Streaming Geometric Optimization using Graphics Hardware *

Pankaj K. Agarwal[1], Shankar Krishnan[2], Nabil H. Mustafa[1], and Suresh Venkatasubramanian[2]

[1] Dept. of Computer Science, Duke University, Durham, NC 27708-0129, U.S.A.
{pankaj, nabil}@cs.duke.edu
[2] AT&T Labs - Research, 180 Park Ave, Florham Park, NJ 07932.
{suresh, krishnas}@research.att.com

Abstract. In this paper we propose algorithms for solving a variety of geometric optimization problems on a stream of points in $\mathbb{R}^2$ or $\mathbb{R}^3$. These problems include various extent measures (e.g. diameter, width, smallest enclosing disk), collision detection (penetration depth and distance between polytopes), and shape fitting (minimum width annulus, circle/line fitting). The main contribution of this paper is a *unified* approach to solving all of the above problems efficiently using modern graphics hardware. All the above problems can be approximated using a constant number of passes over the data stream. Our algorithms are easily implemented, and our empirical study demonstrates that the running times of our programs are comparable to the best implementations for the above problems. Another significant property of our results is that although the best known implementations for the above problems are quite different from each other, our algorithms all draw upon the same set of tools, making their implementation significantly easier.

1 Introduction

The study of *streaming data* is motivated by numerous applications that arise in the context of dealing with massive data sets. In this paper we propose algorithms for solving a variety of geometric optimization problems over a stream of two or three dimensional geometric data (e.g. points, lines, polygons). In particular, we study three classes of problems: (a) *Extent measures:* computing various *extent measures* (e.g. diameter, width, smallest enclosing circle) of a stream of points in $\mathbb{R}^2$ or $\mathbb{R}^3$, (b) *Collision detection:* computing the penetration depth of a pair of convex polyhedra in three dimensions and (c) *Shape fitting:* approximating a set of points by simple shapes like circles or annuli.

Many of the problems we study can be formulated as computing and/or overlaying lower and upper envelopes of certain functions. We will be considering approximate solutions, and thus it suffices to compute the value of these

* Pankaj Agarwal and Nabil Mustafa are supported by NSF grants ITR–333–1050, EIA–9870724, EIA–997287, and CCR–02–04118, and by a grant from the U.S.-Israeli Binational Science Foundation.

envelopes at a set of uniformly sampled points, i.e., on a grid. This allows us to exploit recent developments in graphics hardware accelerators. Almost all modern graphics cards (examples include the nVidia GeForce and ATI Radeon series) provide hardware support for computing the envelope of a stream of bivariate functions at a uniform sample of points in $[-1, +1]^2$ and for performing various arithmetic and logical operations on each of these computed values, which makes them ideal for our applications. We therefore study the above streaming problems in the context of graphics hardware.

Related work. In the standard streaming modelthe input $\{x_1, \ldots, x_n\}$ is written in sequence on an *input tape*. The algorithm has a *read head*, and in each pass, the read head makes one sequential scan over the input tape. The efficiency of an algorithm is measured in terms of the size of the working space, the number of passes, and the time it spends on performing the computation. There are numerous algorithms for computing properties of data streams, as well as various lower bounds on the resources required [20]. Data stream computations of geometric properties like the diameter, convex hull, and minimum spanning tree have also received recent attention [15, 11, 13, 6].

Traditionally, graphics hardware has been used for rendering three dimensional scenes. But the growing sophistication of graphics cards and their relatively low cost has led researchers to use their power for a variety of problems in other areas, and specially in the context of geometric computing [12, 19, 16]. Fournier and Fussel [7] were the first to study general stream computations on graphics cards; a recent paper [8] shows lower bounds on the number of passes required by hardware-based k^{th}-element selection operations, as well as showing the necessity of certain hardware functions in reducing the number of passes in selection from $\Omega(n)$ to $O(\log n)$.

There has been extensive work in computational geometry and computing extent measures and shape fitting [3]. The most relevant work in a recent result by Agarwal *et al.* [2] which presents an algorithm for computing a small size "core set" C of a given set S of points in $\mathbb{R}^d$ whose extent approximates the extent of S, yielding linear time approximations for computing the diameter and smallest spherical shell of a point set. Their algorithm can be adapted to the streaming model, in the sense that C can be computed by performing one pass over S, after which one can compute an ε-approximation of the desired extent measure in $1/\varepsilon^{O(1)}$ time using $1/\varepsilon^{O(1)}$ memory.

