An Empirical Comparison of Techniques for Updating Delaunay Triangulations

Leonidas Guibas
Computer Science Department
Stanford University
Stanford, CA 94305
guibas@graphics.stanford.edu

Daniel Russel
Computer Science Department
Stanford University
Stanford, CA 94305
drussel@graphics.stanford.edu

ABSTRACT

The computation of Delaunay triangulations from static point sets has been extensively studied in computational geometry. When the points move with known trajectories, kinetic data structures can be used to maintain the triangulation. However, there has been little work so far on how to maintain the triangulation when the points move without explicit motion plans, as in the case of a physical simulation. In this paper we examine how to update Delaunay triangulations after small displacements of the defining points, as might be provided by a physics-based integrator. We have implemented a variety of update algorithms, many new, toward this purpose. We ran these algorithms on a corpus of data sets to provide running time comparisons and determined that updating Delaunay can be significantly faster than recomputing.

Categories and Subject Descriptors: F.2.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; I.6.m [Computing Methodologies]: Simulation and Modeling—*Miscellaneous*

General Terms: Algorithms, Experimentation

Keywords: Delaunay triangulation update motion

1. INTRODUCTION

Delaunay triangulations, and their duals Voronoi diagrams, are fundamental to computational geometry. They provide a decomposition of the space surrounding a set of points into well shaped cells which can be used to extract proximity information and detect collisions [30, 29]. However, they are still fairly expensive to compute, despite extensive investigations of how to build them quickly and robustly [10, 28, 33, 11]. Recently modifications to existing algorithms have been proposed which allow the Delaunay triangulation of millions of points to be computed [3], extending the domains in which such techniques can be applied.

In many situations where the Delaunay triangulation or Voronoi diagram is required, the Delaunay triangulation of a perturbed configuration of the points is available. For example, in a physical simulation where Delaunay is used for collision detection, the Delaunay triangulation must be recomputed at each time step as the coordinates are updated by the integrator. These updates are necessarily small, in order to ensure the accuracy of the simulation; thus the Delaunay triangulation of the original and perturbed point sets often have very similar combinatorial structure.

Our problem is then, given a point set with its Delaunay triangulation, and a perturbation of each point in the original set, compute the Delaunay triangulation of the perturbed point set. We present several update techniques and compare their running times on various simulation data sets.

A motivation for this work came from molecular simulations. There, the Delaunay triangulation is used to compute the molecular surface area and volume and their various derivatives [14, 15]. The combinatorial structure of the Delaunay triangulations before and after an integrator updates the atom coordinates generally differ by fewer than 10% of the tetrahedra of the triangulation. We found that by performing flips on the initial triangulation structure, in practice, we can update the triangulation of a molecule after a time step in approximately half to three quarters the time it takes to recompute. The details of this method are discussed in Section 4. A similar method involving a mix of point removals and flips to update a Delaunay triangulation where the points were constrained to stay within their Voronoi cells was independently explored in [20].

Kinetic data structures, first introduced in [7], can be used to maintain a Delaunay triangulation under smooth motion of the underlying points. In Section 3 we discuss the trade offs involved with and several techniques for using kinetic data structures to update a Delaunay triangulation. In addition, we discuss possible areas where improved understanding and optimization might make kinetic data structure based update methods competitive with rebuilding.

There has been some work exploring using a kinetic Delaunay triangulation to perform inter-frame collision detection and dynamically adjust the integrator step size in the context of a particle simulation [22]. In general, the issues and advantages associated with looking at the configuration between frames are quite application dependent. Therefore, we will restrict ourselves to finding the Delaunay of the perturbed conformation and ignore any intermediate state.

An alternative would be to allow the triangulation to deviate slightly from Delaunay in the hopes of achieving greater stability. In [6] the authors used *almost Delaunay simplices* for more robustly

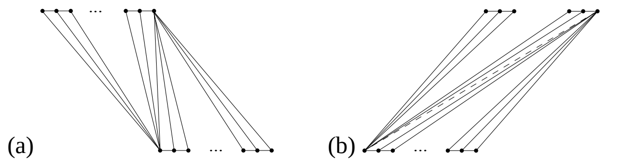


Figure 1: ***The number of certificate failures can be misleading:*** **(a) shows the Delaunay triangulation of an arrangement of points. They are then moved to configuration (b) without changing the triangulation. The resulting triangulation has only edge with a failed certificate, which is dashed, even though none of the triangles are Delaunay. $\Omega(n^2)$ flips are necessary to make triangulation (b) Delaunay.**

computing adjacency. However, there has been no investigation of how to maintain such a set under motion, or maintain a subset of the almost Delaunay simplices which forms a triangulation.

2. DELAUNAY PROPERTIES

The global correctness of a Delaunay triangulation can be verified using a set of local certificates, namely that for each facet (edge in 2D), the sphere (circle) defined by the four (three) points of one cell (triangle) incident to the facet (edge) does not contain the remaining point from the other incident cell (triangle). This test is known as the InCircle predicate. As a result, it is easy to test if a given triangulation is the Delaunay triangulation of a set of points. Unfortunately, even a single failed certificate can mean that the triangulation is arbitrarily far from Delaunay. For an example, see Figure 1. In simulations, when points do not move far, such situations do not occur and the number of failed certificates is a good measure of the amount of work that needs to be done to update a triangulation to Delaunay.

In 2D, a triangulation which is not Delaunay can be made Delaunay by repeatedly finding an edge with an invalid certificate, and replacing it with the other diagonal of the quadrilateral defined by the two triangles sharing the edge. Such flips are called *Delaunay flips*. Converting an arbitrary triangulation to Delaunay using this technique requires $O(n^2)$ flips. The concept of Delaunay flip can be extend to 3D, where a facet is flipped to an edge and vice versa. However, it is not always possible to convert a triangulation to Delaunay using Delaunay flips alone [31]. However, we have found that flipping works most of the time in 3D, as is discussed in Section 5.

If the triangulation is not embedded, flipping becomes problematic even in 2D. Figure 2 shows such an example where Delaunay flips cannot be used to convert a non-embedded triangulation to Delaunay. As a result, in order to use Delaunay flips we will have to ensure that the triangulation is embedded first.

In 3D there are relatively few local operations that can be performed on a Delaunay triangulation, the most important being point insertion and point removal. A point can be inserted by removing all cells whose circumsphere contains the point (i.e. those whose certificate would be invalidated by the new point). The resulting hole is star-shaped around the new vertex and can be filled by connecting each facet on the boundary to the new point with a cell. Point insertion was first proposed using flips [17] and forms the basis for most recent implementations of Delaunay triangulation construction. The cells which are removed by the point insertion can be kept around and used to build a hierarchical point location structure, which speeds insertions.

