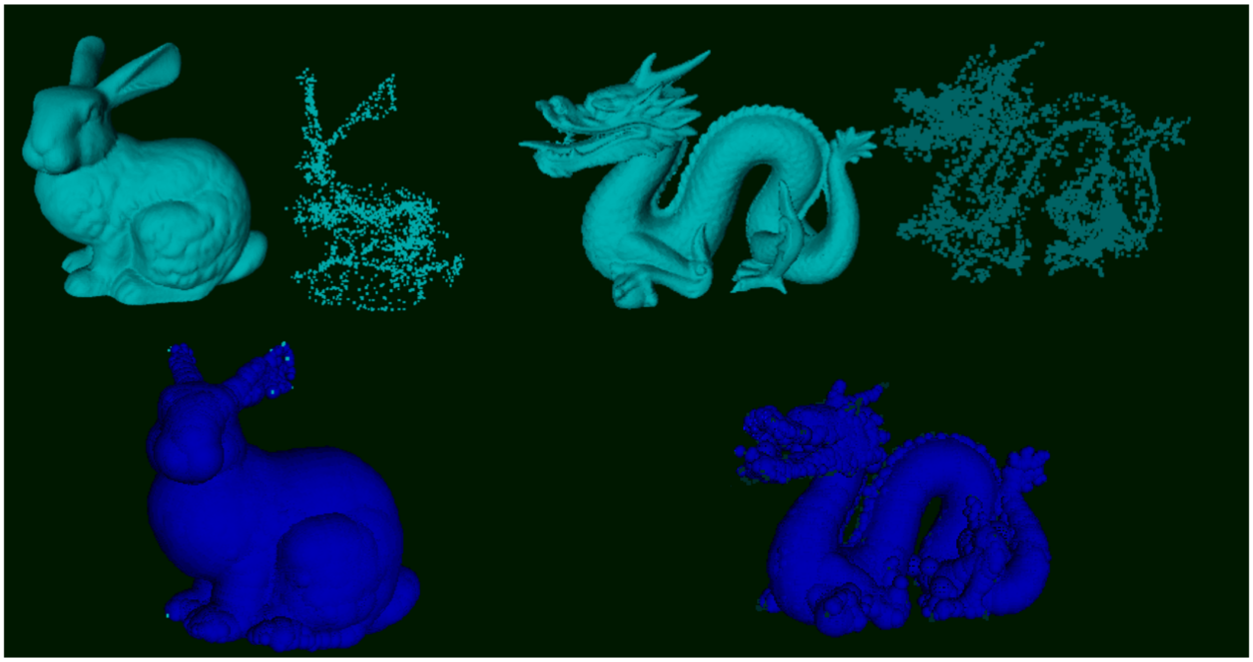
Medial Axis Transform

Rohan Sawhney, Michael Reed, Keenan Crane



Abstract

In this undergraduate research project report, a fast sampling based algorithm and surface reconstruction strategy is presented to compute the medial transform of 2D and 3D geometrical shapes. In addition, a method to isolate the junction points of the 2D medial axis is provided and iterative approaches to surface sampling in 3D are discussed.

Introduction

The medial axis of an object is the set of all points having more than one closest point on the object's boundary. Mathematically, it is defined as the locus of all centers of circles inside the 2D polygon (or spheres inside a 3D object) that are tangent to the boundary in two or more places. Since its introduction by Blum [1] as a means to describe shapes in biology and medicine, the medial axis has become an important tool in computational geometry and geometric modeling.

