Modeling Motion

Amber Stillings
Computational Geometry Project
December 8, 2005

This paper is meant to be an overview of the methods used to model motion in the computational geometry, mesh generation, and graphics communities. The techniques used to model motion, as well as the problems with these techniques, are discussed.

The primary focus of the computational geometry section of this paper will be kinetic data structures. An explanation of what kinetic data structures are is given, and issues that remain are discussed. Techniques for modeling motion in several geometric algorithms and structures are also explained, including extent problems, kd-trees, binary space partitions, collision detection, and Delaunay triangulations/Voronoi diagrams.

Triangulations are central to mesh generation, and therefore they will be the main focus of this section. Mesh properties, generation, refinement, improvement, and maintenance through deformation are addressed in this section. Mesh deformation is then discussed in a space-time setting, and an application of deformable meshes is shown.

The graphics section of this paper focuses on physically based models geometric models of motion. Many different models exist using both of these approaches, and a couple of them are discussed. Hybrid models have also been proposed, and a couple of these are described.

Computational Geometry

Modeling motion in the computational geometry setting is a topic still in need of research. Many new algorithms and techniques have been discovered in the past 10-20 years, but many questions remain open. The following is a description of some of the recent advances in this area, as well as some of the current open questions. In particular, Kinetic Data Structures are discussed, as well as means of evaluating them and their limitations to date. This discussion is followed by several geometry problems that have been applied to moving objects, including extent problems, kd-trees, binary space partitions, collision detection, and Delaunay triangulations/Voronoi diagrams.

Kinetic Data Structures

When modeling motion, the data must be sampled and the combinatorial structure of the data must be updated periodically. Deciding when and how often to sample the data can be problematic. Many applications find the smallest time interval for expected change, and use this interval throughout as the sampling interval.   Finding this interval is not always easy though, and estimates might not have desirable effects. For example, choosing an interval that is too large can lead to a lag when updating. This causes data to be incorrect at times, and the motion being modeled will not be smooth. Choosing an interval that is too small on the other hand can lead to too many computations, slowing the algorithm altogether.

Basch, Guibas, and Hershberger [9] introduced the kinetic data structure framework in 1997 to address this problem. A kinetic data structure (KDS) works similarly to plain sweep algorithms: it looks at the current data, and predicts when events will occur that might change the configuration of the data. These events are then scheduled using an event queue. Certificates are used to ensure that the current configuration being stored is correct. These certificates assert an attribute of the data that is necessary for the configuration. When an event occurs, a certificate will fail. This certificate, and perhaps others, will need to be updated. The combinatorial structure of the data might also need to be updated. Therefore, if none of the certificates are in a state of failure, the correctness of the current configuration of the data being stored is guaranteed.

Normal time analysis fails when analyzing a kinetic data structure. This is because a KDS is used in a real-time setting. Methods of evaluating kinetic data structures are then needed, and Guibas [17] suggested the following criteria be used:

- **Responsiveness** – A kinetic data structure is responsive if at the time of a certificate failure, the time necessary to fix the certificates is small.
- **Efficiency** – Efficiency is a measure of the maximum number of certificate failures versus the actual number of failures in the configuration of the data. A kinetic data structure is efficient if the worst case number of certificate failures is similar to the worst case actual number of failures.
- **Locality –** The locality of a kinetic data structure is the measure of how many certificates a single object is part of. A local KDS will keep this number small. Locality is important as it impacts the time necessary to update certificates when they fail. If the KDS is not local, many certificates will need to be updated at the time of failure.
- **Compactness** – A KDS is compact if the number of certificates needed is linear, or almost linear, to the number of objects being modeled. This is important because the time necessary to update certificates will be lower if there are fewer certificates, and also because this limits the storage requirements of the KDS.

Many geometric algorithms and structures have been modified using kinetic data structures to model motion. A very simple example is explained in the next section to illustrate a kinetic data structure in action.

Topmost point in 1-D

An easy example of using a KDS to model motion is shown by tracking the topmost point of a set of points moving along the y-axis. Consider this example as if no collisions occur (think of the points as spread out over the x-axis, all moving vertically along the y-axis). A KDS can easily be used to track the topmost point by using a simple divide-and-conquer algorithm with certificates. Think of the certificates as a binary tree structure. The certificate at the root of the tree certifies which of the two subtrees (certificates) has the topmost point, in other words, it says whether point x is greater than point y or vice versa. This is then recursively stored down the tree. This KDS is efficient as the number of certificate changes is bounded by *O(nlogn)* (refer to [17] for more details). At the time of a certificate failure, the certificates can be fixed in *O(logn)* time, as the changes just need to be cascaded up the tree. Following from this, the locality is also bounded by *O(logn)*. The KDS is also compact, as the tree has *O(n)* certificates. Therefore, this kinetic data structure is efficient, responsive, local, and compact.