Our work. In this paper, we demonstrate a large class of geometric optimization problems that can be approximated efficiently using graphics hardware. A unifying theme of the problems that we solve is that they can be expressed in terms of minimizations over envelopes of bivariate functions.

Extent problems: We present hardware-based algorithms for computing the diameter and width (in two and three dimensions) and the smallest enclosing ball (in two dimensions) of a set of points. All the algorithms are approximate, and compute the desired answer in a constant number of passes. We note here that although the number of passes is more than one, each pass does not use any information from prior passes and the computation effectively runs in a single pass. For reasons that will be made clear in Section 4, the graphics pipeline requires us to perform a series of passes that explore different regions of the search space.

In addition, the smallest bounding box of a planar point set can also be approximated in a constant number of passes; computing the smallest bounding box in three dimensions can be done in $1/\sqrt{\alpha-1}$ passes, where α is an approximation parameter.

Collision detection: We present a hardware-based algorithm for approximating the *penetration depth* between two convex polytopes. In general, our method can compute any inner product-based distance between any two convex polyhedra (intersecting or not). Our approach can also be used to compute the Minkowski sum of two convex polygons in two dimensions.

Shape fitting and other problems: We also present heuristics for a variety of shape-fitting problems in the plane: computing the minimum width annulus, best-fit circle, and best-fit line for a set of points, and computing the Hausdorff distance between two sets of points. Our methods are also applicable to many problems in *layered manufacturing* [17].

Experimental results: An important practical consequence of our unified approach to solving these problems is that all our implementations make use of the same underlying procedures, and thus a single implementation provides much of the code for all of the problems we consider. We present an empirical study that compares our algorithms to existing implementations for representative problems from the above list; in all cases we are comparable, and in many cases we are far superior to existing software-based implementations.

2 Preliminaries

The graphics pipeline. The graphics pipeline is primarily used as a rendering (or "drawing") engine to facilitate interactive display of complex three-dimensional geometry. The input to the pipeline is a set of geometric primitives and images, which are *transformed* and *rasterized* at various stages of the pipeline to produce an array of *fragments*, that is "drawn" on a two-dimensional grid of pixels known as the *frame buffer*. The frame buffer is a collection of several individual dedicated buffers (color, stencil, depth buffers etc.). The user interacts with the pipeline via a standardized software interface (such as OpenGL or DirectX) that is designed to mimic the graphics subsystem. For more details, the reader may refer to the OpenGL programming guide [21].

Computing Envelopes. Let $F = \{f_1, \ldots, f_n\}$ be a set of d-variate functions. The *lower envelope* of F is defined as $E_F^-(x) = \min_i f_i(x)$, and the *upper envelope* of F is defined as $E_F^+(x) = \max_i f_i(x)$. The projection of E_F^- (resp. E_F^+) is called the *minimization* (resp. *maximization*) diagram of S. Set $f_F^-(x)$ (resp. $f_F^+(x)$) to be the index of a function of F that appears on its lower (resp. upper) envelope. Finally, define $I_F(x) = E_F^+(x) - E_F^-(x)$. We will omit the subscript F when it is obvious from the context. If F is a family of piecewise-linear bivariate functions, we can compute E^-, E^+, f^-, f^+ for each pixel $x \in [-1, +1]^2$, using the graphics hardware. We will assume that function $f_i(x)$ can be described accurately as a collection of triangles.

Computing E^- (E^+): Each vertex v_{ij} is assigned a color equal to its z-coordinate (depth) (or function value). The graphics hardware generates color values across the face of a triangle by performing bilinear interpolation of the colors at the vertices. Therefore, the color value at each pixel correctly encodes the function value. We disable the stencil test, set the depth test to min (resp. max). After rendering all the functions, the color values in the framebuffer contains their lower (resp. upper) envelope. In the light of recent developments in programming the graphics pipeline, nonlinear functions can be encoded as part of a shading language (or fragment program) to compute their envelopes as well.

Computing f^- (f^+): Each vertex v_{ij} of function f_i is assigned the color c_i (in most cases, c_i is determined by the problem). By setting the graphics state similar to the previous case, we can compute f^- and f^+.

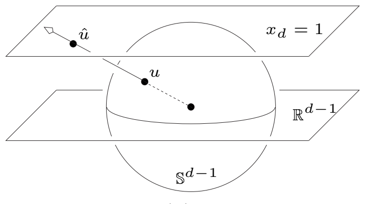
In many of the problems we address, we will compute envelopes of distance functions. That is, given a distance function $\delta(\cdot, \cdot)$ and a set $S = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^2$, we define $F = \{f_i(x) \equiv \delta(x, p_i) \mid 1 \le i \le n\}$, and we wish to compute the lower and upper envelopes of F. For the Euclidean metric, the graph of each f_i is a cone whose axis is parallel to the z-axis and whose sides are at an angle of $\pi/4$ to the xy-plane. For the square Euclidean metric, it is a paraboloid symmetric around a vertical line. Such surfaces can be approximated to any desired degree of approximation by triangulations ([12]).