Shape skeletonization (i.e., medial axis extraction) is a powerful technique in many visual computing applications, such as pattern recognition, object segmentation, registration, and animation [2][3]. This is because as a local lower-dimensional characterization of a solid, the medial axis provides a compact representation of solid models that preserves topological properties. The medial axis also has several other unique advantages in modeling geometric objects. First, it provides localization of features such as anatomical landmarks (which are extremely valuable in bio-medical applications). Second, it separates thickness information (e.g., radius of medial axis) from orientational and topological information, i.e., shape features can be subdivided into radial, orientational and location information in order to facilitate statistical analysis. Third, it allows for shape differences between objects to be quantified in a more intuitive and accurate way. Fourth, as it is often more expeditious to capture only the coarse-scale changes of acquired models, a simplified or pruned medial axis serves as a more stable and robust representation of noisy datasets.

In addition to skeletonization, surface reconstruction is becoming increasingly important in geometric modeling for generating surfaces from data points captured from real objects, often by laser range scanners but also by hand held digitizers, computer vision techniques, edge detection from medical images, or other technologies. Industrial applications include reverse engineering, product design and the construction of personalized medical appliances [2][3]. The medial axis together with the associated radius function of the maximally inscribed discs (or spheres), from now referred to as maximal balls, is called the medial axis transform (MAT). The MAT serves as a complete shape descriptor, meaning that it can be used to reconstruct the shape of the original domain.

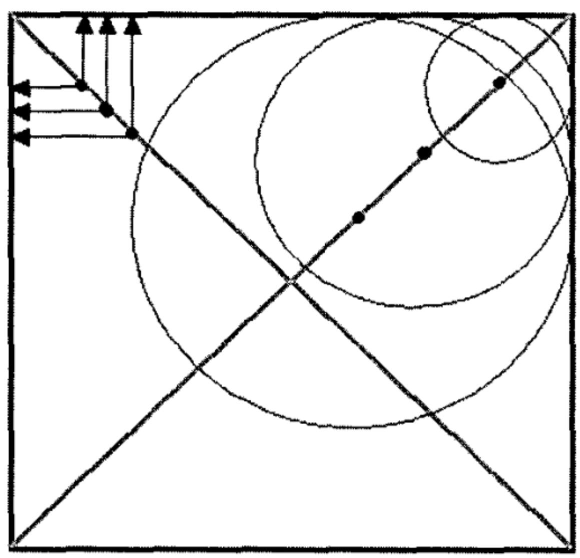


Figure 1: 2D illustration of Medial Axis

However, to date work on the MAT has been limited by the difficulty inherent in developing accurate and efficient algorithms to compute it, especially for 3-D objects. The primary drawback of the medial axis is that it is very sensitive to minor perturbations of the object's boundary, such as that caused by discretization, segmentation errors, image noise and so forth. The goal of most medial axis pruning techniques thus is to remove the branches associated with these artifacts, resulting typically in much cleaner and more usable medial axis as mentioned previously. The de-noised axis can be used to reconstruct a smoother version of the original object [10].

Relevant Work

The usefulness of the MAT has inspired many methods for its computation. In most cases, the algorithm operates on a discrete approximation of the object, such as a set of sample boundary points, and outputs a polygonal approximation of the medial axis. At a broad level, algorithms for medial-axis computation can be classified into the following categories: thinning algorithms, distance field algorithms, algebraic tracing methods, and surface-sampling approaches. These categories differ in terms of the underlying representations used for the medial axis as well as how they compute it.

Thinning Algorithms

Thinning algorithms use a voxel-based representation of the initial figure, and perform erosion operations to arrive at a set of voxels approximating the medial axis. Careful erosion of the object's voxels is performed layer by layer while preserving the object's topological and geometrical properties, i.e, a voxel is removed if and only if its removal does not induce a local change in topology (e.g. breaking the object in two parts, creating a hole or cavity). These methods are significant in the areas of image processing and pattern recognition, since the input data is represented as a discrete grid. However, being fundamentally discrete processes, thinning methods require fully segmented, compact, and connected objects and have difficulties dealing with partial data. They are also sensitive to Euclidean transformations of the data.