Algorithmic Issues

Agarwal et al describe in [1] algorithmic issues that still need to be addressed in order to make KDS and other motion models realistic, including:

- **Motion Sensitivity** – Most current algorithms treat objects as individual sources of motion. Little has been done to correlate the motions of these objects, even when the objects are interconnected in the real world. In other words, the motion of an object is often dependent on the motion of other objects, and little has been done in current algorithms to model this.
- **Trade-offs in complexity measures –** While many KDS algorithms have been found for modeling motion that are efficient, responsive, local, and compact, there are still many

geometric problems where this is not the case. In an effort to solve these, appropriate trade-offs might be necessary. For example, a KDS might be built that is efficient but not compact or vice versa in order to solve a problem for which no KDS is known that satisfies all four criteria. As with many other algorithms, a trade-off between accuracy and time can also be made in order to find approximately correct solutions quickly.

- **Flexible Scheduling –** Kinetic data structures are dependent on the assumption that events are scheduled in the event queue in the correct order, and that no two events are simultaneous. Calculating the times of these events can be costly and inefficient. Agarwal et al [1] suggest adding fixed-time sampling to the algorithm to fix problems with events that are incorrectly scheduled. More research needs to be done to address these scheduling issues.
- **Decentralization –** The current KDS framework is only suitable to centrally located applications. Maintaining the event queue would be a bottleneck in any decentralized application, and certificates could be costly to maintain as other factors such as bandwidth can come into play in fixing them.

Extent Problems

The extent of a set of points is described by measures such as the points' width, diameter, and minimum enclosing rectangle. The diameter of a set of points is found by calculating the convex hull of those points, finding all sets of antipodal pairs of points, and finding the maximum distance between these antipodal pairs. An example diameter of a point set is shown in figure 1 below. According to Agarwal et al [8], when the points are in motion, the diameter is more easily found from the dual of this problem. The dual of the points creates an upper (resp. lower) envelope corresponding to the upper (resp. lower) convex hull of the point set. Each vertex on the convex hull will correspond to a line segment on the envelope. This makes the problem easier because these segments can be merged in x-order, and overlapping intervals indicate antipodal pairs of vertices.

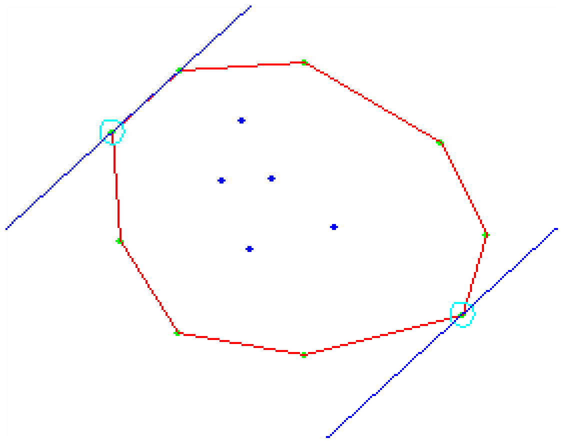

Figure 1: The diameter of this point set is the distance between the two circled antipodal points. (Generated with [24].)

To keep the diameter then in a moving set of points, the merged upper and lower envelopes are kept and this order is what makes the certificates. When a certificate fails, it indicates that two points are no longer antipodal, and an update must occur. If this antipodal pair defined the diameter, or if the new pair of antipodal points have a greater distance between then, then the diameter of the point set has changed.

The definition of the width of a point set is the distance between the closest two parallel lines with all the points in the set lying between the lines. In reality, this corresponds to one line that extends from an edge on the point set's convex hull, and the other line intersects a vertex on the convex hull. Figure 2 illustrates this below. Due to its similarity to diameter, the width of a moving point set is found in a very similar way to the diameter of the set, where antipodal edge-vertex pairs are found using the dual upper and lower envelopes, as before. When finding the diameter, the maximum separation is found. When finding the width, the minimum separation is found instead.

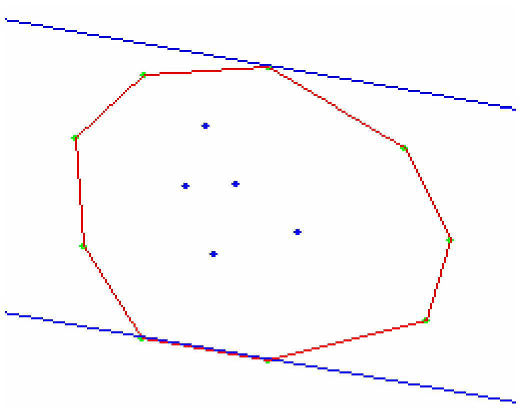
Figure 2: The width of this point set (the same point set as in figure 1) is the distance between the two parallel lines shown. (Generated with [24].)

