Geometric Rounding
Notes for Mini-Course

Kurt Mehlhorn

April 2002

Contents

1 Introduction

Geometric objects are a combination of combinatorial data (a planar graph, a polygon, a face lattice of a polyhedron, ...) and numerical data (the coordinates of the vertices, equations for the faces, ...). There are several reasons for rounding or simplifying geometric objects. I give some.

- You have a detailed map of Germany of scale 1:10000 and have to produce a map of scale 1:100000. Clearly, you have to suppress some detail, e.g., simplify the shape of rivers or borders. However, you may also want to maintain some consistency, e.g., you may keep cities on the correct side of borders or you may want to maintain approximate areas of regions.

- You have computed a geometric object and the representation of the object requires high precision numerical data. You now want to reduce the precision of the numerical data. I can see two situations where this would be required.
  - The object is to be moved to a system which cannot handle the high precision data. This situation arises frequently for users of CGAL and LEDA. Our systems compute objects with exact numerical data. When the object is to be moved to an external system, the numerical data has to be rounded to double precision floating point arithmetic. If only the numerical data is rounded, inconsistencies might result. The combinatorial data must also be rounded.
  - The object needs to undergo further geometric processing. In principle, the further processing can also be done by exact algorithms. However, the growth of numerical precision is prohibitive.

Let me give an example to make this clearer. Assume that we have points in the plane represented by homogeneous coordinates (X, Y, W) all of about B bits. The vertices of the Voronoi diagram have representations with about $3B$ bit coordinates. The vertices of the Voronoi diagram of the vertices will have $9B$ bit coordinates. The following program shows this effect.

I first construct a set of points and then construct their Delaunay triangulation and their Voronoi diagram. I take the vertices of the Voronoi diagram as the input points for the next rounds. I do so in two ways

- as said
- I first round the points to floating point rational coordinates and then back to rational. This keeps the coordinates small (less than 1000 bits)

⟨*voronoi_test.c*⟩≡

```
#include <LEDA/window.h>

#include <LEDA/graph.h>
#include <LEDA/map.h>
#include <LEDA/rat_kernel.h>
#include <LEDA/geo_alg.h>

using namespace leda;

point rat_point::to_point() const
{ rational x = xcoord(), y = ycoord();

  return point(x.to_double(),y.to_double());
}

int main()
{ window W; W.display();
  W.set_node_width(2); W.set_line_width(2);
  point p;
```

```
  list<rat_point> L, LR;
  GRAPH<rat_point,int> G, GR;
  rat_point::use_filter = false;
  while ( W >> p ) { L.append(rat_point(p)); LR.append(rat_point(p)); W << p; }
  int i = 0; float T = used_time();
  while (true)
    { G.clear(); T = used_time();
     DELAUNAY_TRIANG(LR,GR);
     cout << "\n\nround " << i++ << " took " << used_time(T) << " seconds for the rounded points. I show the diagram in blue. ";
     W.clear();
      node v; edge e; rat_point p;
      forall_nodes(v,GR) W.draw_node(GR[v].to_point(),blue);
      forall_edges(e,GR) W.draw_segment(GR[source(e)].to_point(),
                                        GR[target(e)].to_point(),blue);
      DELAUNAY_TRIANG(L,G);
cout << "\nIt took " << used_time(T) << " seconds for the unrounded points. I show the diagram in black. You will have to wait a while for the next prompt since I have to prepare for the next round. ";
      forall_nodes(v,G) W.draw_node(G[v].to_point());
      forall_edges(e,G) W.draw_segment(G[source(e)].to_point(),
                                       G[target(e)].to_point());
      GRAPH<rat_circle,rat_point> VD, VDR;
      DELAUNAY_TO_VORONOI(G,VD);DELAUNAY_TO_VORONOI(GR,VDR);
      L.clear(); LR.clear();
      int maxlogW = 1;
      forall_nodes(v,VD)
      { if (VD.outdeg(v) < 2) continue;
        p = VD[v].center();
        p.normalize();
        maxlogW = leda_max(maxlogW,log(leda_max(integer(2),abs(p.W()))));
        L.append(p);
      }
      cout << "\nlog of maximal homogenizing coordinate = " << maxlogW << "\n\n";
      forall_nodes(v,VDR)
      { if (VDR.outdeg(v) < 2) continue;
        p = VDR[v].center();
        p.normalize();
        LR.append(rat_point(p.to_point()));
      }
      read_int("type any int to proceed   ");
    }
}
```

Here is part of the output.

```
round 5 took 0.02 seconds for the rounded points. I show the diagram in blue.
It took 0.93 seconds for the unrounded points. I show the diagram in black. You
will have to wait a while for the next prompt since I have to
prepare for the next round.

log of maximal homogenizing coordinate = 5840

type any int to proceed   1


round 6 took 0.0599999 seconds for the rounded points. I show the diagram in blue.
It took 12.55 seconds for the unrounded points. I show the diagram in black.
You will have to wait a while for the next prompt since I
have to prepare for the next round.

log of maximal homogenizing coordinate = 22295

type any int to proceed   1

round 7 took 0.110001 seconds for the rounded points. I show the diagram in blue.
It took 132.39 seconds for the unrounded points. I show the diagram in black.
You will have to wait a while for the next prompt since I have to prepare for the
next round.

log of maximal homogenizing coordinate = 68300
```

To my surprise there was no difference in the visual output between the diagram obtained by rounding after every step.