Many algorithms based on partial differential equations of front propagation have also been proposed. Du et al. [4] employ a diffusion-based PDE to allow 3D objects to progressively propagate their boundaries inward using a finite differences and approximate simplified skeletons with user interactions. The distance information from skeletal points to the boundaries are recorded for reconstruction and deformation purposes. However, in addition to being sensitive to the value of time intervals for the diffusion process, such algorithms are also slow for large datasets as they require checking for collisions between sampling points and faces.

Distance Field Algorithms

Because the skeletal or medial surface points usually coincides with the singularities of a Euclidean distance function to the boundary, distance functions can be employed for medial axis extraction. The approaches based on distance functions construct distance field transformation of an object and extract the medial axis based on the distance field. Danielsson [5] uses a scanning approach to create an image in which each pixel contains the Euclidean distance to the nearest pixel on the boundary of the figure being analyzed. Moreover, the resulting distance map can be analyzed for local directional maxima to get an approximation of the medial axis. Such methods however have difficulty in ensuring homotopy with original objects.

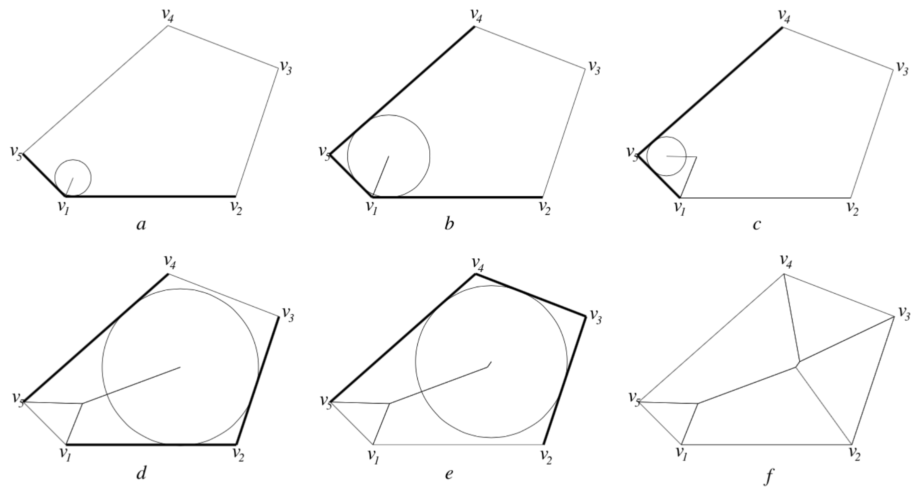


Figure 2: MAT computation for a convex polygon using path tracing, Arinyo et al.

Algebraic Methods

There is a family of methods that rely fundamentally on the fact that the algebraic form is explicitly known for each surface patch of the medial axis of a polyhedron. Most algorithms that represent the medial axis symbolically use a tracing approach. Starting from a junction point on the medial axis, a seam emanating from the junction is followed. The seam terminates at another junction and the process is applied recursively. Using such an approach, Arinyo et al. [6] give an algorithm for computing the medial axis of a planar region bounded by piecewise C2 curves. Culver et al. [7] have demonstrated tracing algorithms for polyhedra, all using different methods to find the endpoints of the seam curves. Culver et al. represent the medial axis exactly by means of systems of algebraic equations manipulated using rational arithmetic. Their method computes an exact representation of the medial axis provided there are no degeneracies (such as more than four seams intersecting at a point). All of the methods in this family have been applied to polyhedra composed of only a few hundred faces. It is not clear whether they can be applied to complex models composed of tens or hundreds of thousands of faces. Either their running time is more than O(n^2), where n is the number of faces, or these algorithms are susceptible to accuracy and robustness problems.