The minimum enclosing rectangle of moving points is also found using a similar strategy. (An example of a minimum enclosing rectangle is shown in figure 3 below.) Now four envelopes must be calculated: the upper, lower, left, and right envelopes. Antipodal pairs are found in the upper and lower envelopes as well as the left and right envelopes. Then the upper/lower pairs are compared with the left/right pairs to see if there are supporting perpendicular lines. When these lines exist, a bounding rectangle can be formed for the entire set of points through these four points. All of these bounding rectangles are then checked for the minimum.

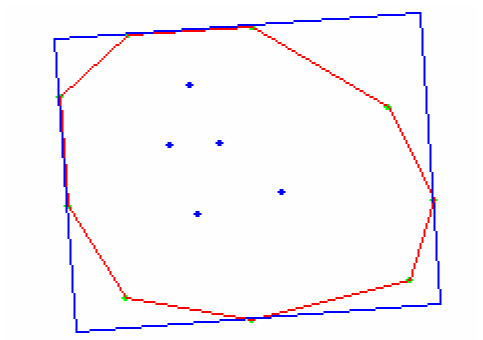
Figure 3: The minimum enclosing rectangle of this point set (the same point set as in figure 1 and figure 2) is shown. (Generated with [24].)

kd-Trees

Agarwal, Gao, and Guibas discuss methods of dealing with kd-trees with moving points in [6]. Recall that kd-trees use bounding boxes in a tree structure for range queries. kd-trees work by finding the median of the points, and dividing the points into two partitions with a vertical line through the median, then recursively finding the median of the partitions, dividing the partitions by alternating horizontal and vertical lines. These partitions can be formed into a tree, where a node's children are the two partitions formed by dividing the partition by its median. An example kd-tree and corresponding point set are shown in figure 4 below. Agarwal et al suggest using kd-trees that allow for overlapping children or children of unequal magnitude. This relaxed version of the kd-tree is easier to maintain as approximate medians are used. The algorithm suggested for maintaining the kd-tree uses a KDS whose certificates guarantee the accuracy of the tree.

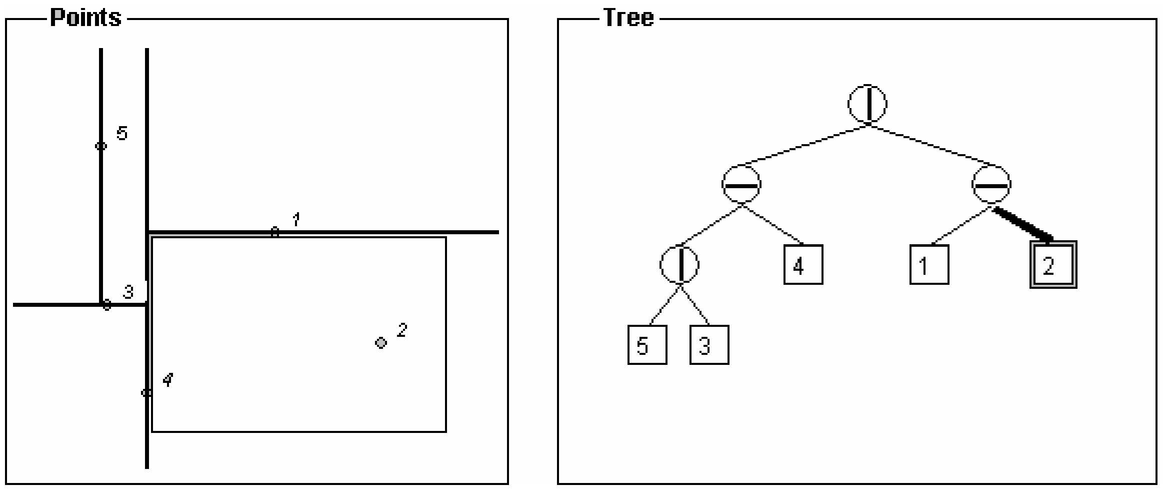

Figure 4: The point set on the right is recursively divided into regions based on the median of the region. The tree on the left is the kd-tree corresponding to this partitioning. (Generated with [27])

Binary space partitions

Agarwal et al describe a KDS algorithm for maintaining binary space partitions as line segments move in [4] and [3]. Binary space partitions keep a tree structure representing the configuration of line segments, in a way similar to kd-trees. A KDS algorithm is used to model motion by keeping certificates that invalidate when line segments move into empty regions, flip to a vertical line, or flip to a region's boundary line. This algorithm is responsive, but not necessarily efficient. Research still needs to be done in this region to answer many questions, including how to build compact binary space partitions, and also how to calculate event times to limit the number of unnecessary certificate updates.

Collision Detection

Collision detection between polygons is an important theme for many applications, including robotics. This problem has been approached in varying ways, and two of these methods are detailed here.