Exercise 1 *Identify an application which requires geometric rounding and work out the requirements on the rounding process.*

2 Topologically Consistent Rounding of Subdivisions

We have a set of line segments in the plane. The segments may intersect at their endpoints, but are disjoint otherwise. Alternatively, we start with a set of possibly intersecting segments and put a vertex at every endpoint and intersection point. In this way, we obtain an straight-line embedded planar graph.

The goal is to round the coordinates of the vertices to the integer grid (coordinate x is mapped to integer i with $i - 1/2 \leq x < i + 1/2$.) whilst maintaining the topology of the subdivision as much as possible. What does it mean to preserve the topology. We replace each line segment by a polygonal line. These polygonal lines may run on top of each other, but proper crossings must correspond to crossings in the original input. Also the polygonal lines must not move across vertices. A formal definition of what constitutes a legal rounding is non-trivial. A nice definition can be found in a paper by Milenkovic [Mil00], see Section 2.3.

2.1 Green - Yao Rounding

Green and Yao [GY86] were the first authors in the computational geometry community to study geometric rounding.

Their suggestion was as follows. Consider the plane with nails put at all lattice points. View edges as rubber bands. Move the vertices of the subdivision to their nearest lattice point and keep the rubber bands tight. In this way each edge is replaced by a polygonal line. It is clear that the process is a geometric rounding. Prove it (you cannot simply argue that the physical process is a deformation as edges might meet before time 1; you may argue along the following lines. Replace each edge by a rubber band of non-zero width and view the edge running at the centerline of the tube. As time progresses from zero to one, move the endpoints of the edges to grid points and simultaneously shrink the widths of the rubber bands to zero.

Green/Yao rounding may produce a large number of new vertices along the polygonal chains. It has the advantage that it moves edges only by a distance of at most 1/2. Later papers tried to reduce the number of additional vertices without introducing larger geometric deviations.

2.2 Snap Rounding

Hobby [Hob99] and Greene (unpublished work) introduced snap rounding. Greene's work is unpublished and Hobby's work appeared only ages after it was done. Later papers on snap rounding are [GGHT97, GM98, HP].

The idea behind snap rounding is as follows. Partition the plane into pixels centered at integer lattice points. A pixel consists of all points that are rounded to its center point.

A pixel is called *hot* if it contains a vertex of G. The embedding is rounded as follows. Each edge of G is split at every hot pixel it passes through, i.e., for every hot pixel crossed by an edge we put a vertex on the edge and inside the pixel. Then we move all vertices (new and old) to the center points of the pixels containing them. In this way each edge is replaced by a polygonal path. The number of vertices on an edge is equal to the number of hot pixels crossed by the edge.

Guibas and Marimont give a very nice analysis of the properties of snap rounding. They refer to the original segments as ursegments and to the resulting polygonal chains as polysegments. An ursegment s is rounded into polysegment σ.

Lemma 1 *After snap rounding, the polysegment σ corresponding to an ursegment s is contained within the Minkowski sum of s with a pixel (= unit square) centered at the* *origin.*

Proof: Consider snap rounding. We add additional vertices with hot pixels and them round the vertices. The rounded vertices are within the Minkowski sum of their original and a pixel. The Minkowski sum of s and a pixel is convex. Thus the rounded version of every edge segment is within the Minkowski sum of its original. ∎

We next show that snap rounding does not introduce crossing. To this effect, we view snap rounding as a continuous deformation. Consider a pixel. All vertices within the pixel are rounded into the pixel center. Edge segments move correspondingly.

We do the following. We introduce additional vertices on the boundaries of hot pixels. In this way we break ursegments into internal edges (running within hot pixels) and external edges (running between hot pixels). We now perform the rounding as follows.

We first shrink pixels to vertical line segments of length one and then to a point. In the first step artificial vertices only move horizontally and in the second step they move vertically. At the end internal edges are shrunken to length zero and the external edges form the polysegments. We do the shrinking process continuously. For example, in phase 1, we move artificial vertices with speed proportional to their distance from the central vertical axis of the pixel. In this way all artificial vertices will arrive at the central axis at the same time.

Assume now that we introduce additional crossings. Then it must happen that a vertex moves across a line segment during shrinking. This clearly does not happen inside hot pixels. So it can only happen between an external segment and an artificial vertex. Thus is must happen that an external segment enters a hot pixel during shrinking. Thus an external segment must overtake a corner of a hot pixel during shrinking. However, corners of pixels move with maximal speed v_0 during shrinking. Other artificial vertices move with speed at most v_0. Internal points of external segments move with a speed which is a convex combination of the speed of the endpoints of the segment and hence with speed at most v_0. Thus an internal segment cannot overtake a corner. We have thus shown:

Theorem 1 (Consistency of Snap Rounding) *Snap Rounding does not introduce crossings between polysegments.*

There are some further properties which deserve being stated. All of them are fairly direct consequences of the consistency theorem of snap rounding.