Surface Sampling Approaches

Surface sampling methods represent the initial figure as a dense cloud of sample points presumed to be on or near the boundary. The medial axis of the figure is approximated by a subset of the Voronoi diagram of the point cloud. Different algorithms based on this approach use different methods for selecting the desired subset of the Voronoi diagram. Many such variations have been proposed. Boissonnat [8] classified certain triangles of the Delaunay tetrahedralization of the point cloud as interior to the model; the Voronoi vertices dual to those tetrahedra approximate the medial axis. Using a similar approach, Amenta et al. [9] construct an approximate, simplified medial axis which they use as a stage in a surface reconstruction from the original point cloud, a common application for this approach. These algorithms have been applied to models composed of tens of thousands of points.

One of the issues when applying these algorithms to polyhedral models is in generating appropriate point samples on the boundary to ensure a tight approximation of the medial axis. In general, the worst-case running time of these algorithms can be O(n^2), where n is the number of point samples. The main problem though with such an approach is that unlike in 2D, the Voronoi vertices (circumcentres of the tetrahedra) in 3D do not converge to the medial axis as the sampling density approaches infinity [Amenta et al. 2001b]. Therefore, regardless of sampling density, there are many tetrahedra that are not even close to the medial axis that are being used for enforcing topological constraints. This can often hinder the regularization process.

Experiments and Observations

Figure 3 outlines a fast sampling based algorithm to compute the medial axis of 2D and 3D objects. For every randomly chosen point P on a mesh, a point Q that lies at intersection of the mesh and the ray along the inward normal to the face of P is computed. P and Q are then used to compute the maximal ball inside the mesh whose center lies on the segment PQ.

```
void computeMedialAxis ( int numSamples )

        for ( int i = 0; i < numSamples; i++ )

                Face f = mesh.getRandomFace ()
                Point basePoint = f.getRandomPoint ()

                Ray r ( basePoint, f.getInwardNormal() )
                Point intersectionPoint = mesh.getIntersection ( r )

                Point medialPoint = computeMedialPoint ( basePoint, intersectionPoint )

                medialAxisPoints.append ( medialPoint )
```