Kirkpatrick et al [21] describe a method of collision detection among simple polygons using a kinetic data structure. A pseudo-triangulation is used between simple polygons, and certificates maintain the corridors, or free space between polygons. A pseudo-triangulation differs from a normal triangulation in that it allows the edges of the triangles to be sets of lines segments that are concave. This is illustrated in figure 5 below. When events cause these certificates to fail, the polygons can be checked for collision and the certificates can be updated. (Agarwal et al also describe a method of collision detection using pseudo-triangulations in [5].)

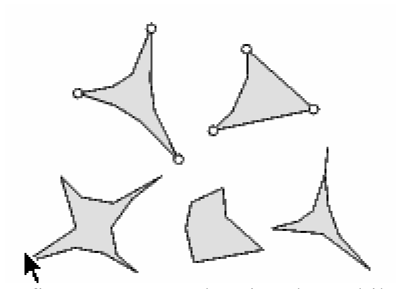

Figure 5: [5] The top two figures are pseudo-triangles, while the bottom three are not.

Collision detection is also done using bounding volumes. These volumes are used because collision detection is easier with these easier shapes. The problem then breaks down into two sub-problems: detecting when these bounding volumes intersect, and then a finer detection of polygons. Guibas, Nguyen, and Zhang propose using zonotopes as bounding volumes in [19]. Zonotopes are the Minkowski sum of line segments, and they are attractive bounding volumes because optimal zonotopes can be easily formed, and the actual zonotope does not need to be constructed: the description of their defining line segments is enough to detect collisions. An example zonotope is shown in figure 6 below. Agarwal et al describe in [2] using a hierarchy of bounding spheres to test for collisions between deformable beads and also self-collision.

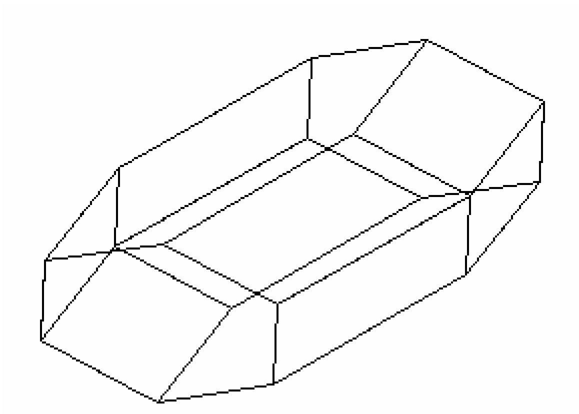

Figure 6: An example Zonotope, which is the Minkowski sum of line segments. (Generated with [26].)

Voronoi Diagrams and Delaunay Triangulations

A Voronoi diagram is a partitioning of space into regions, where all points within a region are closest in distance to the same point. A Delaunay triangulation is the dual of a Voronoi diagram. Both Voronoi diagrams and Delaunay triangulations have important applications in many areas of computational geometry as well as mesh generation and graphics. As such, these structures will be mentioned again in later sections of this paper.

Maintaining a Voronoi diagram/Delaunay triangulation of a set of moving points is a well studied problem in computational geometry, mainly because a tight upper bound for the number of configuration changes in the diagram/triangulation is unknown. The lower bound is known to be n^2 where n is the number of points in the set. The upper bound is believed to be $O(n^2)$, but this has never been proven.

Chew shows a bound of $O(n^2\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann's function for the L_1 and L_∞ Voronoi diagram of moving points in [11]. Delaunay triangulations have the property that any 3 endpoints of segments in the triangulation form a circle, and no other endpoints are contained in this circle. This property is tested in what is called the InCircle test, and an example test is shown on the Delaunay triangulation shown in figure 7 below.

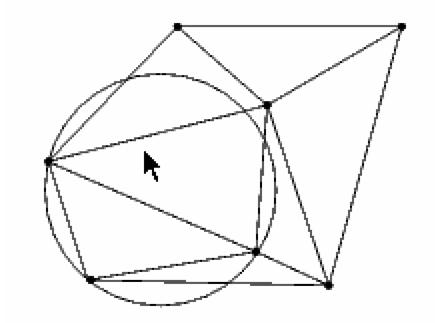

Figure 7: [10] This is a Delaunay triangulation with an example InCircle test. The test indicates that the region is Delaunay because no other vertices are contained in the circumcircle of three vertices on a triangle.

Because of this property, a Delaunay triangulation changes configuration when 4 points become co-circular. The L_∞ Voronoi diagram is not the same as the normal (L_2) Voronoi diagram, as shown in figure 8 below. Instead of circles, the test used to determine whether a Delaunay triangulation is valid is a square (the L_1 metric uses a square tilted 45 degree angle).

The bound is then easily shown for this *L_1* Voronoi diagram by showing that the number of points that can intersect the square is bounded for both corner and noncorner edges. Unfortunately, this does not translate to the normal *L_2* Voronoi diagram as a circle has no edges (or conversely, an infinite number of edges).