Lemma 2 *After snap rounding:*

- *the edges of the polysegments (= fragments) do not intersect except at their endpoints.*
- *the circular order of the fragments ending at the same pixel is consistent with the circular ordering at the pixel's boundary of the corresponding ursegments crossing this pixel*

- *If ursegment r intersects ursegment s before ursegment t, then polysegment ρ cannot intersect polysegment τ before σ.*

In other words, snap rounding can change a proper before into an equal but not into a proper after.

Implementing Snap Rounding Efficiently: We show how to realize snap rounding by plane sweep (= Bentley-Ottmann sweep for line segment intersection, see any book on Computational Geometry). Recall that a pixel is hot if it contains a vertex of the original scene. Add the boundaries of the hot pixels as artificial edges and sweep the scene. The intersections between ursegments and artificial edges correspond to ursegments entering and leaving hot pixels. Replace each ursegment by the polysegment connecting the centers of hot pixels crossed by the segment.

If n is the number of vertices in the original scene and s denotes the total number of intersections between ursegments and hot pixels, the time required is $O((n+s)\log(n+s))$. Observe that the result scene consists of $O(n+s)$ segments.

Dynamic Snap Rounding: Guibas and Marimont [GM98] consider the problem of computing the snap rounded arrangement defined by a set of line segments. The naive approach would first construct the exact arrangement and then snap round it. Guibas and Marimont show how to modify Mulmuley's segment intersection algorithms to as to construct the rounded arrangement directly. This is clearly a nice result. I doubt however, that there is a general strategy for geometric rounding which avoids the construction of the "exact object" as a first step. I would not pursue this line of research.

Iterated Snap Rounding: Halperin and Packer observe that snap rounding may introduce edges passing close to vertices. In other words, it may create new hot pixels. They suggest to iterate snap rounding until there no further hot pixels.

2.3 Shortest Path Rounding

Milenkovic [Mil00] introduced shortest path rounding. It introduces even fewer kinks than snap rounding. Milenkovic also gives a formal definition of geometric rounding.

Mathematical Definition of Geometric Rounding Let G be a planar subdivision, i.e, a plane embedding of a graph. Each node of G is mapped to a point in the plane and each edge is mapped to a curve connecting the endpoints of the edge. The curves are disjoint (except at endpoints).

We may change the embedding my deforming the plane. I use E to denote the two-dimensional Euclidean plane (= $\mathbb{R}^2$ with the Euclidean metric).

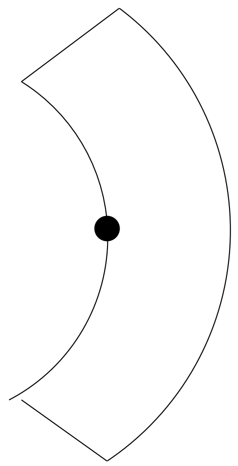

Figure 1: A cell of points all rounded to the point highlighted. The cell is not star-shaped with respect to the highlighted point and a corner point cannot be rounded along a straight-line. This cell shape occurs in rounding polar coordinates.

Definition 1 *A* deformation *is a continuous mapping*

$$\pi[0,1] \times E \mapsto E$$

such that $\pi(0,p) = p$ *for all* $p \in E$ *and such that for all* $t \in [0,1)$*, the function* $p \mapsto \pi_t(p) := \pi(t,p)$ *is a bijection.*

Observe that π_1 is not required to be a bijection, i.e., at time $t = 1$, distinct points may collapse. However, since before time t, distinct objects must stay distinct, there is no time to introduce new crossings.

A geometric rounding specifies for each vertex v a target position $\rho(v)$ and a simple path γ_v from v to $\rho(v)$. The paths γ_v and γ_w for distinct vertices v and w must be disjoint except at endpoints, i.e, $\rho(v) = \rho(w)$ is allowed.

Definition 2 *A geometric rounding of a subdivision* G *is a deformation with the following additional property: Every vertex* v *travels along* γ_v *from* v *to* $\rho(v)$ *during the deformation.*

In snap rounding, the points rounded to a particular lattice point form a square centered at the lattice point. Then γ_v can be taken as the straight line segment from v to $\rho(v)$. Milenkovic allows the paths γ_v to be arbitrary, see Figure2. Where could this be needed?

Consider the representation of points by polar coordinates (r, ϕ). We could round r down to the nearest multiple of $1/M$, and ϕ to the nearest angle of the form $i2\pi/N$ where N is a fixed integer. We cannot choose straight-line segments as our rounding paths. Instead, we could first round the radius (time 0 to 1/2) and then the angle (time 1/2 to 1). In this situation rounding paths are straight line segments followed by circular arcs, see Figure 2.3

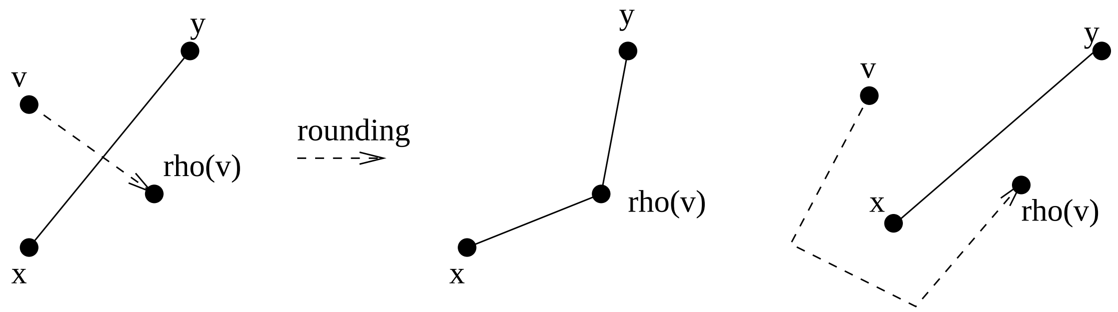

Figure 2: In the figure on the left the rounding path $v \mapsto \rho(v)$ deflects the edge xy, in the figure on the right, the edge is unaffected

Shortest Path Rounding: Let me next define shortest path rounding.