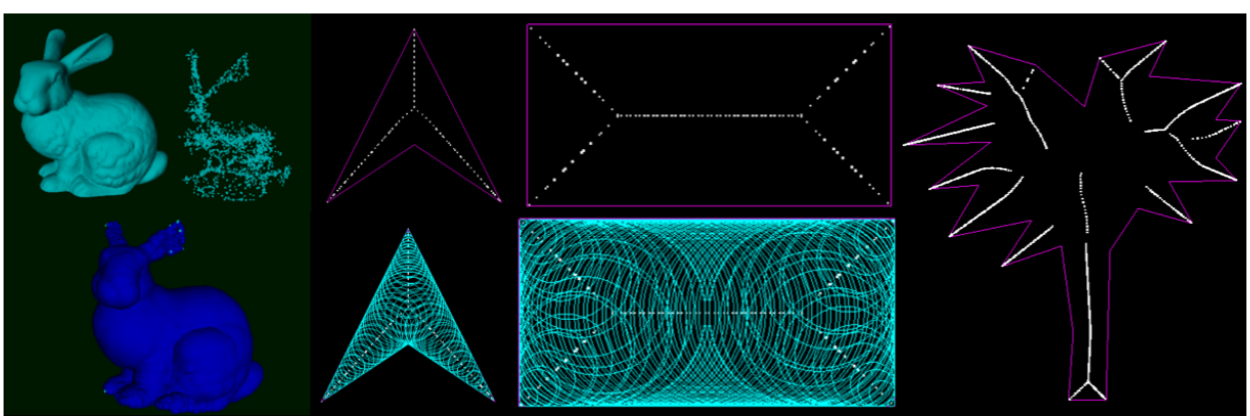
Figure 3

A binary search based approach as listed in Figure 4 is used to determine the center and radius of the maximal ball. Given a ball fixed at the original value of point P in **computeMedialPoint**, it is shrunk if it is not entirely contained inside the mesh (Q is set to the midPoint of P and Q) and grown otherwise (P is set to the midPoint of P and Q) till the distance between P and Q becomes negligible. The point P and Q converge to a point on the medial axis. The radius of the maximal ball centered at this point is kept track of for surface reconstruction.

```
Point computeMedialPoint ( Point P, Point Q )

        Point midPoint = ( P + Q ) / 2

        Point basePoint = P
        double radius =  II midPoint - P II

        while ( II Q - P II > EPSILON )

                if ( mesh.containsBall ( midPoint, radius) )
                        P = midPoint
                else
                        Q = midPoint

                midPoint = ( P + Q ) / 2

                radius =  II midPoint - basePoint II


        midPoint.setRadius ( radius )
        return midPoint
```

Figure 4

The complexity of the algorithm is O (N F log(IIQ - PII / EPSILON)), where N is the number of samples and F is the number of faces in the mesh. However, if the mesh functions **getIntersection** and **containsBall** are implemented with an acceleration structure such as the Bounding Volume Hierarchy (BVH), the runtime cost reduces to O(N log (F) log (IIQ - PII / EPSILON)). The medial axis of various 2D and 3D shapes and their reconstructions are illustrated below.



Samples / 3D Model	Stanford Bunny (626994 Triangles)	Stanford Dragon (900000 Triangles)	Stanford Buddha (900000 Triangles)
500	30 - 35	10 - 12	12 - 15
1000	68 - 73	21 - 23	27 - 30
5000	370 - 400	100 - 110	130 - 150

Timings (in seconds) with BVH

Isolating Junction Points of the Medial Axis in 2D

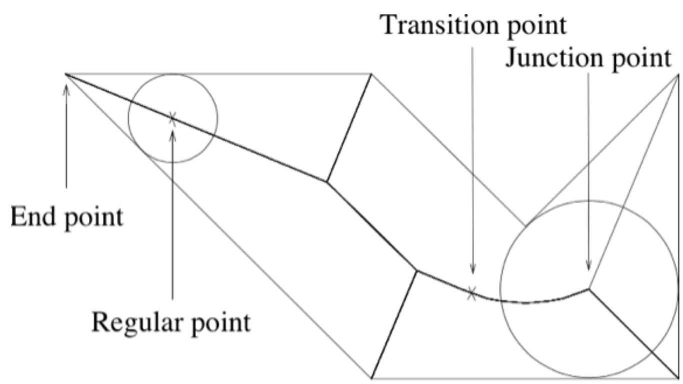


Figure 5: Taxonomy of medial axis points.
Arinyo et al. [6]

In Figure 5, Arinyo et al. [6] note that the medial discs that define the medial axis are tangent only to edges and concave vertices in the boundary. They refer to the edges and concave vertices in the domain boundary as the *active boundary elements* and *governors* as the subset of *active boundary elements* to which a maximal inscribed disc centered on a medial axis point is tangent. Points on the medial axis are classified based on the number of *governors* that define them.

Junction point: Point where the maximal disc is tangent to 3 or more *governors*.

End point: Convex vertex of the polygon where the radius of the medial axis point is zero. Also the point where the medial axis intersects with the boundary.

Regular point: Point where the maximal disc is tangent to 2 *governors.* Points in the medial axis that are neither *junction* points nor end points are *regular* points.

Transition point: Medial axis *regular* point where one of both *governors* changes.

Figure 5 highlights that the junction and end points alone provide a reduced representation of the medial axis itself. These points provide the least amount of information required to completely specify the 2D boundary. The radius of all the maximal balls between adjacent junction - junction or junction - end paris can be determined based on the radius of the points in such a pair.