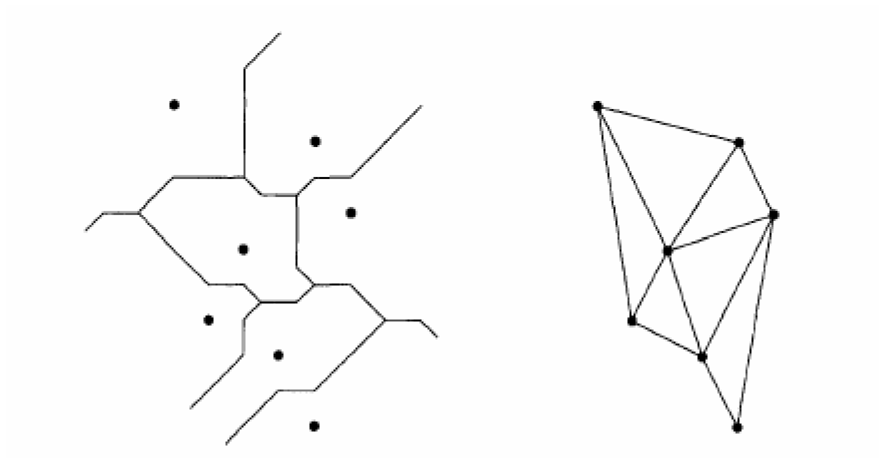

Figure 8: [11] The L_∞ Voronoi diagram and corresponding Delaunay triangulation for a set of points. Notice that the Voronoi diagram does not form regions shaped as convex hulls, as in the normal L_2 Voronoi diagram.

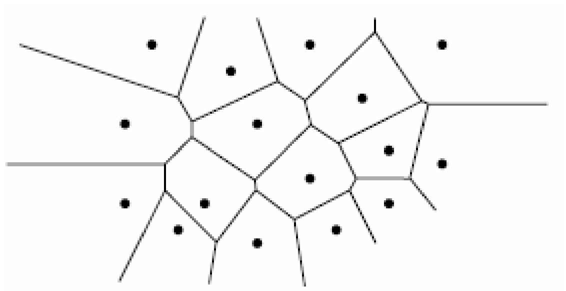

Figure 9: [25] This is a normal L_2 Voronoi diagram, for comparison with the L_∞ Voronoi diagram shown in figure 8 above.

Guibas and Russel compare techniques for updating Delaunay triangulations in [16]. They compare several KDS algorithms for maintaining the structure, and find that maintaining the Delaunay triangulation through edge flips is much faster than rebuilding the triangulation from scratch, as is done in many applications. The cost of the KDS is quite expensive, and for KDS to be a reasonable technique for this, more optimization and research is necessary.

Mesh Generation

Triangulations are central to mesh generation, and therefore they will be the main focus of this section. The quality of a triangulation can be judged by its properties, and these properties are discussed. Methods of generating the mesh and triangulation are described, including Delaunay triangulation generation and mesh generation from quadtrees. After this, a method of dealing with deformable meshes through time as objects move is discussed. This section is concluded with a discussion of a current application of deformable meshes.

Quality of a Mesh

The quality of a mesh can be judged by many of the mesh's properties, including the minimum angle, the maximum angle, the aspect ratio, the longest edge length, and the total edge length are some of these properties. Many methods exist for optimizing these properties, referred to as mesh improvement, including edge flipping and smoothing. Mesh refinement can also be used to add mesh vertices to areas of large error, and derefinement/coarsening can be used to delete mesh vertices where the error is small.

Delaunay Triangulations

Delaunay triangulations are important in mesh generation because they have many important properties, one of which is that the minimum angle of all the triangles is maximized.

A triangulation can be changed into a Delaunay triangulation using edge flips and the InCircle test. The InCircle test checks for points inside the circumcircle formed by the vertices of a triangle. If no such point exists, for every set of triangles, the triangulation is Delaunay. If not, an edge flipping algorithm can be used in conjunction with this test, and the triangulation can be transformed into a Delaunay triangulation in $O(n^2)$ time [10]. Another approach to transforming a triangulation into a Delaunay triangulation involves adding additional points, called Steiner points, in order to improve the triangulation.

Quadtrees

Quadtrees can be used for mesh generation. A quadtree is formed by generating an axis-aligned square that encloses the data. The square is then split into four equal partitions, and this process is repeated in a recursive manner until each square holds a certain amount of information. Other bounds on the squares can be given to improve the splitting. After this is done, the squares are then warped to fit the bounds of the shape. A triangulation is then done one these warped squares. An example mesh generated with a quadtree based algorithm is shown below in figure 10. (Refer to [10] for a more detailed description.)

