[LVJ05] LEE C. H., VARSHNEY A., JACOBS D. W.: Mesh saliency. In *ACM Trans. Graph. (SIGGRAPH 2005)* (2005), vol. 24, pp. 659–666. 9

[Mit91] MITCHELL D. P.: Spectrally optimal sampling for distribution ray tracing. *Computer Graphics (Proceedings of SIGGRAPH '91) 25*, 4 (1991), 157–164. 1

[MMP87] MITCHELL J. S. B., MOUNT D. M., PAPADIMITRIOU C. H.: The discrete geodesic problem. *SIAM J. Comput. 16*, 4 (1987), 647–668. 2

[NS04] NEHAB D., SHILANE P.: Stratified point sampling of 3D models. In *Eurographics Symposium on Point-Based Graphics* (June 2004), pp. 49–56. 2, 3

[ODJ04] OSTROMOUKHOV V., DONOHUE C., JODOIN P.-M.: Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph. 23*, 3 (2004), 488–495. 2

[Ost07] OSTROMOUKHOV V.: Sampling with polyominoes. *ACM Trans. Graph. (SIGGRAPH 2007) 26*, 3 (2007), 78–83. 2

[PC06] PEYRÉ G., COHEN L.: Geodesic remeshing using front propagation. *Int. J. Comput. Vision 69*, 1 (2006), 145–156. 2

[RWCS05] ROVIRA J., WONKA P., CASTRO F., SBERT M.: Point sampling with uniformly distributed lines. In *Eurographics Symposium on Point-Based Graphics* (2005), p. 109. 2

[SAG03] SURAZHSKY V., ALLIEZ P., GOTSMAN C.: Isotropic remeshing of surfaces: a local parameterization approach. In *Meshing Roundtable* (2003), pp. 215–224. 2

[Set96] SETHIAN J. A.: A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci* (1996), pp. 1591–1595. 2, 7

[SSK*05] SURAZHSKY V., SURAZHSKY T., KIRSANOV D., GORTLER S. J., HOPPE H.: Fast exact and approximate geodesics on meshes. In *ACM Trans. Graph. (SIGGRAPH 2005)* (2005), pp. 553–560. 2

[SV01] SETHIAN J., VLADIMIRSKY A.: Ordered Upwind Methods for Static Hamiltonian-Jacobi Equations. *Proceedings of the National Academy of Sciences of the United States of America 98*, 20 (2001), 11069–11074. 2

[Tur92] TURK G.: Re-tiling polygonal surfaces. *Computer Graphics (Proceedings of SIGGRAPH '92) 26*, 2 (1992), 55–64. 1, 2, 3, 9

[WCC*04] WANG W., CHOI Y.-K., CHAN B., KIM M.-S., WANG J.: Efficient collision detection for moving ellipsoids using separating planes. *Computing 72*, 1-2 (2004), 235–246. 5

[WCE07] WHITE K. B., CLINE D., EGBERT P. K.: Poisson disk point sets by hierarchical dart throwing. In *IEEE / EG Symposium on Interactive Ray Tracing 2007* (2007), pp. 129–132. 2, 3, 5

[WDB*08] WEBER O., DEVIR Y. S., BRONSTEIN A. M., BRONSTEIN M. M., KIMMEL R.: Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph. 27*, 4 (2008), 1–16. 2, 7

[Wei08] WEI L.-Y.: Parallel poisson disk sampling. In *ACM Trans. Graph. (SIGGRAPH 2008)* (2008), pp. 1–9. 2

[WH05] WITKIN A., HECKBERT P.: Using particles to sample and control implicit surfaces. In *Int. Conf. on Computer Graphics and Interactive Techniques* (2005), pp. 269–277. 2

[WP04] WU X., PETERS J.: Interference detection for subdivision surfaces. *Computer Graphics Forum: Proceedings of Eurographics 2004 23*, 3 (2004), 577–584. 5

[WWK01] WANG W., WANG J., KIM M.-S.: An algebraic condition for the separation of two ellipsoids. *Comput. Aided Geom. Des. 18*, 6 (2001), 531–539. 5, 6

© 2009 The Author(s)
Journal compilation © 2009 The Eurographics Association and Blackwell Publishing Ltd.