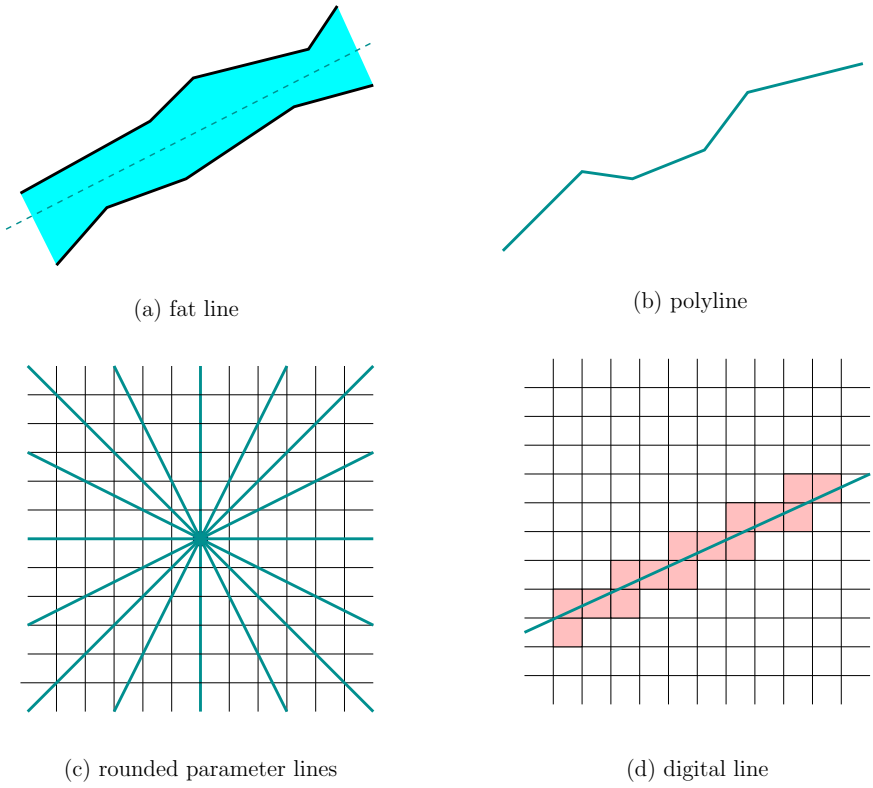
Lecture 5
GEOMETRIC APPROACHES

We look at some geometric approaches to nonrobustness. An intriguing idea here is the notion of "fixed precision geometry". After all, if nonrobustness is a geometric phenomenon, it makes sense to modify the geometry to reflect the fixed precision we find in numbers. There are many ways to create such ersatz geometries*, which in various ways try to recover the basic properties of standard Euclidean geometry. We illustrate several such geometries through a discussion of "fixed precision lines".*

§1. What is a Finite Precision Line?

Since qualitative errors is geometric in nature, we may attempt to modify the underlying geometry to be achieve robust behavior. E.g., with finite precision arithmetic, it seems reasonable to replace standard Euclidean geometry by some finite precision geometry. We have already met such a geometry in Chapter 1: we interpreted the standard epsilon-tweaking trick as manipulating new kinds of "points" and "lines", namely, fat points and fat lines.

There are many potential candidates for finite-precision geometries, even for the simple concept of lines. We give some common answers [26]. to the question "what is a finite precision line?" Each answer can be viewed as representative of a general approach to finite precision geometry.



(a) fat line

(b) polyline

(c) rounded parameter lines

(d) digital line

Figure 1: Finite Precision Line Geometry

- Interval Geometry: We use the term "interval geometry" to refer any approach which replaces standard geometric objects by "fat geometric objects". A "fat point" can be a disc, but it can be generally be any simply-connected bounded region. A "fat line" can be any region bounded by two infinite polygonal paths; the region is retractable to a continuous curve that separates the plane. Such a fatline is illustrated in Figure 1(a), generalizing the concept in Lecture 1. This is the geometric analogue of interval arithmetic, and has been proposed in many different settings, e.g., [19, 12]. Guibas and Stolfi [6] investigated a form of such geometry ("ε-geometry") by focusing on geometric ε-predicates that can have yes/no/uncertain values. Interval geometry is also attractive from another viewpoint: in the field of mechanical design and manufacture, the tolerancing of geometric objects [25, 13] is a critical problem. The computational aspects of this field is being pursued under the name of computational metrology [9, 27, 14].