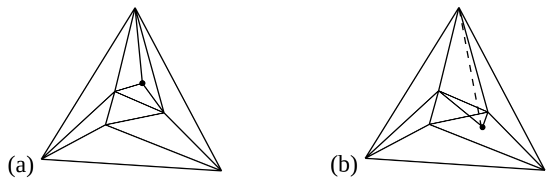


Figure 2: ***Flips do not work when the triangulation is not embedded:*** **(a) shows the Delaunay triangulation of an arrangement of points. The point marked with a dot is then moved as shown in (b) without changing the triangulation, making it no longer embedded. The only edge that is not locally Delaunay is the dashed edge, and this edge cannot be flipped (since the edge that would be created by the flip already exists).**

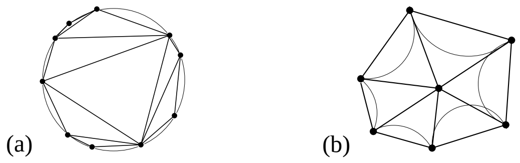


Figure 3: ***Stability and Delaunay triangulations:*****(a) shows an arrangement of points with a very unstable Delaunay triangulation due to the degenerate conformation. In most cases there is some tolerance around each vertex, as shown in (b).**

A point can be also be deleted from a Delaunay triangulation. This operation is slightly more complicated and is covered in [13]. We will use both these operations for updating Delaunay triangulations.

Delaunay triangulations can be very brittle structures. In certain configurations, for example the nearly co-circular points in Figure 3 (a), arbitrarily small displacements can result in large changes to the triangulation. While the example shown is highly degenerate and is unlikely to arise in practice, in much of the data we looked at the distance the vertices must be moved in order to invalidate a certificate is quite small. The *tolerance* of a certificate is the minimal amount each vertex in the certificate must move in order to invalidate the certificate and can be easily computed [1]. We found the average tolerance to be around 10% of the local edge length and that perturbations of 1% of the edge length could invalidate 20% of the certificates.

2.1 A Note on Terminology

In the remainder of the paper we will restrict ourselves to 3D Delaunay triangulations, although much of what is said also applies to 2D. We will use *cell* to mean the full dimensional simplex (a tetrahedron) and *facet* to mean the simplex with a dimensionality of $d - 1$ (a triangle).

The *lifting map* is a convenient way of thinking about Delaunay triangulations [16]. It is the mapping lifting the 3D point set to a paraboloid in four dimensional space, namely

$$(x, y, z) \rightarrow (x, y, z, x^2 + y^2 + z^2)$$

In the lifted space, the InCircle test is a point/plane orientation test and the Delaunay triangulation is the lower convex hull of the lifted point set.

Once we are in the lifted space it is natural to generalize Delaunay triangulations to power complexes [5] by allowing the lifting coordinate to be specified independently, namely $(x, y, z) \to (x, y, z, l)$. A way to do this is by giving each point a weight so $l = x^2 + y^2 + z^2 - w$. The weight can by interpreted as the square of the radius of a sphere around the point. The resulting lower convex hull is the dual of the power diagram, a Voronoi diagram of the weighted points using the power distance. This interpretation is used in the molecular surface area computations mentioned in the introduction. For simplicity, we will restrict the initial and final weights of the points to be zero.

Since all of our current approaches only look at one time step at a time we will let $\mathcal{P}$ be the coordinates of the points before the time step and $\mathcal{P}'$ the perturbed coordinates. The Delaunay triangulation of a point set will be denoted $\mathcal{D}(P)$. So the overall problem is to compute $\mathcal{D}(\mathcal{P}')$ given $\mathcal{P}$, $\mathcal{P}'$ and, $\mathcal{D}(\mathcal{P})$.

3. KINETIC DELAUNAY

Given a set of continuous trajectories for the points, we can use a kinetic data structure to maintain the Delaunay triangulation during the motion. In contrast to most studies of kinetic data structures [9], in our problem, no trajectory between $\mathcal{P}$ and $\mathcal{P}'$ is specified. As a result we are free to choose it as we see fit in order to minimize the amount of work that is performed. In Section 3.3 we discuss the trade offs involved. We only are interested in events which occur during a narrow interval of time, which affects our choice of type of solver to use. These issues are discussed in Section 3.5. As with all known extant work on kinetic data structures, we will restrict our trajectories to be polynomials of time.

3.1 Kinetic Data Structures Overview

Computational geometry is built on the idea of *predicates* — functions of parameters defining the geometric data set (e.g. point coordinates) which return discrete sets of values. Many predicates reduce to determining the sign of an algebraic or even arithmetic expression on the coordinates of the primitive objects. For example, to test whether a point lies above or below a plane (i.e. the InCircle test under the lifting map), we compute the dot product of the point with the normal of the plane and subtract the plane's offset along the normal. If the result is positive, the point is above the plane, zero on the plane, negative below. The validity of many combinatorial structures built on top of geometric primitives can be proved by checking a finite number of predicates of the geometric primitives, called *certificates*. For a Delaunay triangulation, the certificates are one InCircle test per facet of the triangulation, plus point plane orientation test for each facet and edge of the convex hull.

The kinetic data structures framework is built on top of this view of computational geometry [25]. Let the geometric objects move by replacing each of their coordinates with a function of time. As time advances, the objects now trace out paths through space called *trajectories*. The values of the algebraic functions of the coordinates used to evaluate the certificates now also become functions of time. We call these *certificate functions*. As long as these functions maintain the correct sign, the original data structure is still correct. However, if one of the certificate functions changes sign, the original structure must be updated and some new predicate functions computed. We call such occurrences *events*.

Maintaining a kinetic data structure is then a matter of determining which certificate function changes sign next (i.e. determining which predicate function has the first root after the current time) and then updating the structure and certificate functions.

3.2 Maintaining a Delaunay Triangulation

Maintaining a Delaunay triangulation using a kinetic data structure is well understood in theory and has been implemented numerous times [8].