Approximations. For purposes of computation, the two-dimensional plane is divided into pixels. This discretization of the plane makes our algorithms approximate by necessity. Thus, for a given problem, the cost of a solution is a function both of the algorithm and the screen resolution. We define a (α, g)-approximation algorithm to be one that provides a solution of cost at most α times the optimal solution, with a grid cell size of $g = g(I, \alpha)$, where I is the instance of the problem. This definition implies that different instances of the same problem may require different grid resolutions.

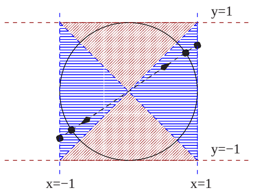
3 Gauss Maps And Duality

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$. A direction in $\mathbb{R}^d$ can be represented by a unit vector $u \in \mathbb{S}^{d-1}$. For $u \in \mathbb{S}^{d-1}$, let $\hat{u}$ be its *central* projection, i.e., the intersection point of the ray $\overrightarrow{ou}$ with the hyperplane $x_d = 1$ (resp. $x_d = -1$) if u lies in the positive (resp. negative) hemisphere.

For a direction u, we define the *extremal point* in direction u to be $\lambda(u, S) = \arg\max_{p \in S} \langle \hat{u}, p \rangle$, where $\langle \cdot, \cdot \rangle$ is the inner product. The *directional width* of S is $\omega(u, S) = \max_{p \in S} \langle \hat{u}, p \rangle - \min_{p \in S} \langle \hat{u}, p \rangle$. The *Gaussian map* of the convex hull of S is the decomposition of $\mathbb{S}^{d-1}$ into maximal connected regions so that the the extremal point is the same for all directions within one region.



(a)



(b)

Fig. 1. (a) An illustration of central projection. (b) Two duals used to capture the Gaussian Map.

For a point $p = (p_1, \ldots, p_d)$, we define its *dual* to be the hyperplane $p^* : x_d = p_1x_1 + \cdots + p_{d-1}x_{d-1} + p_d$. Let $H = \{p^* \mid p \in S\}$ be the set of hyperplanes dual to the points in S. The following is easy to prove.

Lemma 1. *For $u \in \mathbb{S}^{d-1}$, $\lambda(u, S) = f_H^+(\hat{u}_1, \ldots, \hat{u}_{d-1})$ if u lies in the positive hemisphere, and $\lambda(u, S) = f_H^-(\hat{u}_1, \ldots, \hat{u}_{d-1})$ if u lies in the negative hemisphere; here $\hat{u} = (\hat{u}_1, \ldots, \hat{u}_d)$.* □

Hence, we can compute $\lambda(u, S)$ using f_H^+ and f_H^-. Note that the central projection of the portion of the Gaussian map of S in the upper (resp. lower) hemisphere is the maximization (resp. minimization) diagram of H. Thus, for $d = 3$ we can compute portion of the Gaussian map of S whose central projection lies in the square $[-1, +1]^2$, using graphics hardware, as described in Section 2. In other words, we can compute the extremal points of S for all u such that $\hat{u} \in [-1, 1]^2 \times \{1, -1\}$. If we also take the central projection of a vector $u \in \mathbb{S}^2$ onto the planes $y = 1$ and $x = 1$, then at least one of the central projections of u lies in the square $[-1, +1]^2$ of the corresponding plane. Let R_x (resp. R_y) be the rotation transform that maps the unit vector $(1, 0, 0)$ (resp. $(0, 1, 0)$) to $(0, 0, 1)$. Let H_x (resp. H_y) be the set of planes dual to the point set $R_x(S)$ (resp. $R_y(S)$). If we compute $f_{H_x}^+, f_{H_x}^-, f_{H_y}^+$, and $f_{H_y}^-$ for all $x \in [-1, +1]^2$, then we can guarantee that we have computed extremal points in all directions (see Fig. 1(b) for an example in two dimensions).

In general, vertices of the arrangement of dual hyperplanes may not lie in the box $[-1, +1]^3$. A generalization of the above idea can be used to compute a family of three duals such that any vertex of the dual arrangement is guaranteed to lie in the region $[-1, +1]^2 \times [-n, n]$ in *some* dual. Such a family of duals can be used to compute more general functions on arrangements using graphics hardware; a special case of this result in two dimensions was proved in [16]. In general, the idea of using a family of duals to maintain boundedness of the arrangement can be extended to d dimensions. We defer these more general results to a full version of the paper.