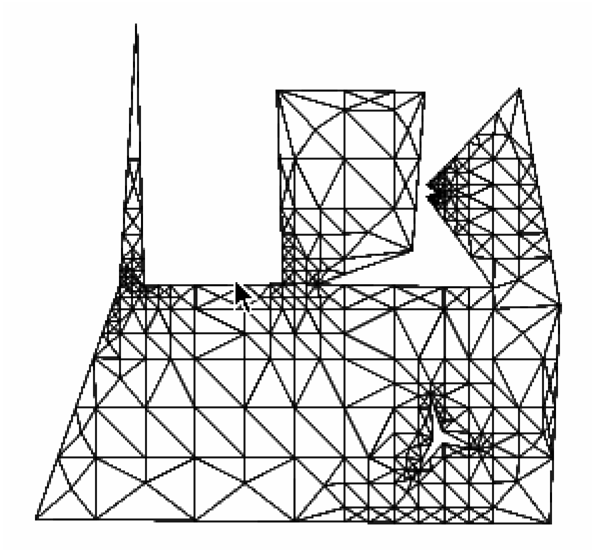

Figure 10: [10] This mesh was generated using quadtrees.

Deformable Meshes – Discontinuous Galerkin Method

According to Üngör et al [23], Discontinuous Galerkin (DG) “methods depend on the notion of domains of influence for dynamic data.” These domains of influence are found using cones, with the vertex of the cone being centered at the location of the event causing a change.

The algorithm suggested by Üngör et al in [23] builds meshes over space-time by adding a dimension for time and building a layer based solution. The algorithm takes advantage of the domains of influence by simultaneously building meshes that are outside each other's domains., hence speeding up generation time. Figure 11 below illustrates this.

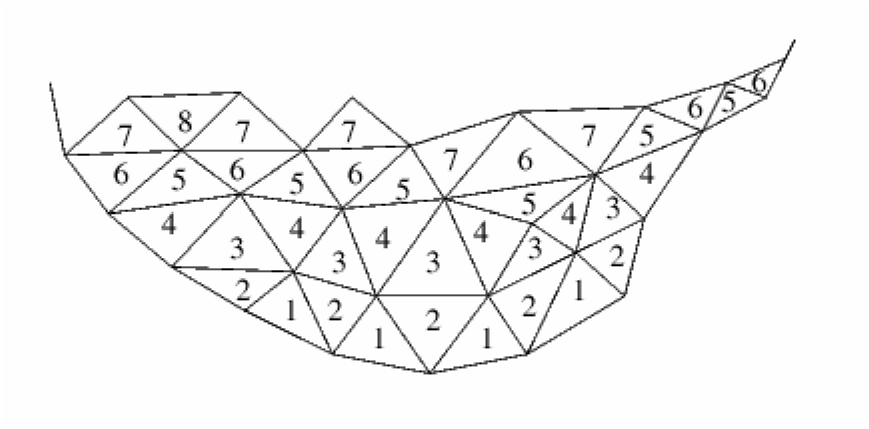

Figure 11: [23] Regions with the same number can be done in parallel. Regions with numbers higher than others are dependent upon the lower-numbered regions, and therefore those must be done first.

Animation Using Multiresolution

Deformable meshes are especially useful in animation. For this application, a dense mesh is often used for purposes of efficiency and simplicity. These meshes have much more detail than necessary though, and can make computations take longer. Methods exist for coarsening static graphs where needed, but little has been done to deal with meshes that deform through time. Kircher and Garland [20] propose a method that uses multilevel meshes "that represent the input surface at multiple levels of detail." By using these multilevel meshes to build "time-varying multiresolution hierarchies", they can get high quality approximations in every frame of an animation. Figure 12 below illustrates a multiresolution hierarchy of an image.

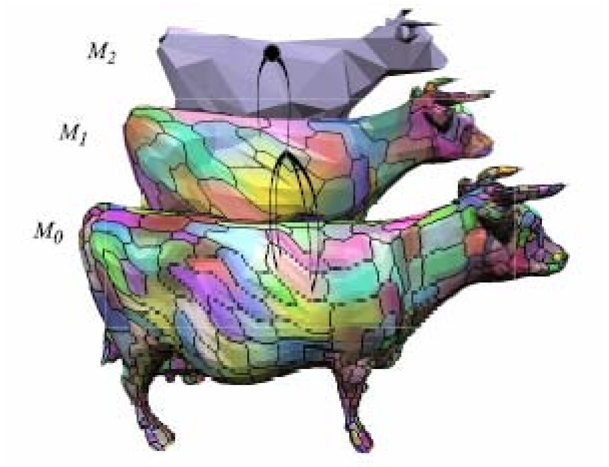

Figure 12: [20] A hierarchy of meshes at different resolutions. These are used to build high-quality image approximations over time.