Definition 3 *A shortest path rounding (see Figure 2) of an edge* uv *is a polygonal chain* C *connecting* u *and* v *such that*

- *there is a geometric rounding of* (V, uv) *(observe just the edge* uv *and all vertices) transforming* uv *into* C*.*
- C *has shortest length among all chains having this property.*

Observe that each edge uv is rounded individually. The edges interact through the nodes which are present as obstacles for each rounding.

Theorem 2 (Properties of Shortest Path Rounding)

- *The result of shortest path rounding is unique and depends only on the equivalence classes of the rounding paths (i.e., we may replace any rounding path by an equivalent one and obtain the same result).*
- *The union of the shortest path roundings of the individual edges of a straight-line subdivision* G *is a geometric rounding of* G*.*

What does it mean for two rounding paths to be equivalent: one can be deformed into the other without moving the path over a node in $\{v, \rho(v) \; ; \; v \in V\}$.

The paper contains some nice pictures illustrating shortest path rounding, also for an unusual choice of rounding path. Milenkovic describes two general algorithms and one algorithms working for particularly simple rounding regions.

A Local Algorithm: The first algorithm simulates physics. Simply do the following. Move all vertices continuously along the rounding path from their original to their target position; the speed is chosen such that the points reach their destination at time 1. As the vertices move, they deflect the edges into polygonal paths. If one thinks of the edges as rubber bands, the polygonal paths stay shortest. The local algorithm manipulates all paths simultaneously (but the paths do not notice each other; they only notice the forces from the moving vertices).

Implementation is easy. We keep an event queue. An event is three vertices becoming collinear. At an event point, a kink may appear or disappear in one of the polysegments. The edges stay shortest at all times. There can be a large number of events. Even if all rounding paths are straight, there may be as many as $O(n^3)$ events.

Since the edges are replaced by rubber bands they are shortest at all times and hence the system always is in a state of locally minimal potential energy.

As described the local algorithm is not a deformation. We can make it a deformation by replacing every edge by a small tube. The actual edge runs along the medial axis of the tube and the width of the tube goes to zero as time progresses. Clearly, the result will be the same.

A Global Algorithm: For each edge uv we compute a shortest topologically equivalent path. The rounded path must not move across vertices.

We use a standard method to specify topological equivalence. Let T be a triangulation of the plane having the rounded nodes as its vertices. Any path connecting two rounded nodes $\rho(u)$ and $\rho(v)$ can than be described by the sequence of triangles it passes through. We can transform sequences by the rules

- $\ldots T_i T_j T_i \ldots$ is transformed into $\ldots T_i \ldots$
- if the first and the second triangle are incident to $\rho(u)$, delete the first.
- if the last and next to last triangle is incident to $\rho(v)$, delete the last.

Application of the rules gives us a shortest list of triangles.

How can we construct an initial sequence of triangles, see Figure 3. Consider the following path from $\rho(u)$ to $\rho(v)$. We go from $\rho(u)$ to u (using γ_u), then along uv, and finally from v to $\rho(v)$ along γ_v. Whenever uv hits a rounding path γ_w connecting w and $\rho(w)$, we follow γ_w to $\rho(w)$ and then go back on the other side of γ_w. In this way we obtain a path from $\rho(u)$ to $\rho(w)$. Call this path the loose path $\rho(u)$ to $\rho(v)$. The path defines a sequence of triangles.

We simplify the sequence by using the cancellation rules introduced above. This gives us the sequence of triangles the path must pass through.

Finally, we use the algorithm of Guibas et al [GHL$^+$87] for computing shortest paths within simple polygons to compute the shortest path passing through our given sequence of triangles.

The result of global rounding is independent of the routing paths. Observe that the algorithm first constructs the topology of the rounded paths (the loose path) and then pulls it tight by canceling out triangles. Assume now that we continuously deform the rounding path. This will continuously deform the loose path. The sequence of triangles traversed will change only as described by the simplification rules and their inverses.

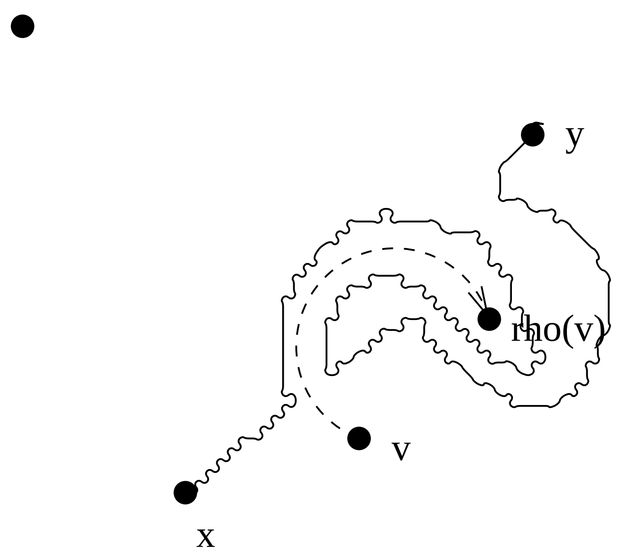

Figure 3: The path chosen for edge xy in first phase of global shortest path rounding.