4 Extent Measures

Let $S = \{p_1, \ldots, p_n\}$ be a set of points in $\mathbb{R}^d$. We describe streaming algorithms for computing the diameter and width of S for $d \leq 3$ and the smallest enclosing box and ball of S for $d = 2$.

Diameter. In this section we describe a six-pass algorithm for computing the diameter of a set S (the maximum distance between any two points of S) of n points in $\mathbb{R}^3$. It is well known that the diameter of S is realized by a pair of antipodal points, i.e., there exists a direction u in the positive hemisphere of $\mathbb{S}^2$ such that $\text{diam}(S) = \|\lambda(u,S) - \lambda(-u,S)\| = \|f_H^+(\hat{u}_1,\hat{u}_2) - f_H^-(\hat{u}_1,\hat{u}_2)\|$, where H is the set of planes dual to the points in S. In order to compute $\|f_H^+(x) - f_H^-(x)\|$, we assign the RGB values of the color of a plane p_i^* to be the coordinates of p_i. The first pass computes f_H^+, so after the pass, the pixel x in the color buffer contains the coordinates of $f_H^+(x)$. We copy this buffer to the texture memory and compute f_H^- in the second pass. We then compute $\|f_H^+(x) - f_H^-(x)\|$ for each pixel. Since the hardware computes these values for $x \in [-1,+1]^2$, we repeat these steps for $R_x(S)$ and $R_y(S)$ as well. Since our algorithm operates in the dual plane, the discretization incurred is in terms of the directions, yielding the following result.

Theorem 1. *Given a point set $S \subset \mathbb{R}^3, \alpha > 1$, there is a six-pass $(\alpha, g(\alpha))$-approximation algorithm for computing the diameter of S, where $g(\alpha) = O(1/\sqrt{\alpha})$.*

Width. Let S be a set of n points in $\mathbb{R}^3$. The *width* of S is the minimum distance between two parallel planes that enclose P between them, i.e., $\text{width}(S) = \min_{u \in \mathbb{S}^2} \omega(u,S)$. The proof of the following lemma is relatively straightforward.

Lemma 2. *Let R_x, R_y be the rotation transforms as described earlier, and let H (resp. H_x, H_y) be the set of planes dual to the points in S (resp. $R_x(S)$, $R_y(S)$). Then*

$$\text{width}(S) = \min_{p \in [-1,+1]^2} \frac{1}{\|(p,1)\|} \min\{I_H(p), I_{H_x}(p), I_{H_y}(p)\}.$$

This lemma implies that the algorithm for width can be implemented similar to the algorithm for diameter. Consider a set of coplanar points in $\mathbb{R}^3$. No discretized set of directions can yield a good approximation to the width of this set (which is zero). Hence, we can only prove a slightly weaker approximation result, based on knowing a lower bound on the optimal width w^*. We omit the details from this version and conclude the following.

Theorem 2. *Given a point set $S \subset \mathbb{R}^3$, $\alpha > 1$, and $\tilde{w} \leq w^*$, there is a six-pass $(\alpha, g(\alpha, \tilde{w}))$-approximation algorithm for computing the width of S.*

1-center. The 1-center of a point set S in $\mathbb{R}^2$ is a point $c \in \mathbb{R}^2$ minimizing $\max_{p \in P} d(c,p)$. This is an envelope computation, but in the primal plane. For each point $p \in S$, we render the colored distance cone as described in Section 2. The 1-center is then the point in the upper envelope of the distance cones with the smallest distance value. The center of the smallest enclosing ball will always lie inside conv(S). The radius of the smallest enclosing ball is at least half the diameter Δ of S. Thus, if we compute the farthest point Voronoi diagram on a grid of cell size $g = \alpha\Delta/2$, the value we obtain is a α-approximation to the radius of the smallest enclosing ball. An approximate diameter computation gives us $\tilde{\Delta} \leq 2\Delta$, and thus a grid size of $\alpha\tilde{\Delta}/4$ will obtain the desired result.