The predicate functions for kinetic Delaunay triangulation are the determinant of the matrix corresponding to the lifted point/-hyperplane orientation test as was mentioned in Section 2.1. The matrix is:

$$\begin{vmatrix} x_0(t) & y_0(t) & z_0(t) & l_0(t) & 1 \\ x_1(t) & y_1(t) & z_1(t) & l_1(t) & 1 \\ x_2(t) & y_2(t) & z_2(t) & l_2(t) & 1 \\ x_3(t) & y_3(t) & z_3(t) & l_3(t) & 1 \\ x_4(t) & y_4(t) & z_4(t) & l_4(t) & 1 \end{vmatrix} \tag{1}$$

where $l_i(t) = x_i^2(t) + y_i^2(t) + z_i^2(t) - w_i(t)$. If the initial and final triangulations are Delaunay rather than power complexes $w(t_0) = w(t_f) = 0$.

When a certificate fails, a flip must be performed and seven new certificate functions must be computed. However, the negation of the certificate function of the facet/edge being flipped is the certificate function of the edge/facet created by the flip. The function and roots can be cached and reused, leaving only six new ones to be handled. Additionally, each time the motion of a point changes, all the predicate functions corresponding to facets of cells incident to the point must be recomputed and re-solved.

In order to compare the costs of the various trajectory types considered, it helps to establish some notation. There are two logical components to the cost of using a kinetic Delaunay data structure to interpolate between two point sets:

- *trajectory change cost*: the cost of computing and solving the predicate functions for each of the facets of the Delaunay triangulation every time the point set's trajectory changes (including the initial specification of the trajectory).
- *per flip cost*: the cost associated with computing and solving the six predicate functions mentioned in the previous paragraph.

Let f be the number of facets in the final triangulation, $S(d)$ the cost to generate and solve a certificate polynomial of degree d, e_m the number of flips (events) that occur during the motion m and let p be the number of times the motion of all the points is changed after the initial specification. Then the total cost of the kinetic data structure is $(f(p+1) + 6e)S(d)$. Alternatively, we could move each point independently, one after another, and only recompute the certificates involving each point when its trajectory changes. Then, the cost is $(5f + 6e)S(d)$ since each certificate must be recomputed once for each of the five points involved.

If the points are given weights then there is an extra source of certificate functions. Some points, called *redundant points* are points which are above the convex hull in lifted space and therefore not part of the triangulation. As a result, for each point which is incident to four other points in the triangulation, we need to maintain a certificate to verify that the point has not moved above the hyperplane defined by its four neighbors and off the convex hull. In addition we need certificates to track the location of each redundant point. These add a cost $(t + 3er)S(d)$ where t is the number of degree four vertexes and r is the number of redundant points per cell of the triangulation. In our examples, t and r are close to zero since the weights are never very uneven. As a result those components of the cost can be ignored.

3.3 Interpolating Motions

We are now free to choose the motion interpolating between $\mathcal{P}$ and $\mathcal{P}'$ to minimize our total work. We found that for the data sets investigated, the initialization cost dominated, so we chose to focus on minimizing that and ignore the cost of flips. There the cost we consider is the product of the number of times the motion is changed and the cost of generating and solving a certificate function.

3.3.1 *Minimize Trajectory Changes: Linear Interpolation*

We can minimize the number of times the motion of each point changes. This means picking a single motion for each point that interpolates its initial and perturbed positions. The simplest way to do this is by allowing x, y, z to change linearly at the same time. The resulting certificates have degree five.

If we are already dealing with regular triangulations, or are willing to bear the added complexity, we can instead interpolate linearly in the lifted space — i.e. interpolate x, y, z and l linearly. The corresponding Cartesian space motion is the same as before for x, y, and z. The weight varies quadratically with time. If the initial and final weights are 0 and we vary t from 0 to 1 during the interpolation, then $w(t) = (t^2 - t)|\mathbf{d}|^2$ where $\mathbf{d} = (d_x, d_y, d_z)$ is the perturbation vector for the point. Note that the weight is always negative and is bounded by $|\mathbf{d}|^2/4$.

Interpolating in lifted space reduces the degree of the certificates to four at little additional cost. We call these methods *linear* interpolation and *lifted linear* interpolation respectively. The latter is the lowest degree interpolating motion which does not require modifying the motions during the interpolation. The cost is then $(f + 6e_{lli})S(4)$.

3.3.2 *Minimize Degree: Linear Certificate functions*

Generating and solving high degree certificate functions is expensive compared to computing static predicates. Polynomial multiplication is quadratic in the degree (the algorithms with better asymptotic bounds have constants which are too large to be useful for the polynomials in question) and there is overhead from allocating space for all the intermediate values. In addition, solving a degree five polynomial takes three times as long as solving a linear one as is shown in Figure 4.

For all these reasons it is advantageous to minimize the degree of the certificate functions. There are two ways to make the certificate functions linear, either allow one row of the certificate matrix (Equation 1) to vary linearly, or allow one column.

If we allow one row to vary linearly and hold the others constant, it corresponds to moving each point as in *lifted linear* interpolation, but moving them one at a time. We call this method *point at a time* interpolation. Using the above notation, work is $(5f + 6e_{pat})S(1)$ since each certificate must be recomputed five times, once for each point involved.

If, instead, we allow one column to vary linearly and hold the other columns constant, it corresponds to linearly interpolating all points along each coordinate successively (including l). The motion will need to be changed three times during the interpolation, as the coordinate being interpolated shifts from x to y to z to l. We call this method *coordinate at a time* interpolation. The work is $(4f + 6e_{cat})S(1)$. This has better trajectory change cost than *point at a time* interpolation. In this interpolation method the weight can become quite large. If x, y, z are all moved before l, then the maximum weight of a point is $2(\mathbf{x} \cdot \mathbf{d}) + |\mathbf{d}|^2$ where $\mathbf{x}$ is the initial coordinates of the point and $\mathbf{d}$ is the displacement vector, as before. In practice this large change in the weight results in many more events occurring then in *point at a time* interpolation.

3.4 Cost Comparison

Certain aspects of the costs associated with the kinetic data structure based update methods and with rebuilding the triangulation can be compared a priori, namely the costs associated with evaluating the static predicates and generating the kinetic certificate functions.

- Naive computation of an InCircle certificate function where the result is linear takes twice as many multiplications as for the static predicate. We can reduce this to approximately the same number of multiplications as for the static predicates by reordering the input to the determinant computation to avoid linear intermediate results.

- An InCircle test where all of the Cartesian space motions are linear, which results in a degree five certificate function, takes approximately five times as many multiplications to compute as the static determinant.

- An InCircle test where are the trajectories are quadratic (including the lifted coordinate), takes approximately ten times as many multiplications as a static determinant.