Graphics

In graphics, several physically based models of motion are used. Two of these, mass spring models and finite element methods, are discussed. Geometric approaches are also used by the graphics community to model motion. An algorithm that combines two geometric approaches is discussed. Finally, a framework for combining physically based models and geometric based models is discussed.

Mass Spring Models

Mass spring models are a physically based model of motion. In this model, an object is represented as points attached to springs. When a point deforms, the surrounding points deform with it based on spring laws of motion (such as Hooke's law). These models have been widely used to model motion. An example of a part of a mass spring model is shown in figure 13 below.

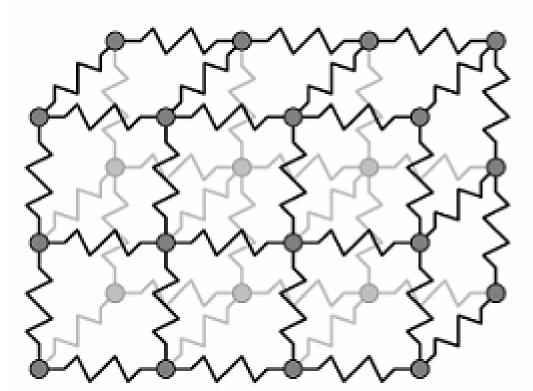

Figure 13: [15] A portion of a mass spring model, with points and attached springs.

These models have been used in facial animation. The varying layers of the skin can be modeled using different spring constants. (Other forces were added to the points to maintain skin movement properties that cannot be modeled using springs alone.) Figure 14 below shows this.

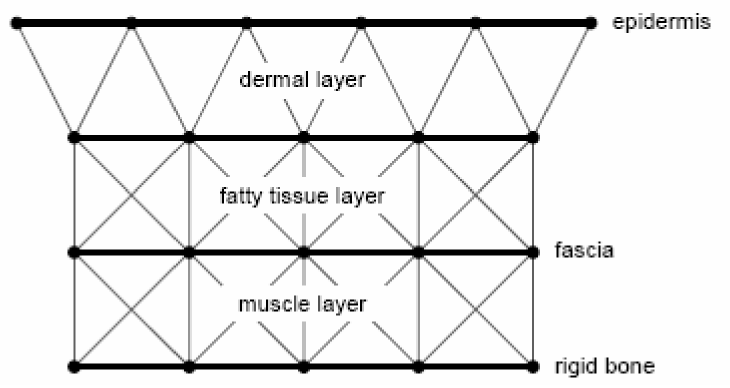

Figure 14: [15] Skin can be modeled as a mass spring system.

According to Gibson and Mirtich [15], mass spring models have the advantages of being simple and usable for interactive and real-time simulations. The use of spring constants to model nearly rigid bodies can lead to an unwanted stiffness though, and as with many approximations, the approximation is only as good as the features given to it. When the properties of a material are difficult to measure, this approximation may not be good.

Finite Element Methods

According to Gibson and Mirtich [15], "Finite Element Methods (FEMs) divide the object into a set of elements and approximate the continuous equilibrium over each element." These methods use the potential energy of the elements to model their motion. While the mass spring model is a discrete model, FEMs are continuous models. FEMs offer greater accuracy than mass spring models also, but are used little in practice because they are computationally expensive.

Geometric Based Model

According to Pauly et al in [22], geometric representations of motion in graphics fall into two categories: implicit and parametric representations. Implicit representations include level sets, radial basis functions. Parametric representations include splines, subdivisions, and triangular meshes. Both representations have their faults; so Pauly et al [22] propose a hybrid of the two. In this hybrid, a cloud of points are used to represent the shape of the object, and an implicit surface definition is also used. This hybrid allows for Boolean operations and free form deformation of the data.

Layered Approach

Chua and Neumann [12] propose a hybrid approach to deformable modeling that combines both physical and geometric elements. They suggest separating the model into three parts: a controlling mechanism, a suface deformation engine, and a mesh refinement system. The controlling mechanism deals with physically based deformations or user defined deformations. The surface deformation engine deals with the surface geometry. The last part, the mesh refinement system, obviously deals with refining deformable meshes over time.

Conclusions

This paper is an overview of current techniques and approaches being used to solve the problem of modeling motion from the viewpoints of the computational geometry, mesh generation, and graphics communities.

Modeling motion is an area still in need of research across all three of the communities discussed in this paper. As all three communities hope to find realistic methods of modeling motion in a real-time setting, the open problems reach across communities. A fair amount of work is being done to take advantage of results from other communities, especially in the graphics community.

[1] P.K. Agarwal, L. J. Guibas, H. Edelsbrunner, J. Erickson, M. Isard, S. Har-Peled, J. Hershberger, C. Jensen, L. Kavraki, P. Koehl, M. Lin, D. Manocha, D. Metaxas, B. Mitrich, D. Mount, S. Muthukrishnan, D. Pai, E. Sacks, J. Snoeyink, S. Suri, and O. Wolfson. Algorithmic Issues in Modeling Motion, ACM Computing Surveys, 24 (2002), 550–572.