Theorem 3. *Given a point set S in $\mathbb{R}^2$ and a parameter $\alpha > 1$, there is a two-pass $(\alpha, g(\alpha))$-approximation algorithm for computing the smallest-area disk enclosing S.*

Smallest bounding box. Let S be a set of points in $\mathbb{R}^2$. A rectangle enclosing S consists of two pairs of parallel lines, lines in each pair orthogonal to the other. For a direction $u \in \mathbb{S}^1$, let $u^\perp$ be the direction normal to u. Then the side lengths of the smallest rectangle whose edges are in directions u and $u^\perp$ that contains S are $W(u) = \omega(u, S)$ and $H(u) = \omega(u^\perp, S)$. Hence, the area of the smallest rectangle containing S is $\min_{u \in \mathbb{S}^1} W(u) \cdot H(u)$. The algorithm to compute the minimum-area two-dimensional bounding box can now be viewed as computing the minimum widths in two orthogonal directions and taking their product. Similarly, we can compute a minimum-perimeter rectangle containing S. Since the algorithm is very similar to computing the width, we omit the details and conclude the following.

Theorem 4. *Given a point set S in $\mathbb{R}^2$, $\alpha > 1$, and a lower bound $\tilde{a}$ on the area of the smallest bounding box, there is a four-pass $(\alpha, g(\alpha, a))$-approximation algorithm for computing the smallest enclosing bounding box.*

It is not clear how to extend this algorithm to $\mathbb{R}^3$ using a constant number of passes since the set of directions normal to a given direction is $\mathbb{S}^1$. However, by sampling the possible choices of orthogonal directions, we can get a $(1 + \alpha)$-approximation in $1/\sqrt{\alpha - 1}$ passes. Omitting all the details, we obtain the following.

Theorem 5. *Given point set $S \subset \mathbb{R}^3$, $\alpha > 1$ and lower bound $\tilde{a}$ on the area of the smallest bounding box, there is an $O(1/\sqrt{\alpha - 1})$-pass $(\alpha, g(\alpha, a))$-approximation algorithm for computing the smallest bounding box.*

5 Collision Detection

Given two convex polytopes P and Q in $\mathbb{R}^3$, their penetration depth, denoted $PD(P, Q)$ is defined as the length of the shortest translation vector t such that P and $Q + t$ are disjoint. We can specify a placement of Q by fixing a reference point $q \in Q$ and specifying its coordinates. Assume that initially q is at the origin o. Since $M = P \oplus -Q$ is the set of placements of Q at which Q intersects P, $PD(P, Q) = \min_{z \in \partial M} d(o, z)$ For a direction $u \in \mathbb{S}^2$, let $h_M(u)$ be the tangent plane of M normal to direction u. As shown in [1], $PD(P, Q) = \min_{u \in \mathbb{S}^2} d(o, h_M(u))$

Let A be a convex polytope in $\mathbb{R}^3$ and let V be the set of vertices in A. For a direction $u \in \mathbb{S}^2$, let $g_A(u) = \max_{p \in V} \langle p, \hat{u} \rangle$. It can be verified that the tangent plane of A in direction u is $h_A(u) : \langle \hat{u}, x \rangle = g_A(u)$. Therefore $PD(P, Q) = \min_{u \in \mathbb{S}^2} \frac{g_M(u)}{\|\hat{u}\|}$. The following lemma shows how to compute $h_M(u)$ from $h_P(u)$ and $h_{-Q}(u)$.

Lemma 3. *For any* $u \in \mathbb{S}^2$,$g_M(u) = g_P(u) + g_{-Q}(u)$ $\square$

This lemma follows from the fact that for convex P and Q, the point of M extreme in direction u is the sum of the points of P and Q extreme in direction u. Therefore, $PD(P, Q) = \min_{u \in \mathbb{S}^2} g_P(u) + g_{-Q}(u)/\|u\|$.

Hence, we discretize the set of directions in $\mathbb{S}^2$, compute $g_P(u), g_{-Q}(u)$, $(g_P(u) + g_{-Q}(u))/\|\hat{u}\|$ and compute their minimum. Since g_P and g_{-Q} are upper envelopes of a set of linear functions, they can be computed at a set of directions by graphics hardware in six passes, as described in Section 4. We note here that the above approach can be generalized to compute any inner product-based distance between two non-intersecting convex polytopes in three dimensions. It can also be used to compute the Minkowski sum of polygons in two dimensions.

6 Shape Fitting

We now present hardware-based heuristics for shape analysis problems. These problems are solved in the primal, by computing envelopes of distance functions.