These are all underestimations of the amount of work necessary to compute the certificate functions as it ignore the extra overhead associated with memory management and branches.

In our data sets, building a Delaunay triangulation from scratch takes approximately four InCircle predicate evaluations per facet in the final triangulation. Using this we can compare the lower bound on the kinetic data structure cost with the static algorithm. Table 1 shows this comparison. The estimates of the lower bounds on the kinetic data structure update cost and the rebuilding cost are all quite close for the methods we tried, and prohibitive for any higher degree trajectories.

This analysis ignored many types of work associated both with building a Delaunay triangulation and with maintaining a kinetic data structure, however it does capture the most important components of the work. On the static Delaunay side, [12] found that 40-100% of the running time of Delaunay computation was taken by predicate evaluation. That 40-100% includes point/plane orientation tests used during point location, of which there are typically 50% more than the InCircle tests. However, those are lower degree and as a result require fewer than one fourth as many operations, so are not a large fraction of the running time.

3.5 Solvers

There are a number of aspects of the Delaunay update problem which create different requirements on the solvers than with normal kinetic data structure implementations. Unfortunately, we have not yet been able to exploit these differences to our advantage. The key differences are:

- We are only interested in events that occur during a brief window corresponding to the interval between the two frames in question, or even to some shorter interval until the motion is next due to change. Effort spent computing and tracking certificate failure times outside this interval is wasted.

- We need to robustly handle degeneracies and numerical issues.

Both issues can be addressed by using interval based solvers. Extending earlier work published in [26] we have implemented

method	*description*	*cost*	*determinant cost*
linear	move the points linearly in Cartesian space	$(f + 6e_{li})S(5)$	$5f$
lifted linear	move the points linearly in lifted space	$(f + 6e_{lli})S(4)$	$4f$
point at a time	move points one at a time linearly in lifted space	$(5f + 6e_{pat})S(1)$	$5f$
coordinate at a time	move all the points linearly along each coordinate, one after another	$(5f + 6e_{cat})S(1)$	$4f$
quadratic	move all the points linearly along some sort quadratic trajectories in lifted space	$(5f + 6e_q)S(8)$	$8f$
rebuilding	rebuild the Delaunay triangulation from scratch	—	$4f$

Table 1: *A comparison of the various kinetic data structure based methods:* f is the number of facets in the final triangulation, e_m the number of flips (events) caused by motion m, $S(d)$ is the cost to solve a polynomial of degree d. The *cost* column is how many certificate functions will be generated and solved by the kinetic Delaunay update. The *determinant cost* is an estimate the in initialization cost in units of a the cost of a static determinant evaluation. Note that the base costs of the low degree kinetic data structures are very close to that of rebuilding. This agrees with our experimental findings. The cost for quadratic trajectories is too high to be of practical interest.

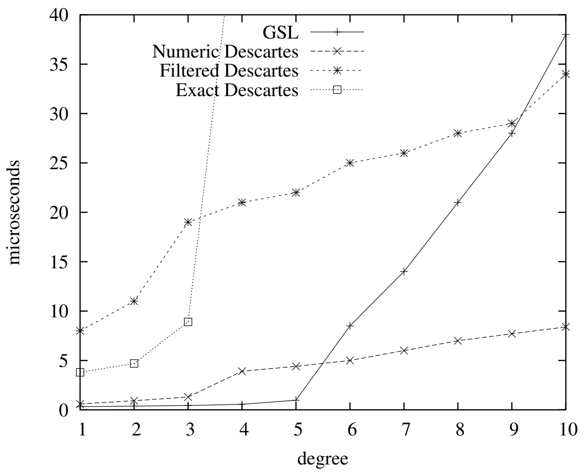


Figure 4: *Solver costs:* The table shows the time in μs of the various solves for isolating roots pf polynomials of specified degree. The exact Descartes solver and filtered Descartes solver both perform exact root isolation operations and allow exact root comparisons. GSL is a numerical, eigenvalue based solver using the GNU Scientific Library [23] which wraps the ATLAS [35] linear algebra package. Currently the eigenvalue based solver is faster than our solvers for low degree polynomials. We expect to be able to bring down the costs of the interval based solvers. The polynomials were generated with integer coefficients chosen uniformly between -1000 and 1000 and the roots were isolated over an interval from 0 to 1. The timings were done on a 1.8Ghz Pentium 4.

solvers that use Descartes rule of sign and Sturm sequences to isolate roots in intervals. These solvers naturally only act on an interval of the real line, and so can ignore roots outside of the interval of interest. Root isolation and comparison can all be implemented using field operations, and so exact comparison of roots can be done. We have also implemented interval based solvers which use floating point filters to accelerate the root isolation and comparison.

Unfortunately, the necessary operations in the kinetic setting are fundamentally predicates (root comparisons) acting on constructions (roots of certificate polynomials), rather than pure predicates, making the filtering process more difficult and costly than in the static case. As shown in Figure 4, the filtered interval solver outperforms the eigenvalue based solver for high degree polynomials, but is more than ten times slower for the sorts of low degree polynomials we are interested in. We expect that we can reduce the gap significantly in the future, but doubt the penalty for using an exact solver will be as low as that for filtered static computations except in the case of linear certificates.

We plan to publish a more detailed discussion of our solver package in a future paper.

4. STATIC UPDATE TECHNIQUES

The kinetic data structure based approaches involve creating a smooth morph between the initial and final triangulation. In most instances this is doing too much work, since we do not care about any of the intermediate state. Therefore, we propose a set of alternative update schemes which only involve computing predicates on the initial and final coordinates and directly transforming the initial triangulation into the final answer. These perform better in practice than the kinetic data structures based approaches, at least in part because they leverage much of the work that has gone into optimizing static geometric algorithms.

The most straight forward such technique is to take each point in turn, remove it from the triangulation, walk to the cell containing its new location and then reinsert it. We call this method *placement*. Single point removals and insertions preserve the Delaunayhood of the triangulation, so the final triangulation is guaranteed to be Delaunay. If motions are small, the walk is cheaper than traversing the point location hierarchy. In practice, this is an extremely poor way to update Delaunay triangulations since all the structure, even the conserved part, is rebuilt. In addition deleting a node from a triangulation is more expensive than inserting one.

A particularly poor case would be $\mathcal{P} = \mathcal{P}'$. The *placement* method would take every point, remove it, re-triangulate the hole and then reinsert the point exactly as it was before. While stationary points can be easily skipped, it is hard to robustly handle rigid transformations or triangulation preserving perturbations.