It remains to show that the local and the global algorithm construct the same paths. This follows from the fact that the local algorithm constructs a local minimum in path length and that the global argument shows that there are no local minima except the global minimum.

Monotone Shortest Path Rounding: The strategy of rounding vertices to their nearest grid point allows one to optimize the calculation of the geometric rounding. Observe that the rounded path will be monotone in x- and y direction (it may run vertical or horizontal but it will not go back). This follows immediately from the local algorithm. Going back would imply that we have vertices v with w with $x(v) < x(w)$ and w being rounded to a point left of v.

We can therefore construct the shortest rounding with an algorithm akin to the computation of the upper hull of a set of points. We sort the vertices according to their projection onto the segment uv to be rounded and classify them as left and right. We construct the convex hull of the polygons on each side an obtain an hour-glass-like structure. The shortest path is then easy to obtain.

An Application to a Placement Problem: Victor describes an application to a placement problem. I understand that the application has unbounded cascading of geometric Minkowski sum computations. I do not see why shortest path rounding solves his problem. It seems to me that he also needs some inclusion property, e.g., that a region is always rounded to a superset.

3 Simplifying Polygonal Chains and Subdivisions

Consider a polygonal chain P without self-intersections (in brief: a simple polygonal chain). The basic task is the following:

Task 1 *Given a simple polygonal chain P and a positive real ϵ, find a simplest equivalent polygonal chain Q with distance at most ϵ of P.*

As stated, the task is not a mathematical problem yet. What do we mean by simplest chain? What do we mean by equivalent? What do we mean by distance? Do we allow Steiner points?

Steiner points: Must all vertices of Q be also vertices of P or do we allow to invent vertices? In the former case, we call Q a no-Steiner-point-simplification. In the latter case, we call it a simplification.

Distance: There are several possible definitions:

- for every point p on P, there must be a point q on Q of distance at most ϵ.
- for every point q on Q, there must be a point p on P of distance at most ϵ.
- both items above
- Let a and b be the endpoints of P and Q. One must be able to simultaneously move a point p from a to b on P and a point q from a to b on Q such that their distance is never more than ϵ. This is called Frechet distance or dog-leash distance.
- As above, but one always has to move forward.
- The total area of the bounded regions enclosed by $P \circ Q^{-1}$ is at most ϵ.

Equivalence: We require that Q is homotopic to P. We identify paths with continuous functions from $[0,1]$ into the plane. A homotopy of two path P and Q with endpoints a and b and within a region $\mathcal{R}$ is a continuous functions H from $[0,1]^2$ to $\mathcal{R}$ such that $H(x,0) = P(x)$, $H(x,1) = Q(x)$ for all x and $H(.,t)$ is a path from a to b in $\mathcal{R}$ for all $t \in [0,1]$. If we take $\mathcal{R}$ as the entire plane the only requirement is that P and Q have the same endpoints. If we take $\mathcal{R}$ as a proper subset of the plane, we put additional restrictions on Q.

For example, when simplifying a polyline representing the border of a country, we might take $\mathcal{R}$ as the plane minus the points representing cities. Then homotopy ensures that no city switches countries.

Or we might take $\mathcal{R}$ as the points of distance at most ϵ from P. Then homotopy guarantees that Q is close to P.

Exercise 2 *Define homotopy for subdivisions.*

Simplicity: In some applications (all??) it is important that Q is a simple path.

Complexity of the Solution: The natural measure is to count the number of edges of Q.

3.1 Negative Results

I survey some negative results.

Guibas/Hershberger/Mitchell/Snoeyink [GHMS91] show two negative results.

The first one concerns simplifying paths. The problem is to simplify a path P into a simple path Q of the same homotopy. The region $\mathcal{R}$ is a punctured plane, i.e., the plane with a polynomial number of holes. Steiner points are allowed. An instance is a path P, a region $\mathcal{R}$ and an integer k. The question is whether there is a simple path Q with at most k links homotopic to P. This problem is NP-complete.

The second result concerns simplification of subdivisions. We are given a subdivision S in a polygonal region $\mathcal{R}$. Can S be simplified to a "subdivision" S' with at most k links. The edges of S' are allowed to cross. The problem is NP-complete (even without the requirement of simplicity.

Estkowski and Mitchell [EM01] have an even stronger (= more discouraging) result. For them an instance is a subdivision S and a parameter ϵ. We simplify the subdivision by removing vertices of degree two; in this way polychains are replaced by simpler polychains. The requirement is that no intersections are introduced and that if a chain P connecting a and b is replaced by the segment $\overline{ab}$, then no point of P has distance more than ϵ from $\overline{ab}$. The goal is to construct a subdivision S' with a minimal number of vertices. This problem is hard to approximate. Let n be the number of vertices of S and let opt be the minimal number of vertices in a simplified subdivision. Then for any positive δ, no polynomial time algorithm can be guaranteed to construct an S' whose number of vertices is less than $optn^{1/5-\delta}$, unless P = NP.