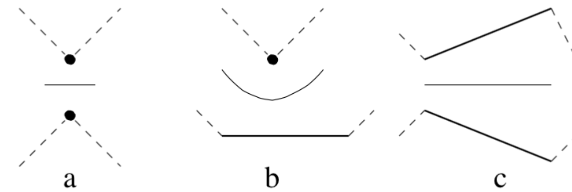


Figure 6: path s of medial paths based on governors

Arinyo et al. [6]

Additionally, Figure 6 shows that the paths between junction - junction or junction - end pairs can be either linear or parabolic. Given a concave vertex and an edge as in Figure 6 b), it can be demonstrated that the medial path is indeed parabolic in the vicinity of such a configuration by setting the distance from X to A equal to the distance from X to B in Figure 7 and solving from y in terms of x.

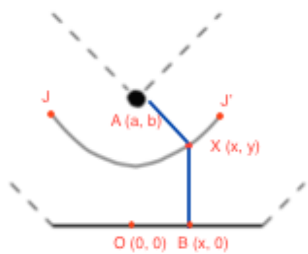


Figure 7: parabolic shape of medial path

With any three points on the medial path in Figure 7, the parabolic shape of the medial path can be determined by solving three linear equations to compute the coefficients of the parabolic function. Assuming two of these three points are junction points J and J', the third point can be found by sampling along the projection of the line JJ' on the boundary and computing the medial point as in Figure 4.

Figure 8 modifies **computeMedialAxis** to isolate the junction and end points of the medial axis. **computeMedialJunctionPoints** assumes that the boundary edges are in either clock wise or anti clockwise order. It then samples a percentage of the total sample points weighted by edge lengths on each edge. As seen in Figure 1, **computeEffectiveNormal** sums the normals of all *governors* for a medialPoint and assigns the net normal vector to it (Figure 10 b). If the effective normals of two adjacent medial points do not point in the same direction, then the junction point known to exist between these two medial points is computed. Otherwise, the two medial points lie on the same linear medial path. Note in the case where the effective normal sums to zero, the normal of any one *governor* is assigned to the medial point. **computeMedialJunctionPoints** also appends the convex vertices of the boundary to

```
void computeMedialJunctionPoints ()

    for each edge e in mesh.edges
        for ( int i = 0; i < samplesForEdge ( e ); i++ )

            double t = ( i+1 ) / e.length
            Point basePoint  = e.length * t

            Ray r ( basePoint, e.getInwardNormal () )
            Point intersectionPoint = mesh.getIntersection ( r )

            Point medialPoint = computeMedialPoint ( basePoint, intersectionPoint )
            computeEffectiveNormal ( medialPoint )

            medialAxisPoints.append ( medialPoint )

            if ( ( int s = medialAxisPoints.size ) > 1 )

                Point p1 = medialAxisPoints [s-2]
                Point p2 = medialAxisPoints [s-1]

                double n = p1.effectiveNormal.cross ( p2.effectiveNormal )

                if ( || n || > 0 )

                    Point junctionPoint = computeJunctionPoint ( p1, p2 )

                    medialJunctionPoints.append ( junctionPoint )


        if ( e.cross ( e.next ) > 0 )

            medialJunctionPoints.append ( e.secondPoint )
```

Figure 8

```
Point computeJunctionPoint ( Point P, Point Q )

    Point midPoint = ( P + Q ) / 2
    Point junctionPoint

    while ( || P - Q || > EPSILON )

        Point basePoint = mesh.getClosestPoint ( midPoint )

        Ray r ( basePoint, basePoint.edge.getInwardNormal () )
        Point intersectionPoint = mesh.getIntersection ( r )

        junctionPoint = computeMedialPoint ( basePoint, intersectionPoint )
        computeEffectiveNormal ( junctionPoint )

        if ( junctionPoint.effectiveNormal.cross ( P.effectiveNormal ) == 0 )
            P = midPoint

        else if ( junctionPoint.effectiveNormal.cross ( Q.effectiveNormal ) == 0 )
            Q = midPoint

        midPoint = ( P + Q ) / 2

    return midPoint
```

Figure 9

its list of junction points (Figure 10 c).

computeJunctionPoint in Figure 9 uses a binary search approach to compute the junction point between two regular medial points. It picks the midpoint of the line segment PQ and finds its closest *governor* (by using BVH). It then checks if the effective normals of the newly computed medial point returned from **computeMedialPoint** lies in the same direction as those of P or Q. Based on the result, it updates either P or Q to the midpoint. The point P and Q converge to a junction point of the medial axis.