4.1 Point Removal

A valuable property for any Delaunay update technique would be to do minimal work in the above case where the triangulation structure is unchanged. Let $(\mathcal{T}, \mathcal{Q})$ signify assigning the coordinates of the point set $\mathcal{Q}$ to the vertices of the triangulation $\mathcal{T}$. Using this notation, Delaunay triangulations of $\mathcal{P}$ and $\mathcal{P}'$ are equal when $(\mathcal{D}(\mathcal{P}), \mathcal{P}')$ is Delaunay. In order to verify this, we must first check the orientation certificate for each cell in the triangulation using the perturbed coordinates, $\mathcal{P}'$. This pass verifies that the triangulation is an embedding. Second, we must check the InCircle certificate for each facet using the perturbed coordinates, verifying that the triangulation is Delaunay.

The placement algorithm can be modified as follows. We try to verify that $(\mathcal{D}(\mathcal{P}), \mathcal{P}')$ is Delaunay by checking the above certificates. If it is not, we remove some point, p, from $\mathcal{P}$ and $\mathcal{P}'$. We then compute $\mathcal{D}(\mathcal{P} \setminus p)$ and check if $(\mathcal{D}(\mathcal{P} \setminus p), \mathcal{P}')$ is Delaunay. We repeat the process until we have a set R such that $(\mathcal{D}(\mathcal{P} \setminus R), \mathcal{P}')$ is Delaunay. Once we are done, we can reinsert the points from R in to $\mathcal{D}(\mathcal{P}' \setminus R)$. If the points do not move too far, their location in $\mathcal{D}(\mathcal{P}' \setminus R)$ will be near one of their neighbors in the initial triangulation, so walking through the triangulation from their neighbor will be an efficient method of point location. The removal process is guaranteed to terminate, if only because $(\mathcal{D}(\emptyset), \emptyset)$ is trivially Delaunay.

As a note, the determinant based InCircle tests do not make sense when the points defining the cell are not oriented properly. When the orientation of the points in a cell is reversed, the result of any InCircle tests involving this sphere is also reversed. We can explicitly check the orientation of each cell before evaluating the InCircle predicate, but this is needlessly expensive. As a result, it is best to divide the point removal in to two phases. In the first only check cell orientation, and in the second, when the triangulation is an embedding, check InCircle certificates.

In order to implement this algorithm, a list of invalid facets and cells must be maintained as well as a queue ordering the points for removal based on some sort of score. Removing a point from a Delaunay triangulation only changes cells that were are incident to that point. As a result it is easy to track which cells and certificates may have become valid or invalid in response to the removal of a point. Then, as long as the scoring function for a point only depends on the local neighbors, it can be updated efficiently.

We have investigated a number of heuristics for scoring functions. The first method, gives each vertex a score of one if it is involved in an invalid certificate, and zero otherwise. This means picking any point which is involved in an invalid certificate. We call this method *random* point removal.

As an improvement on *random* point removal, we can assign a score to each point based on how many failed certificates it is involved in. This ranking function is also local so it too can be efficiently maintained. We call this method *worst first* point removal.

The previous heuristics are simply based on combinatorial properties of the triangulation and the predicates. Instead, we can score each point based on how much the perturbation has locally distorted the geometry of the point set. To do this, for each point, p, we can compute the optimal rigid transform of its link vertices from their coordinates in $\mathcal{P}$ to their coordinates in $\mathcal{P}'$. This transform gives a predicted location of p. Each point's score is then how much its location in $\mathcal{P}'$ differs from its predicted location. This method is called *farthest first* and is also local. However, the computations involved are significantly more expensive.

Finally, we can try removing each point from the triangulation and look at how much this improves the score of the triangulation as a whole. We can then remove the point which improves the score the most. Although the ranking function is still local, this method is significantly more expensive than other methods and requires more book-keeping. We call this method *look ahead* point removal.

4.2 Flipping in 3D

Removing a point from a Delaunay triangulation is a fairly expensive process. The algorithm to fill in a hole when a point is removed is quadratic in the number of neighboring vertices [13], which is typically 14 or 15. In addition, many new cells/facets are created all of which must be checked for validity under the new coordinates. Flips are a much cheaper alternative, requiring constant time updates to the triangulation and only creating six facets whose InCircle certificates have to be checked.

With this in mind, an alternative algorithm is to remove points until all cells are properly oriented — i.e. the triangulation, $(\mathcal{D}(\mathcal{P} \setminus R), \mathcal{P}')$ is embedded. Then we can try to use Delaunay edge flips to finish the conversion to Delaunay. We call this method *hybrid*. Most of the time, we will succeed. Sometimes though, the algorithm will reach an un-flippable conformation.

Unfortunately, the previous solution, removing points, can no longer be safetly applied in this case, since the triangulation is no longer necessarily an embedding using the original coordinates. In addition, the triangulation is not Delaunay with the perturbed coordinates and the hole created by removing a point from a non-Delaunay triangulation in 3D may not be able to be filled with cells [32].

Fortunately, flipping rarely gets stuck in practice, so it is acceptable to use a very expensive handler when it does. For the time being we try to find a point which is adjacent to a non-Delaunay cell, but which can be removed from the triangulation using the new coordinates. Failing that we just recompute the triangulation from scratch, using the preexisting connectivity to speed point location.

One alternative would be to try a Markov walk-like process of flipping away from Delaunay and then trying to flip back to Delaunay. We have not thoroughly investigated this route.

Another alternative is to try to find a subset of the vertices that can be removed so that the boundary of the hole is Delaunay. Then the hole can be filed by computing the Delaunay triangulation of the boundary vertices and pruning cells that are outside the hole. The result may not be Delaunay along the boundary, so we must proceed with flipping.

5. EXPERIMENTAL RESULTS

We tested our algorithms on frame pairs taken from several different types of simulation. Properties of the data are shown in Tables 2 and 3. The simulations are as follows:

- **molecular simulations**: *hairpin* is a short 177 atom beta hairpin and Staphyloccocal protein A, PDB code 2SPZ, (*protein A* for short), a 601 atom globular protein. The simulations were performed using the Tinker package [34] with 2fs time steps.

- **muscle simulation**: *bicep* is volumetric data from a simulation of a bicep contracting. The points move comparatively little between the frames, as is shown in Table 2.