[2] P. Agarwal, L. Guibas, A. Ngyuen, D. Russel, and L. Zhang, Collision detection for deforming necklaces, Computational Geometry: Theory and Applications, 28(2-3):137-163, 2004

[3] P. K. Agarwal, L. Guibas, T.M. Murali, and J.S. Vitter, Cylindrical Static and Kinetic Binary Space Partitions, Proc 13th annual ACM Symposium on Computational Geometry, pp 39-48, 1997

[4] P. K. Agarwal, J. Erickson, L. J. Guibas, Kinetic Binary Space Partitions for Intersecting Segments and Disjoint Triangles, 9th ACM-SIAM Symp. Discrete Algorithms, 1998

[5] P. Agarwal, J. Basch, L. Guibas, J. Hershberger, and Li Zhang, Deformable Free Space Tilings for Kinetic Collision Detection, 4th International Workshop on Algorithmic Foundations of Robotics, 2000.

[6] P. K. Agarwal, J. Gao, L. J. Guibas, Kinetic Medians and kd-trees, Proc. of the 10th Annual European Symposium on Algorithms (ESA'02), Lecture Notes in Computer Science 2461, 5-16, September 2002.

[7] P.K. Agarwal, J. Basch, M. de Berg, L. J. Guibas, and J. Hershberger, Lower Bounds for Kinetic Planar Subdivisions, Discrete Comput. Geom., 24 (2000), 721-733.

[8] P.K. Agarwal, L. Guibas, J. Hershberger, and E. Veach, Maintaining the Extent of a Moving Point Set, 5th Workshop on Algorithms and Data Structures (WADS`97), August 1997

[9] J. Basch, Leonidas J. Guibas, and J. Hershberger. Data structures for mobile data. In Proc. 8th ACM-SIAM Sympos. Discrete Algorithms, pages 747-756, 1997.

[10] M. Bern and P. Plassman, "Mesh Generation", Handbook of Computational Geometry, Jörg-Rudiger Sack and Jorge Urrutia, ed., Elsevier, 2000, pp. 291–332

[11] L. P. Chew, Near-quadratic bounds for the L1 Voronoi diagram of moving points, Comput. Geom. Theory Appl., 7:73-80, 1997.

[12] Clint Chua and Ulrich Neumann,A Modular Approach to Deformable ing and Animation, IEEE Computer Animation 2001, pp.184-191, Seoul, Korea, November 2001.

[13] H. Edelsbrunner. Geometry and Topology for Mesh Generation. Cambridge Univ. Press, England, 2001.

[14] J.G. Erickson, D. Guoy, J.M. Sullivan, and A.Üngör, Building space-time meshes over arbitrary spatial domains, Proc. 11th Int. Meshing Roundtable, Sandia Nat. Lab., 2002

[15] Gibson and Mirtich, "A Survey of Deformable Modeling in Computer Graphics", MERL Technical Report, TR97-19, 1997

[16] L. Guibas, D. Russel, An Empirical Comparison of Techniques for Updating Delaunay Triangulations, SoCG 2004

[17] L. J. Guibas. Kinetic data structures --- a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, Workshop on Algorithmic Foundations of Robotics, pages 191--209. 1998.

[18] L. Guibas. Modeling Motion. In Handbook of Discrete and Computational Geometry, J. Goodman and J. O'Rourke, Eds, 2nd Ed., Chapman and Hall/CRC, 2004, pp. 1117-1134.

[19] L. J. Guibas, A. Nguyen, and L. Zhang, Zonotopes as Bounding Volume, Symposium on Discrete Algorithms, 2003, 803-812

[20] S. Kircher and M. Garland. Progressive Multiresolution Meshes for Deforming Surfaces. ACM/Eurographics Symposium on Computer Animation, pp. 191–200, 2005.

[21] D. Kirkpatrick, J. Snoeyink and B. Speckmann, Kinetic Collision Detection for Simple Polygons, International Journal of Computational Geometry and Applications, 12(1&2):3-27, 2002.

[22] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross, Shape modeling with point-sampled geometry, in Proceedings of ACM SIGGRAPH 2003, Vol 22, Issue 3, July 2003, pp 641-650

[23] A. Üngör, A. Sheffer, R.B. Haber, and S. Teng, Layer based solutions for constrained space-time meshing, Applied Numerical Mathematics 46(3–4):425–443, September 2003

[24] http://valis.cs.uiuc.edu/~sariel/research/CG/applets/bounding_rectangle/main.html

[25] http://www.cs.umd.edu/class/fall2005/cmsc754/Lects/741-0930.pdf

[26] http://www.decatur.de/personal/zono/applet.html

[27] http://www.rolemaker.dk/nonRoleMaker/uni/algogem/kdtree.htm