The algorithm in Figure 8 and 9 reduces the ordering on the boundary edges to the list of junction points, thereby providing a way to connect adjacent junction points (Figure 10 d). It can be extended to account for parabolic medial paths by detecting concave vertices along the boundary.

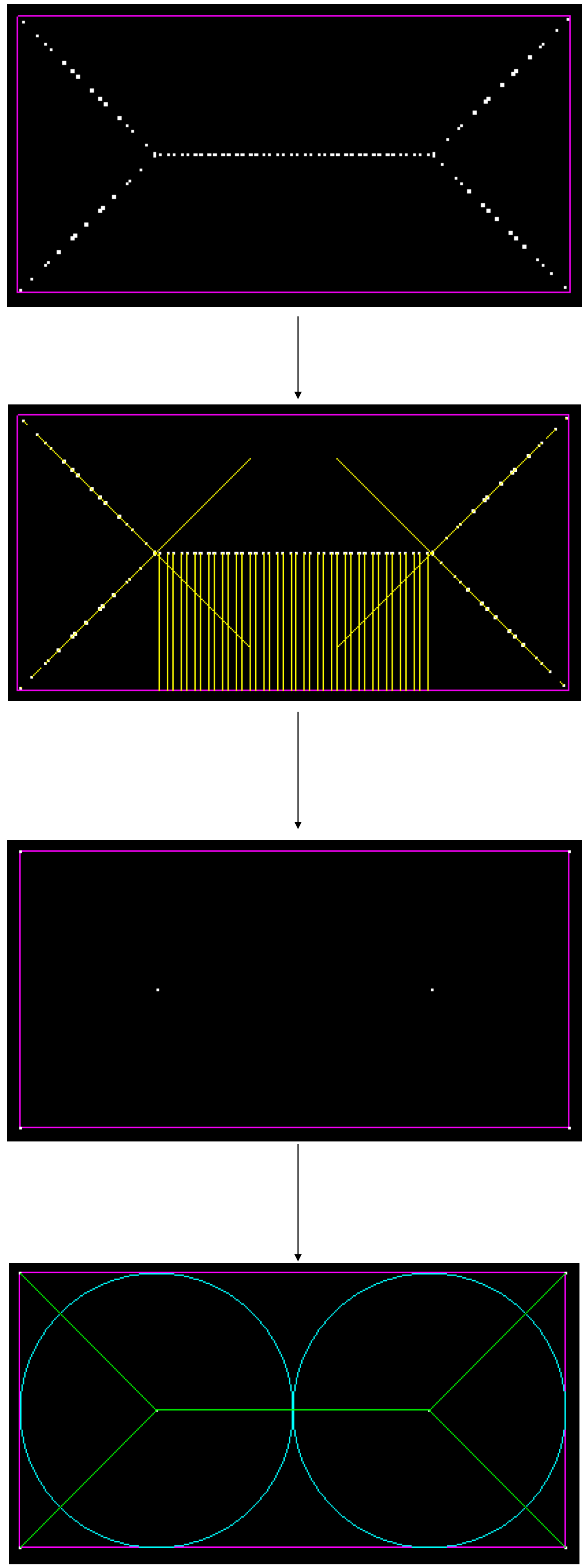
Iterative Sampling Strategies

Establishing connectivity between medial axis points in 3D is difficult because there does not exist any obvious scheme to sample the surface of a 3D mesh in order as in 2D. Therefore, sampling strategies need to be devised to avoid oversampling the boundary. Two relatively simple iterative strategies are presented to do so.