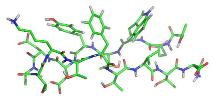
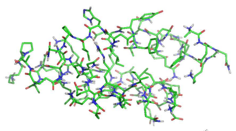
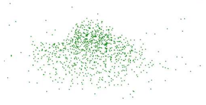
simulation	*picture*	*points*	*cells*	*ephemeral cells*	*InCircle tests*	*tolerance (%)*	*20% tolerance (%)*
hairpin		177	1114	2554	7516	8.5	1.5
protein A		601	3943	10250	31430	8.6	1.5
bicep		3438	21376	66553	210039	9.0	1.6
falling objects		1000	6320	17137	52958	12	1.7
shifting objects		1000	6381	17742	55299	13	1.2

Table 3: *Attributes of the static Delaunay triangulation: Ephemeral cells* are cells created during the construction process which are not in the final triangulation. There were generally three times as many ephemeral cells as cells in the final triangulation. Their number gives some idea of the excess work done by the rebuilding process. *Tolerance %* is the average fraction of the local edge length that points must move to invalidate on in circle certificate. The *20% tolerance* is the fraction of the local average edge length that points need to move to destroy 20% of the certificates. Very small motions compared to the edge length can invalidate many cells in a Delaunay triangulation.

simulation	*timesteps*	*failed cells (%)*	*motion (%)*
hairpin	1	3.5	1.1
	2	8.3	2.1
	4	14	4.0
	8	25	7.3
protein A	1	8	3.1
	2	14	5.7
	4	24	14
bicep	1	.64	.03
	2	1.2	.07
	4	2.3	.17
falling objects	1	1.5	1.8
	2	2.6	4.1
	4	5.2	11
	8	9.3	37
shifting objects	1	2.2	1.0
	2	3.0	2.0
	4	4.0	3.8
	8	6.0	7.5

Table 2: *Attributes of the motion:* The *failed cells* is how many cells from the Delaunay triangulation of the first frame were not in the Delaunay triangulation of the second. This varied quite significantly between data sets, with the molecular simulation datasets being the most rapidly changing ones. The *motion* is the average of how far each point moved divided by its average edge length.

simulation	*degree: 1*	*4*	*10*	*rebuild time*
hairpin	8.4ms	210ms	620ms	15ms
protein A	29ms	790ms	5.6s	69ms
bicep	250ms	4.5s	25s	430ms
falling objects	72ms	1.2s	16s	153ms
shifting objects	55ms	1.2s	17s	120ms

Table 4: *Kinetic Delaunay initialization costs:* Here we show the costs to initialize the event queue for various degree polynomial trajectories. This means compute the failure time for each certificate, given the initial conformation and trajectory and insert these events into the queue. The cost of computing the triangulation from scratch is shown for comparison. Note that a linear certificate based kinetic Delaunay must have at least four initialization passes, the cost of just one is shown.

- **falling objects simulation**: is data taken from a simulation of a collection of 1000 objects dropped into a pile. Initially the objects fall through the air with occasional contacts, but later in the simulation they form a dense, although still shifting pile. We call the two phases *falling objects* and *shifting objects*. The data came from research published in [24].

The CGAL Delaunay triangulation package was used for all of our static computations. This is a quite mature Delaunay triangulation package, which sets a rather high bar for our algorithms. However, we think this reflects most practical situations since fast and robust static Delaunay computation is well understood and a number of good packages are available both commercially and under free software licenses.

5.1 Kinetic Delaunay Results

The kinetic data structure based approaches are not currently competitive with static Delaunay computation, as is shown in Ta-

		linear		*lifted linear*		*coordinate at a time*		*point at a time*	
simulation	*rebuild*	*event*	*time*	*events*	*time*	*events*	*time*	*events*	*time*
hairpin	15ms	13	350ms	13	220ms	126	32ms	21	37ms
protein A	69ms	164	1.46s	134	840ms	198	120ms	217	140ms
bicep	430ms	106	7.6s	98	4.5s	160	1.1s	101	1.1s
falling objects	153ms	36	2.3s	30	1.6s	150	300ms	78	330ms
shifting objects	120ms	89	2.1s	45	1.7s	128	220ms	69	270ms

Table 5: *Kinetic Delaunay update costs:* running times and event counts for different interpolating motions are shown as well as the rebuilding time. The linear certificate based methods are within a factor of two of rebuilding, but we suspect that novel optimizations are necessary to have them outperform rebuilding. The kinetic data structures were implemented using the kinetic data structures framework presented in [27] and the CGAL triangulation package. Rebuilding was done using the CGAL Delaunay triangulation code. Timings were done on a 1.8Ghz Pentium 4.

ble 5. The running time was dominated by the initialization cost—i.e. computing the initial failure time for the certificates, shown in Table 4. For the nonlinear certificates this computation is complicated by the need to multiply polynomials of unknown degree, requiring many loops and memory management. For the linear certificate function based motions we were able to use a specialized polynomial representation which removed much of this overhead.

As a note, the static rebuilding was done using filtered predicates, so it is exact, where as we were forced to use purely numerical methods for the kinetic data structures as filtering is still too slow for low degree nonlinear polynomials. The non-filtered static construction code ran in to numerical issues with the shifting objects data set since many objects are resting on the support platform. Inexact kinetic Delaunay did not have any problems.

The fastest kinetic Delaunay based method was the *coordinate at a time* interpolation with *point at a time* interpolation close behind. The later algorithm actually generally had fewer events, due to the smaller changes in the weights. However its extra initialization cost made it slightly slower. Both techniques were about a factor of two more expensive than recomputing the triangulation. We are not sure if further optimizations will bring the running time low enough to be competitive. The most promising optimization one would be to reorder the determinant computation to reduce the number of multiplications to be closer to that of the static computation as was described in Section 3.4.

An important optimization used was to cache redundant matrix minor calculations. Each cell in a Delaunay triangulation is involved in up to four certificate computations, one for each facet. Each of the certificate generation computations involving a cell is that of a matrix determinant (Equation 1) with one row differing and can be seen as testing whether the additional point is above or below the lifted hyperplane defined by the cell. The matrix minors defining this hyperplane can be cached in a lazy manner, speeding up the certificate polynomial computation. Since each InCircle test involves two cells, we can check to see if either cell has the appropriate minors cached before generating them and computing the certificate. This optimization reduces the running time by approximately 25%.

5.2 Static Update Results

The static predicate based methods leverage much of the existing Delaunay triangulation computation code and consequently are much simpler to implement and readily track most improvements in Delaunay computation. We implemented the techniques on top of the CGAL Delaunay triangulation code and predicates. Running times are shown in Table 7.