Question 1 *Are these problems tractable in practice?*

The paper by Estkowski and Mitchell suggests that the answer is yes despite the negative results just cited.

3.2 Simplification without Steiner Points

No Homotopy Requirement: Douglas and Peuker [DP73] gave a simple heuristic for constructing an equivalent path Q. It is best described as a recursive procedure. Let a and b be the endpoints of P and consider the line segment $\overline{ab}$ connecting a and b. If every vertex of P (and hence every point on P) has distance at most ϵ from $\overline{ab}$, use $\overline{ab}$ as the simplification of P. Otherwise, let c be a vertex of P of maximal distance from $overlineab$ and return the simplification of $P[a, c]$ followed by the simplification of $P[c, b]$.

The simplification guarantees that for every $p \in P$ there is a $q \in Q$ of distance at most ϵ. It also guarantees the converse. We only need to show this for the base case of the recursion where we replace P by the segment $\overline{ab}$. For every point $p \in P$ let $q(p)$ be a point on $Q = \overline{ab}$ closest to p. Since Q is a line segment, $q(p)$ is unique. Also $q(a) = a$, $q(b) = b$, and $q(p)$ changes continuously as p moves from a to b along P. Thus $q(p)$ moves along Q from a to b. We conclude that for every $q \in Q$ there is an $p \in P$, namely the p with $q(p) = q$, such that the distance from q to p is at most ϵ. We have just proved that we also have distance less than ϵ in the dog-leash distance. There is no bound for the monotone dog-leash distance.

The naive implementation of the Douglas-Peuker heuristic has running time $O(n^2)$, since it uses linear time on every level of the recursion to find r. Hershberger and Snoeyink [HS92, HS97] reduce the running time first to $O(n \log n)$ and later to $O(n \log^* n)$. They make use of a very nice algorithm of Melkman [Mel87] to determine the convex hull of a simple polygon.

Douglas and Peuker do not minimize the number the number of edges in Q. One can compute the simplification with the minimum number of links as follows. Define the short-cut graph G as follows. Let $v_1, \ldots, v_n$ be the vertices of P. We have the edge (v_i, v_j) with $i < j$ if no vertex of $P[v_i, v_j]$ has distance larger than ϵ from $\overline{v_i, v_j}$, i.e., if the segment $\overline{v_i, v_j}$ is a good approximation of the chain. We then compute a minimum length path from v_1 to v_n in the short-cut graph. The short-cut graph may have quadratic size. It is acyclic and hence the shortest path computation takes time $O(m) = O(n^2)$. The construction of the short-cut graph can also be done in quadratic time [CC96]. The naive algorithms takes time $O(n^3)$.

Question 2 *Can one compute a no-Steiner-points-simplification with a near minimum number of links in near linear time?*

The Douglas-Peuker algorithm may generate a non-simple simplification. This is undesirable in many applications. For example, a polygonal chain may represent the border of a country or an iso-line in a map. Then self-intersections are an error.

Question 3 *Can one compute a simple minimum-link (or near minimum link) no-Steiner-points-simplification in polynomial time? Is the problem NP-complete?*

There is some indication [GHMS91, EM01] that the answer is NO, but, as far as I know, it has not actually be shown.

With Homotopy Requirement: Mark de Berg, Marc van Kreveld, and Stefan Schirra [dBvKS95] extend the approach of the previous paragraph to handle homotopy requirements. Let $\mathcal{R}$ be a punctured plane, i.e., the plane with a set X of points removed. A short cut edge (v_i, v_j) is homotopy-preserving if $P[v_i, v_j]$ is homotopic to the line segment $\overline{v_i v_j}$. In other words, straightening out the subpath $P[v_i, v_j]$ will not move the subpaths across a point in X.

It is easy to check in polynomial time whether (v_i, v_j) is homotopy-preserving. We triangulate the plane using the points in X as the vertices of our triangles. Then we check whether the sequence of triangles traversed by (v_i, v_j) and by $P[v_i, v_j]$ are equivalent (see the Section on shortest path rounding).

The authors show:

- For an x-monotone chain P with n vertices and a set X of m points, one can find all homotopy-preserving shortcuts in time $O(n(n+m)\log n)$.
- In the same time, one can find a minimum-link homotopic simplification with maximum error ϵ.

  One takes only the shortcuts that are homotopy-preserving and obey the distance constraints. The one finds a shortest path in the shortcut graph.
- In the same time one can find a homotopic simplification with maximum error ϵ for an arbitrary chain. NO guarantee on the number of links is given.

They go on to show that the algorithm for homotopic simplification of polylines with maximum error ϵ can be used to simplify subdivisions. Let S be a subdivision and let X be the set of vertices of S. They suggest to simplify each chain of degree two vertices separately. Since the simplifications are homotopic with respect to vertex set X no crossings and no self-intersections are introduced.