Circle fitting. The *minimum-width annulus* of a point set $P \subset \mathbb{R}^2$ is a pair of concentric disks R_1, R_2 of radii $r_1 > r_2$ such that P lies in the region $R_1 \setminus R_2$ and $r_1 - r_2$ is minimized. Note that the center of the minimum-width annulus could be arbitrarily far away from the point set (for example, the degenerate case of points on a line). Furthermore, when the minimum-width annulus is *thin*, the pixelization induces large errors which cannot be bounded. Therefore, we look at the special case when the annulus is not thin, i.e. $r_1 \geq (1 + \varepsilon) r_2$. For this case, Chan [4] presents a $(1+\varepsilon)$ approximation algorithm by laying a grid on the pointset, snapping the points to the grid points, and finding the annulus with one of the grid points as the center. This algorithm can be implemented efficiently in hardware as follows: for each point p_i, draw its Euclidean distance cone C_i as described in Section 2. Let $C = \{C_1, C_2, \ldots, C_n\}$ be the collection of distance functions. Then the minimum-width annulus can be computed as $\min_{x \in B} I_C(x)$ with center $\arg\min_{x \in B} I_C(x)$. This approach yields a fast streaming $(1 + \varepsilon)$-approximation algorithm for the minimum-width annulus (and for the minimum-area annulus as well, by using paraboloids instead of cones).

The *best-fit circle* of a set of points $P = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ is a circle $C(c, r)$ of radius r centered at c such that the expression $\sum_{p \in P} d^2(p, C)$ is minimized. For a fixed center c, elementary calculus arguments show that the optimal r is given by $r^* = 1/n \sum_{p \in P} d(p, c)$. Let $d_i = \|p_i - c\|$. The cost of the best fit circle of radius r^* centered at c can be shown to be $\sum_{i \leq n} d_i^2 - (1/n)(\sum_{i \leq n} d_i)^2$.

Once again, this function can be represented as an overlay of distance cones, and thus for each grid point, the cost of the optimal circle centered at this grid point can be computed. Unfortunately, this fails to yield an approximation guarantee for the same reasons as above.

Hausdorff distance. Given two point sets $P, Q \subset \mathbb{R}^2$, the *Hausdorff distance* d_H from P to Q is $\max_{p \in P} \min_{q \in Q} d(p, q)$. Once again, we draw distance cones for each point in Q, and compute the lower envelope of this arrangement of surfaces *restricted to points in* P. Now each grid point corresponding to a point of P has a value equal to the distance to the closest point in Q. A maximization over this set yields the desired result. For this problem, it is easy to see that as for the width, given any lower bound on the Hausdorff distance we can compute a $(\beta, g(\beta)$-approximation to the Hausdorff distance.

7 Experiments

In this section we describe some implementation specific details, and report empirical results of our algorithms, and compare their performance with software-based approximation algorithms.

Cost bottleneck. The costs of operations can be divided into two types: geometric operations, and fragment operations. Most current graphics cards have a number of *geometry engines* and *raster managers* to handle multiple vertex and fragment operations in parallel. Therefore, we can typically assume that the geometry transformation and each buffer operation takes constant time.

As the number of fragments increases, the rendering time is roughly unchanged till we saturate the rendering capacity (called the fill-rate), at which point performance degrades severely. We now propose a hierarchical method that circumvents the fill limitation by doing refined local searches for the solution.

Hierarchical refinement. One way to alleviate the fill-rate bottleneck is to produce fewer fragments per plane. Instead of sampling the search space with a uniform grid, we instead perform adaptive sampling by constructing a coarse grid, computing the solution value for each grid point and then recursively refining candidate points. The advantage of using adaptive refinement is that not all the grid cells need to be refined to a high resolution. However, the local search performed by this selective refinement could fail to find an approximate solution with the guarantee implied by this higher resolution. In our experiments, we will compare the results obtained from this approach with those obtained by software-based methods.

Empirical results. In this section we report on the performance of our algorithms. All our algorithms were implemented in C++ and OpenGL, and run on a 2.4GHz Pentium IV Linux PC with an ATI Radeon 9700 graphics card and 512 MB Memory. Our experiments were run on three types of inputs: (i) randomly generated convex shapes [9] (ii) large geometric models of various objects, available at http://www.cc.gatech.edu/graphmodels/ and (iii) randomly generated input using `rbox` (a component of `qhull`). In all our algorithms we use hierarchical refinement (with depth two) to achieve more accurate solutions.

Penetration depth. We compare our implementation of penetration depth (called `HwPD`) with our own implementation of an exact algorithm (called `SwPD`) based on Minkowski sums which exhibits quadratic complexity and with DEEP [14], which to the best of our knowledge is the only other implementation for penetration depth. We used the convex polytopes available at [9], as well as random polytopes found by computing the convex hull of points on random ellipsoids as inputs to test our code. The performance of the algorithms on the input set is presented in Table 1. `HwPD` always outperforms `SwPD` in running time, in some cases by over three orders of magnitude. With regard to DEEP, the situation is less clear. DEEP performs significant preprocessing on its input, so a single number is not representative of the running times for either program. Hence, we report both preprocessing times and query times (for our code, preprocessing time is merely reading the input). We note that DEEP crashed on some of our inputs; we mark those entries with an asterisk.