Of the purely point removal based techniques, we found the *worst first* point removal method to be the fastest, followed closely by the

simulation	*step size (frames)*	*stuck frequency (%)*
hairpin	1	2
	2	4
bicep	1	5
	2	33
falling objects	1	2
	2	5
	4	17
shifting objects	1	23
	2	31

Table 6: *Frequency of flipping getting stuck:* for each simulation, the frequency with which Delaunay flips get stuck is shown for various step sizes. When Delaunay flips get stuck, expensive steps have to be taken, such as rebuilding the triangulation from scratch. Fortunately, this happens infrequently on most models, so the overall cost is small.

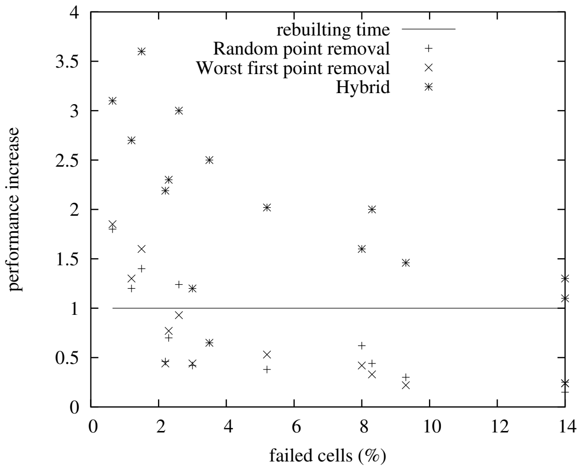


Figure 5: *Static update performance as a function of failed certificates:* the plot shows the ratio of the running times of the *random* point removal, *worst first* point removal and *hybrid* methods to rebuilding compared to the number of invalid cells in the triangulation. With the exception of the two points from the protein A simulation, the hybrid method performed quite well.

			placement	*random*		*worst first*		*hybrid*		
simulation	*steps*	*rebuild*	*time*	*removed*	*time*	*removed*	*time*	*removed*	*flips*	*time*
hairpin	1	15ms	84ms	13	23ms	15	23ms	0	17	**6.1ms**
	2	16ms	87ms	27	34ms	31	45ms	0	39	**7.5ms**
	4	16ms	87ms	57	62ms	51	58ms	9	50	**12ms**
protein A	1	69ms	320ms	111	213ms	98	163ms	7	128	**42ms**
	2	67ms	310ms	237	420ms	150	270ms	33	183	**60ms**
bicep	1	420ms	1.7s	21	**231ms**	19	**228ms**	1	17	**133ms**
	2	420ms	1.7s	42	**350ms**	38	**320ms**	1	37	**153ms**
	4	430ms	1.8s	80	600ms	70	540ms	3	67	**180ms**
	8	420ms	1.8s	955	5.3s	630	4.2s	13	469	570ms
falling objects	1	153ms	510ms	37	**110ms**	31	**95ms**	1	36	**43ms**
	2	160ms	530ms	73	**129ms**	55	171ms	1	66	**53ms**
	4	160ms	520ms	158	419ms	107	300ms	3	137	**79ms**
	8	160ms	524ms	460	524ms	350	725ms	4	247	**109ms**
shifting objects	1	123ms	510ms	80	264ms	58	267ms	8	31	**56ms**
	2	120ms	510ms	104	280ms	72	270ms	52	27	**100ms**

Table 7: *Static update performance:* the times to rebuild the triangulation and the number of removals for the various heuristics are shown. Times in boldface are ones which are better than rebuilding (the left most time). The time of the *look ahead* point removal was an order of magnitude worse than the others, although it did manage to remove 50% fewer points on some of the models.

random point removal. They both outperformed rebuilding on the bicep and the falling objects simulations. They performed especially poorly on the hairpin and shifting objects data. The former is probably due to the comparatively small cost associated with traversing the point location hierarchy in the hairpin model, the smallest tested. The latter, seems to be related to handling the many degeneracies along the bottom of the point set. In general, they outperformed rebuilding when 1-2% of the cells were invalid after perturbation.

Our attempt to use the actual geometry of the motion to order points for removal was useless, as the magnitude of the motion vectors were too uniform in the simulations and uncorrelated with actual changes in the Delaunay triangulation. Consequently, the results are not shown. In addition, the cost of computing the rigid transforms was large, making the method quite slow overall. Placement was also slow, as could be predicted.

The fastest method overall was the *hybrid* method. It was able to update the triangulation faster than rebuilding in all the simulations, even over multiple time steps. It was over three times as fast as rebuilding for the falling objects and the bicep. Overall, when less than 2% of the cells are invalidated by the perturbation, performing flips is two to three times faster than rebuilding. As the percentage rises to ten, the advantage goes to rebuilding. However, the simulations we looked at were firmly in the 2-3 times faster domain. Figure 5 shows this comparison.

The main draw back of performing flipping is that sometimes it gets stuck and an expensive recovery step has to be taken. Frequencies of getting stuck on the various data sets are shown in Table 6. With the exception of the shifting objects, the frequency of getting stuck was less than 10% for single time steps. This means that even if we rebuild after getting stuck, the speed penalty is small.

6. CONCLUSIONS AND FUTURE WORK

Using Delaunay flips to update a Delaunay triangulation is much faster than rebuilding with the perturbation sizes used in most simulations. Other static Delaunay based techniques are also faster than rebuilding in many cases. However, kinetic data structures based approaches need significantly more optimization to be competitive. The most promising kinetic techniques use linear certificate functions and are within a factor of two of rebuilding.

Most of the certificates computed in the kinetic Delaunay-based techniques did not fail during the time interval in question. This suggests that we perform filtering before generating the certificates, skipping the expensive generation step for ones which can be shown not to fail. For most of the data sets, the perturbation distance was less that the tolerance of half of the certificates, providing one possible filtering mechanism. We have only just begun to investigate this issue.

When we remove points and reinsert them as part of the point removal algorithm, the insertion/removal operations must be done one at a time. This can lead to the creation of ephemeral tetrahedra — tetrahedra created after the removal of one point but then destroyed shortly thereafter when a later point is removed. It could speed things up if we were able to efficiently patch holes created by removing more than one point. However, these holes have a much less restricted structure than those created by the removal of a single point. They are not star shaped and may not be genus 0. While such holes are guaranteed to be triangulable [32], there are no fast algorithms for doing so.

Our handling of un-flippable configurations is extremely expensive. It would be advantageous to have a technique to handle this case that was more local and that reduced the problem to a slightly simpler one.