Question 4 *How well does this algorithm perform in practice? It is easy to give examples, where the algorithm does not do well. It might depend on the application. Maybe, the algorithm works well for maps that are not crowded, and does not work well for crowded maps (elevation maps).*

Estkowski and Mitchell [EM01] describe a different approach and give very impressive experimental results for their approach. They first simplify each chain separately using any line simplification algorithm (e.g., Douglas-Peuker) and then repair the resulting subdivision S'. More precisely, they construct a sequence $S_0 = S'$, S_1, S_2, $\ldots$ of subdivisions with each subsequent S_i being closer to simplicity.

We describe how S_{i+1} is obtained from S_i. Let s and s' be a pair of intersecting segments in S_i (if they do not exist, S_i is simple and we are done) and let $C(s)$ and $C(s')$ be the chains in the original subdivision simplified to s and s', respectively. Then $C(s)$ and $C(s')$ do not intersect.

Select one of the segments, call it s, as the detour segment. In their implementation they choose s such that s intersects $C(s')$ an odd number of times.

Let me briefly argue that such an s exists. For two chains P and Q let $\#(P, Q)$ be their number of intersections. We have

$$\begin{aligned}\#(s \circ C(s), s' \circ C(s')) &= \#(s, s') + \#(s, C(s')) + \#(C(s), s') + \#(C(s), C(s')) \\ &= 1 + \#(s, C(s')) + \#(C(s), s')\end{aligned}$$

Two closed chains intersect an even number of times and hence the number on the left is even. Thus exactly one of the two terms on the right must be odd.

Next construct a short-cut graph for $C(s)$. A segment is added to the short-cut graph if it does not intersect s' and if it is ϵ-feasible (i.e., maximal distance from replaced subchain is at most ϵ). Then a minimum link path in the short-cut graph is constructed and s is replaced by this path.

If there is no path in the short-cut graph for either s and s' the algorithm goes into panic mode. They consider the graph of ϵ-feasible shortcuts for $C(s)$ and $C(s')$ (but exclude the trivial shortcuts s and s') and find minimum link short-cuts for each. This will not necessarily remove the intersection.

The algorithm terminates because each iteration introduces at least one additional vertex and no vertex is ever removed. The authors give an example requiring n iterations. In their experiments, a small number of iterations sufficed.

An extended heuristic also handles homotopy requirements. Please take a look at the paper to see the very impressive experimental results.

3.3 Simplification with Steiner Points

I discuss a paper by Hershberger and Snoeying [HS94]. They show how to compute minimum-length and minimum-link paths of a given homotopy. See [GJK$^+$88] for a local paper on the subject.

Let M be a triangulation of some bounded polygonal region (holes are allowed) with all vertices of M on the boundary of the unbounded face. Let α be a path connecting two points p and q of M. Again we view paths as continous maps from $[0,1]$ into M. Thus $\alpha(0) = p$ and $\alpha(1) = q$. A path β is homotopic to α if there is continuous map H H from $[0,1]^2$ to M such that $H(x,0) = \alpha(x)$, $H(x,1) = \beta(x)$ for all x and $H(.,t)$ is a path from p to q in M for all $t \in [0,1]$. This is exactly as above. I have repeated the definition in order to be consistent with the notation used by Hershberger and Snoeyink.

They show how to compute minimum length and a minimum link paths homotopic to a given path α in time $O(C_\alpha + \Delta_\alpha)$. Here C_α is the number of vertices of α and Δ_α) is the number of times α crosses a triangulation edge of M.

Shortest Paths: We proceed in two phases. We first determine the sequence of triangulation edges crossed by β and then construct β.

For the first phase, consider the sequence of triangulation edges crossed by α. We simplify the sequence using the rule $xeey \mapsto xy$. The rule is justified by the observation that no shortest path will cross the same edge turn around and then come back.

For the second phase, the funnel method is used. Let $e_1, e_2, \ldots, e_k$ be the triangulation edges crossed by α (after simplification). We construct a funnel for each edge e_i. Let $e_i = \overline{uv}$ and consider the shortest paths from p to u and v, respectively. They have a common initial part and then split into two paths, one always turning left and one always turning right, see Figure 4 in their paper. Figure 5 shows how the funnel changes when proceeding from e_i to e_{i+1}. Of course, the art is to make the update fast.

A remark on recurrences of the form

$$T(n) = \max_{1 \le i < n} (T(i) + T(n-i) + f(i, n-i))$$

is in order at this point. We have

$$T(n) = \begin{cases} O(n^2) & \text{if } f(i, n-i) = n \\ O(n \log n) & \text{if } f(i, n-i) = \log n \\ O(n \log n) & \text{if } f(i, n-i) = \min(i, n-i) \\ O(n) & \text{if } f(i, n-i) = \log \min(i, n-i) \end{cases}$$

Minimum Link Paths: We first compute a shortest length path homotopic to α. Call it again α. When traversing α from p to q we can classify vertices and left and right turns. An edge is called an inflection edge if its endpoints have distinct labels or if it is incident to p or q.

Lemma 3 *Let e be an inflection edge. Then there is minimum link path from p to q containing a segment supported by e.*

Proof: We argue for an edge which is not incident to either p or q. Extend e on eihter side until it hits the boundary of M. The extended segment splits M into several parts and p and q do not belong to the same part. Consider a minimum link path β and assume that is does not contain a segment supported by e. Then it must intesect the extension of e at least thrice. We can therefore shortcut, without increasing the number of links. See Figure 6 in their paper. ∎

Now read page 10 of their paper.