Polygon		`HwPD`			DEEP			`SwPD`	
Size	Size	Preproc.	Time	Depth	Preproc.	Time	Depth	Time	Depth
500	500	**0**	**0.04**	1.278747	**0.15**	**0**	1.29432	**27.69**	1.289027
750	750	**0**	**0.08**	1.053032	**0.25**	**0**	1.07359	**117.13**	1.071013
789	1001	**0.01**	**0.067**	1.349714	*	*	*	**148.87**	1.364840
789	5001	**0.01**	**0.17**	1.360394	*	*	*	-	-
5001	4000	**0.02**	**0.30**	1.362190	*	*	*	-	-
10000	5000	**0.04**	**0.55**	1.359534	**3.28**	**0**	1.4443	-	-

Table 1. Comparison of running times for penetration depth (in secs.). On the last three datasets, we stopped `SwPD` after it ran for over 25 minutes. Asterisks mark inputs for which DEEP crashed.

2D minimum width annulus. We compute an annulus by laying a $1/\varepsilon^2 \times 1/\varepsilon^2$ grid on the pointset, snapping the points to the grid, and then using the hardware to find the nearest/furthest neighbour of each grid point. The same algorithm can be implemented in software. We compare our implementation (called `HAnnWidth`) with the software implementation, called `SAnnWidth`. The input point sets to the programs were synthetically generated using `rbox`: R-Circle-r refers to a set of points with minimum width annulus r and is generated by sampling points from a circle and introducing small perturbations. See Table 2.

Error:	$\epsilon^2 = 0.002$	`HAnnWidth`		`SAnnWidth`	
Dataset	size	Time	Width	Time	Width
R-Circle-0.1	(1,000)	**0.36**	0.099882	**0.53**	0.099789
R-Circle-0.2	(1,000)	**0.35**	0.199764	**0.42**	0.199442
R-Circle-0.1	(2,000)	**0.66**	0.099882	**0.63**	0.099816
R-Circle-0.1	(5,000)	**1.58**	0.099882	**26.44**	0.099999
R-Circle-0.1	(10,000)	**3.12**	0.099882	**0.93**	0.099999

Table 2. Comparison of running time and approximation for 2D-min width annulus

3D width. We compare our implementation of width (called `HWidth`) with the code of Duncan *et al.* [5] (`DGRWidth`). Algorithm `DGRWidth` reduces the computation of the width to $O(1/\epsilon)$ linear programs. It then tries certain pruning heuristics to reduce the number of linear programs solved in practice. The performance of both the algorithms on a set of real graphical models is presented in Table 3: column four gives the $(1+\epsilon)$-approximate value of the width computed by the two algorithms for the ϵ given in the second column (this ϵ value dictates the window size required by our algorithm, as explained previously, and the number of linear programs solved by `DGRWidth`). `HWidth` always outperforms `DGRWidth` in running time, in some cases by more than a factor of five.

	Error	`HWidth`		`DGRWidth`	
Dataset size	ϵ	Time	Width	Time	Width
Club (16,864)	0.250	**0.45**	0.300694	**0.77**	0.312883
Bunny (35,947)	0.060	**0.95**	1.276196	**2.70**	1.29231
Phone (83,034)	0.125	**2.55**	0.686938	**6.17**	0.697306
Human (254,721)	0.180	**6.53**	0.375069	**18.91**	0.374423
Dragon (437,645)	0.075	**10.88**	0.813487	**39.34**	0.803875
Blade (882,954)	0.090	**23.45**	0.715578	**66.71**	0.726137

Table 3. Comparison of running time and approximation quality for 3D-width.

3D diameter. We compare our implementation (`HDiam`) with the approximation algorithm of Malandain and Boissonnat [18] (`MBDiam`), and Har-Peled [10] (`PDiam`). `PDiam` maintains a hierarchical decomposition of the point set, and iteratively throws away pairs that are not candidate for the diameter until an approximate distance is achieved by a pair of points. `MBDiam` is a further improvement on `PDiam`. Table 4 reports the timing and approximation comparisons for two error measures for graphical models. Although our running times in this case are worse than the software implementations, they are comparable even for very large inputs, illustrating the generality of our approach.