There has recently been a great deal of work on how bound the performance of geometric algorithms based on properties of the input data. Example of this include ϵ-sampling [2], local graphs [18], as well has various point set attributes which allow the complexity of the 3D Delaunay triangulation to be bounded [4, 19] among others.

In contrast there has been little accomplished with regards to bounding work in needed to maintain structures under motion. The efforts by the kinetic data structures community have generally bounded the number of changes that occur as points are allowed to move over specified complexity trajectories for an unlimited amount of time. These bounds are not useful in cases where only a finite time interval is of interest. Moreover the existing bounds are loose, $O(n^4)$ changes for a kinetic Delaunay data structure in 3D, while it is generally believed that the correct bound is closer to $O(n^3)$.

We would like to produce models of moving objects which allow us to bound the number of combinatorial changes during a period of motion. We have looked at ideas for measuring local coherence such as was used in the *farthest first* point removal heuristic, however, we have not been able to relate them to the amount of work performed.

In general, the problem of how to update geometric structures after small displacements of their defining elements defines an interest research area. The Delaunay triangulation may not be the best structure to maintain in such a setting, because it is canonical and very sensitive to the locations of its defining points. Non-canonical structures, such as the deformable spanner of [21], can behave much better in this respect.

7. ACKNOWLEDGMENTS

This work was supported in part by NSF grants CCR-0204486, ITR-0086013, ITR-0205671, ARO grant DAAD19-03-1-0331, as well as by the Bio-X consortium at Stanford.

References

[1] M. Abellanas, F. Hurtado, and P. A. Ramos. Structural tolerance and Delaunay triangulation. *Information Processing Letters*, 71(5–6):221–227, 1999.

[2] N. Amenta and M. W. Bern. Surface reconstruction by voronoi filtering. In *Symp. on Comp. Geom.*, pages 39–48, 1998.

[3] N. Amenta, S. Choi, and G. Rote. Incremental constructions con BRIO. In *Proc. of the 19th conference on Comp. geometry*, pages 211–219, 2003.

[4] D. Attali, J.-D. Boissonnat, and A. Lieutier. Complexity of the delaunay triangulation of points on surfaces the smooth case. In *Proc. of the 19th conference on Comp. geometry*, pages 201–210. ACM Press, 2003.

[5] F. Aurenhammer. Power diagrams: properties, algorithms, and applications. *SIAM J. on Computing*, 16:78–96, 1987.

[6] D. Bandyopadhyay and J. Snoeyink. Almost-Delaunay simplices: Nearest neighbor relations for imprecise points. In *ACM-SIAM Symp. on Discrete Algorithms*, pages 403–412, 2004.

[7] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. In *ACM-SIAM Symp. on Discrete Algorithms*, pages 747–756, 1997.

[8] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *ACM-SIAM symp. on discrete algorithms*, pages 747–756. Society for Industrial and Applied Mathematics, 1997.

[9] J. Basch, L. J. Guibas, C. D. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. of the 13th annual symp. on Comp. geometry*, pages 388–390, 1997.

[10] P. Cignoni, C. Montani, and R. Scopigno. A fast divide and conquer Delaunay triangulation algorithm in E^d. *Computer Aided Design*, 30:333–341, 1998.

[11] O. Devillers. Improved incremental randomized Delaunay triangulation. In *Proc. of the 14th annual symp. on comp. geometry*, 1998.

[12] O. Devillers and S. Pion. Efficient exact geometric predicates for delaunay triangulations. In *Proc. 5th Workshop Algorithm Engineering Experiments*, pages 37–44, 2003.

[13] O. Devillers and M. Teillaud. Perturbations and vertex removal in a 3d Delaunay triangulation. In *ACM-SIAM symp. on Discrete algorithms*, pages 313–319. Society for Industrial and Applied Mathematics, 2003.

[14] H. Edelsbrunner. The union of balls and its dual shape. *Discrete Comp. Geom.*, 13:415–440, 1995.

[15] H. Edelsbrunner and P. Koehl. The weighted volume derivative of a space-filling diagram. In *Proc. of the National Academy of Science*, volume 100, pages 2203–2208, 2003.

[16] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. In *Proc. of the 1st annual symp. on comp. geometry*, pages 251–262. ACM Press, 1985.

[17] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. In *Proc. of the 8th annual symp. on comp. geometry*, pages 43–52. ACM Press, 1992.

[18] J. Erickson. Local polyhedra and geometric graphs. In *Proc. of the 19th annual conference on Comp. geometry*, pages 171–180. ACM Press, 2003.

[19] J. Erikson. Dense point sets have sparse Delaunay triangulations. In *ACM-SIAM Symp. on Discrete Algorithms*, pages 125–134, 2002.

[20] G. D. Fabritiis and P. V. Coveney. Dynamical geometry for multiscale dissipative particle dynamics. *arXiv:cond-mat:0301378*.

[21] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and their applications. 2004.

[22] M. Gavrilova and J. Rokne. Collision detection optimization in a multi-particle system. *International Workshop on Comp. Geom. and Applications*, 2331:105–114, 2002.

[23] GNU scientific library. http://www.gnu.org/software/gsl/.

[24] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics*, 22(3):871–878, 2003.

[25] L. Guibas. Kinetic data structures: A state of the art report. In *Proc. 3rd Workshop on Algorithmic Foundations of Robotics*, 1998.

[26] L. Guibas and M. Karavelas. Interval methods for kinetic simulations. In *Proc. of the 15th annual symp. on comp. geometry*, pages 255–264, 1999.

[27] L. Guibas, M. Karaveles, and D. Russel. A comp. framework for handling motion. In *ALENEX 2004, Lecture notes in Compute science*, 2004. to appear.

[28] L. Guibas, D. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[29] L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. In *Proc. of the 18th annual symp. on comp. geom*, pages 33–42. ACM Press, 2002.

[30] L. J. Guibas and L. Zhang. Euclidean proximity and power diagrams. In *Canadian Conference on Comp. Geom.*, pages 90–91, 1998.

[31] B. Joe. Three-dimensional triangulations from local transformations. *SIAM J. on Scientific and Statistical Computing*, 10:718–741, 1989.

[32] J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proc. of the 14th annual symp. on Comp. geometry*, pages 76–85, 1998.

[33] P. Su and R. L. D. III. A comparison of sequential Delaunay triangulation algorithms. In *Proc. of the 11th annual symp. on Comp. Geom.*, pages 61–70, 1995.

[34] Tinker molecular simulation package. http://dasher.wustl.edu/tinker/.

[35] R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.