1) Closest neighbor: For each medial point P, the medial point Q closest to it is found such that the point does not already have a neighbor and the maximal balls of these two points do not overlap. These conditions provide a linear ordering on the medial points. The line segment PQ is then inserted into a max heap sorted by the length of line segments. Points are sampled recursively along the projection of the segment PQ (returned from the heap) on the boundary to compute a new medial point M until the maximal balls of P and M or Q and M do not overlap.

2) Closest medial point to mesh face: Having identified a set of governors (that are faces) for a set of medial axis points, the distance for each of the remaining faces on the boundary to the medial point closest to them is determined. Face medial point pairs are inserted in a max heap sorted by the distance between the face and the medial point closest to it. The faces of the entries returned from the heap are then sampled. Heap entries containing faces whose distance to the newly computed medial point is smaller than the previously computed distance are discarded.

Both these strategies have one major drawback: they do not sample in regions of the boundary that lack initial samples.



a) Medial Axis Points from Figure 8

b) Effective Normals

c) Junction and End Points

d) Connected Junction and End Points with radius function

Figure 10

Future Directions

1) Through an extension of the scheme provided in the section **Isolating Junction Points of the Medial Axis in 2D**, the junction and end points of 3D shapes can also be determined. In 3D, the governors of medial points would consist of various possible combinations of faces, edges and concave points resulting in planar, parabolic and hyperbolic medial paths [6]. However, imposing an ordering and thus connecting junction and end points is a difficult problem as there does not exist any notion of an ordered traversal of the surface of a 3D shape. Therefore, further work needs to be done on establishing connectivity between junction - junction and junction - end pairs of the medial axis in 3D.

2) Sampling: Smarter sampling strategies for both 2D and 3D shapes need to be developed to avoid oversampling the boundary.

3) Simplification: The primary drawback of the medial axis is that it is very sensitive to minor perturbations of the object's boundary, such as that caused by discretization, segmentation errors, image noise and so forth. To further extend algorithm developed, the next logical step would be to develop pruning techniques to remove noisy branches of the resulting medial axis.

4) Implementing Papers: Arinyo et al. [6] provide a tracing algorithm in 2D similar to the one outlined in this report and claim that it is possible to extend their algorithm to 3D. It would be a worthwhile exercise to implement this paper to develop further insights into medial axis computations. Implementing and improving on Du et al. [4]s PDE based approach is another direction worth exploring given the work on adaptive simulation being done by the Columbia Computer Graphics Group.

Acknowledgements

I would like to thank Professor Michael Reed and Dr. Keenan Crane for their support and guidance throughout the course of this undergraduate project. I would also like to thank Professor Eitan Grinspun for giving me the opportunity to undertake this project.

References

[1] H Blum. *A transformation for extracting new descriptors of shape*

[2] http://en.wikipedia.org/wiki/Medial_axis

[3] Dominique Attali, Jean-Daniel Boissonnat, and Herbert Edelsbrunner. *Stability and Computation of Medial Axes — a State-of-the-Art Report*

[4] Haixia Du and Hong Qin. *Medial Axis Extraction and Shape Manipulation of Solid Objects Using Parabolic PDEs*

[5] Danielsson, P. E. 1980. *Euclidean Distance Mapping*

[6] R. Joan-Arinyo, L. Perez, J. Vilaplana. *Computing the Medial Axis Transform of Polygon Domains By Tracing Paths.*

[7] Tim Culver, John Keyser, Dinesh Manocha. *Accurate Computation of the Medial Axis of a Polyhedron*

[8] Dominique Attali, Jean-Daniel Boissonnat. *Approximation of the Medial Axis*

[9] Nina Amenta, Sunghee Choi, Ravi Krishna Kolluri. *The Power Crust, Unions of Balls, and the Medial Axis Transform*

[10] Roger Tam and Wolfgang Heidrich. *Shape Simplification Based on the Medial Axis Transform*