Error: $\epsilon = 0.015$	`HDiam`		`MBDiam`		`PDiam`	
Dataset size	Time	Diam	Time	Diam	Time	Diam
Club (16,864)	**0.023**	2.326992	**0.0**	2.32462	**0.00**	2.32462
Bunny (35,947)	**0.045**	2.549351	**0.75**	2.54772	**0.03**	2.54772
Phone (83,034)	**0.11**	2.416497	**0.01**	2.4115	**0.07**	2.4115
Human (254,721)	**0.32**	2.020594	**3.5**	2.01984	**0.04**	2.01938
Dragon (437,645)	**0.55**	2.063075	**17.27**	2.05843	**0.21**	2.05715
Blade (882,954)	**1.10**	2.246725	**0.1**	2.23939	**0.22**	2.22407

Table 4. Comparison of running time and approximation quality for 3D-diameter.

References

1. Agarwal, P., Guibas, L., Har-Peled, S., Rabinovitch, A., and Sharir, M. Penetration depth of two convex polytopes in 3d. *Nordic J. Comput. 7*, 3 (2000), 227–240.

2. Agarwal, P. K., Har-Peled, S., and Varadarajan, K. Approximating extent measures of points. Submitted for publication, 2002.
3. Agarwal, P. K., and Sharir, M. Efficient algorithms for geometric optimization. *ACM Comput. Surv. 30* (1998), 412–458.
4. Chan, T. M. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *Proc. 16th Annu. Sympos. on Comp. Geom.* (2000), pp. 300–309.
5. Duncan, C., Goodrich, M., and Ramos, E. Efficient approximation and optimization algorithms for computational metrology. In *ACM-SIAM Symp. Discrete Algo.* (1997), pp. 121–130.
6. Feigenbaum, J., Kannan, S., and Zhang, J. Computing diameter in the streaming and sliding-window models. DIMACS Working Group on Streaming Data Analysis II, 2003.
7. Fournier, A., and Fussell, D. On the power of the frame buffer. *ACM Transactions on Graphics* (1988), 103–128.
8. Guha, S., Krishnan, S., Munagala, K., and Venkatasubramanian, S. The power of a two-sided depth test and its application to CSG rendering and depth extraction. Tech. rep., AT&T, 2002.
9. Har-Peled, S. http://valis.cs.uiuc.edu/ sariel/research/papers/99/nav/nav.html.
10. Har-Peled, S. A practical approach for computing the diameter of a point-set. In *Proc. 17th Annu. Symp. on Comp. Geom.* (2001), pp. 177–186.
11. Hersberger, J., and Suri, S. Convex hulls and related problems in data streams. In *SIGMOD-DIMACS MPDS Workshop* (2003).
12. Hoff III, K. E., Keyser, J., Lin, M., Manocha, D., and Culver, T. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics 33*, Annual Conference Series (1999), 277–286.
13. Indyk, P. Stream-based geometric algorithms. In *SIGMOD-DIMACS MPDS Workshop* (2003).
14. Kim, Y. J., Lin, M. C., and Manocha, D. Fast penetration depth estimation between polyhedral models using hierarchical refinement. In *6th Intl. Workshop on Algo. Founda. of Robotics* (2002).
15. Korn, F., Muthukrishnan, S., and Srivastava, D. Reverse nearest neighbour aggregates over data streams. In *Proc. 28th Conf. VLDB* (2002).
16. Krishnan, S., Mustafa, N., and Venkatasubramanian, S. Hardware-assisted computation of depth contours. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms* (2002), pp. 558–567.
17. Majhi, J., Janardan, R., Smid, M., and Schwerdt, J. Multi-criteria geometric optimization problems in layered manufacturing. In *Proc. 14th Annu. Symp. on Comp. Geom.* (1998), pp. 19–28.
18. Malandain, G., and Boissonnat, J.-D. Computing the diameter of a point set. In *Discrete Geometry for Computer Imagery* (Bordeaux, France, 2002), A. Braquelaire, J.-O. Lachaud, and A. Vialard, Eds., vol. 2301 of *LNCS*, Springer.
19. Mustafa, N., Koutsofios, E., Krishnan, S., and Venkatasubramanian, S. Hardware assisted view dependent map simplification. In *Proc. 17th Annu. Symp. on Comp. Geom.* (2001), pp. 50–59.
20. Muthukrishnan, S. Data streams: Algorithms and applications. Tech. rep., Rutgers University, 2003.
21. Woo, M., Neider, J., Davis, T., and Shreiner, D. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2,* 3 ed. Addison-Wesley, 1999.