4 Some Questions

I list some questions which came to my mind.

Minimum-Link Simplifications without Self-Intersections: Let P be a simple polygonal chain. We want to find a minimum-link simple chain with maximal error ϵ. I believe that the complexity of this problem is open. The task is easily formulated as an integer linear program or as a constraint satisfaction problem.

Let a and b be the endpoints of P. We have a variable x_p for every vertex p of P and a variable x_e for every ϵ-feasible shortcut.

For the integer linear program our variables are 0/1-variables. We have the constraints:

$$x_p \in \{0, 1\} \text{ for all } p \quad x_a = x_b = 1$$

$$\sum_{\{e\ ;\ e \text{ is incident to } p\}} x_e = d_p x_p \quad \text{for all } p$$

where $d_a = d_b = 1$ and $d_p = 2$ otherwise. Finally

$$x_e + x_f \leq 1 \quad \text{if edges } e \text{ and } f \text{ intersect in their interior}$$

The goal is to minimize $\sum_e x_e$. Let me argue that the solution is a path from a to b. A vertex p is either chosen ($x_p = 1$) or not chosen ($x_p = 0$). A chosen vertex has degree two, except for a and b which have degree one. Thus the solution consists of a path connecting a and b and a number of cycles. Since the goal is to minimize the number of edges chosen, there are no spurios cycles.

What happens if we drop the integrality constraints? Give an example where the optimal solution is fractional.

The constraint $x_e + x_f \leq 1$ can be strengthened. Consider any set X of edges any two of which intersect in their interior. Then $\sum_{e \in X} x_e \leq 1$. For the ILP this constraint is implied. It might strengthen the LP-relaxation. How can one generate such clique constraints efficiently?

Alternatively, we might use constraint satisfaction techniques. We explore the solution space by enumeration. Suppose we have already fixed some variables. We can fix other variables using the constaints that edges are not allowed to cross and that we need to choose exactly two edges incident to each node. Also, we can close a partial instantiation if it cannot be extended to a path anymore.

How could we do the optimization. We could add an additional constraint that the sum of the x_p variables should be at most k. We would optimize by doing binary search on k.

5 Rounding Polyhedra

```
@inproceedings{ fortune98vertexrounding,
    author = "Steven Fortune",
    title = "Vertex-Rounding a Three-Dimensional Polyhedral Subdivision",
    booktitle = "Symposium on Computational Geometry",
    pages = "116-125",
    year = "1998",
    url = "citeseer.nj.nec.com/fortune99vertexrounding.html" }
```

```
@InProceedings{Milenkovic90,
  author =       "Milenkovic",
```

```
  title =        "Rounding Face Lattices in d Dimensions",
  booktitle =    "CCCG: Canadian Conference in Computational Geometry",
  year =         "1990",
}
```

References

[CC96] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *Internat. J. Comput. Geom. Appl.*, 6:59–77, 1996.

[dBvKS95] M. de Berg, M. van Kreveld, and S. Schirra. A new approach to subdivision simplification. In *Auto-Carto 12*, pages 79–88, 1995.

[DP73] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10:111–122, 1973.

[EM01] R. Estkowski and J.S.B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Symposium on Computational Geometry*, pages 40–49, 2001.

[GGHT97] M. Goodrich, L. Guibas, J. Hershberger, and P. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 284–293, New York, June 4–6 1997. ACM Press.

[GHL+87] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[GHMS91] Guibas, Hershberger, Mitchell, and Snoeyink. Approximating polygons and subdivisions with minimum link paths. In *ISAAC: 2nd International Symposium on Algorithms and Computation*, 1991. appeared in Computational Geometry Theory and Appliations.

[GJK+88] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *ACM Geometry Conference 88*, pages 392–402, 1988.

[GM98] L. J. Guibas and D.H. Marimont. Rounding arrangements dynamically. *International Journal of Computational Geometry and Applications*, 8:157 – 176, 1998.

[GY86] D. H. Greene and F. E. Yao. Finite-resolution computational geometry. In *27th Annual Symposium on Foundations of Computer Science*, pages 143–152, Los Angeles, Ca., USA, October 1986. IEEE Computer Society Press.

[Hob99] John D. Hobby. Practical segment intersection with finite precision output. *Computational Geometry Theory and Applications*, 13(4):199–214, October 1999.

[HP] Dan Halperin and Eli Packer. Iterated snap rounding.

[HS92] John Hershberger and Jack Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, volume 1, pages 134–143, Charleston, South Carolina, 1992.

[HS94] Hershberger and Snoeyink. Computing minimum length paths of a given homotopy class. *CGTA: Computational Geometry: Theory and Applications*, 4, 1994.

[HS97] John Hershberger and Jack Snoeyink. Cartographic line simplification and polygon CSG formulae and in o(n log* n) time. In *Workshop on Algorithms and Data Structures*, pages 93–103, 1997.

[Mel87] A. Melkman. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.*, 25:11–12, 1987.

[Mil00] Milenkovic. Shortest path geometric rounding. *ALGRTHMICA: Algorithmica*, 27, 2000.