- Topologically Consistent Geometry: A line can be distorted into a curve, which in practice would become a **polyline**, i.e., a polygonal line that separates the plane into two components. This is illustrated in Figure 1(b). The goal of distortion is to preserve some desired topological property of the original lines. Such polylines arise when computing the arrangements of lines in which intersection points must be rounded to grid points. Consider the following grid model studied by Greene and Yao [4]. Let G be a grid, typically $G = \mathbb{Z} \times \mathbb{Z}$. Assume the input are segments whose end points lie on this grid. For each input pair (s, s') of intersecting line segments, we move the intersecting point to a close grid point $p \in G$. The segment s, s' are now converted into polysegments (a polygonal path) with the same end points as before, but now passing through p. Greene and Yao require the transition from s into a polysegment not to cross any gridpoint, leading to polysegments with continued fraction type properties.

  Sugihara and Iri [23, 24] propose topological consistency as a fundamental principle for achieving robustness.

- Rounded Parametric Geometry: Assume our geometric objects live in some parametric space. For instance, lines can be represented in the form $L = \mathsf{Line}(a, b, c)$ as in Chapter 1. This corresponds to the Euclidean line with equation $aX + bY + c = 0$. We want to round the line parameters (a, b, c) to some (a', b', c') where a', b', c' comes from some finite set of representable values (say L-bit integers). We call $\mathsf{Line}(a', b', c')$ a **rounded line**. This approach was introduced by Sugihara [20] who discussed several approaches to rounding lines. For instance, we might like to restrict $|a'|, |b'|, |c'|$ to lie in the range $[0, 2^L)$. Suppose $L = 1$, then a', b', c' comes from the set $\{-2, -1, 0, 1, 2\}$. In Figure 1(c), we show all 8 lines for the case $L = 1$ in which $c' = 0$.

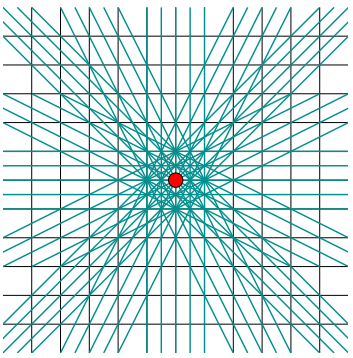


Figure 2: All 40 parametric lines with $L = 1$.

In Figure 2, we show all the 40 possible lines for the case $L = 1$. You will note a somewhat uneven

distribution of lines; Sugihara suggested that we get a nicer class of rounded lines if we let allow $|c'|$ to be in the larger range $[0, 4^L)$.

- Digital Geometry: This is well-known in computer graphics: a line is a suitable set of pixels on a finite screen. In contrast to the previous rounding in parameter space, in digital geometry we discretize the underlying Euclidean space where the geometric object lives. Figure 1(d) illustrates a digital line represented as a set of darkened pixels. This is known as a Bresenham line in computer graphics. In the field of image processing, an area called "digital geometry" investigates properties of such discretized representation of objects.

¶1. Software and Language Support. In the previous Lecture, we noted the available software and language support for the arithmetic approaches to non-robustness. For geometric approaches, we need higher level software support. There are well-known numerical libraries such as `LAPACK` which users can use to implement robust geometric primitives. Newer efforts such as `LEDA` and `CGAL` aim to directly provide robust geometric primitives, and basic algorithms and data structures as building blocks for large-scale geometric applications. Important primitives that ought to be supported by languages or libraries include scalar product [16] and accurate determinant evaluation. More generally, we could extend programming languages to support geometric objects and their manipulation [18].

Exercises

Exercise 1.1: Consider lines with integer parameters (a, b, c) representing the line with equation $aX + bY + c = 0$. Let $\mathcal{L}(A, B, C)$ denote the set of lines (a, b, c) where $|a| < A, |b| < B, |c| < C$.
(i) Draw the set $\mathcal{L}(2^L, 2^L, 2^L)$ where $L = 3$.
(ii) Draw the same set with $\mathcal{L}(2^L, 2^L, 4^L)$ where $L = 3$.
(iii) Why did Sugihara suggest that $\mathcal{L}(2^L, 2^L, 4^L)$ is superior to $\mathcal{L}(2^L, 2^L, 2^L)$? ♢

End Exercises

§2. Robust Line Segment Intersection

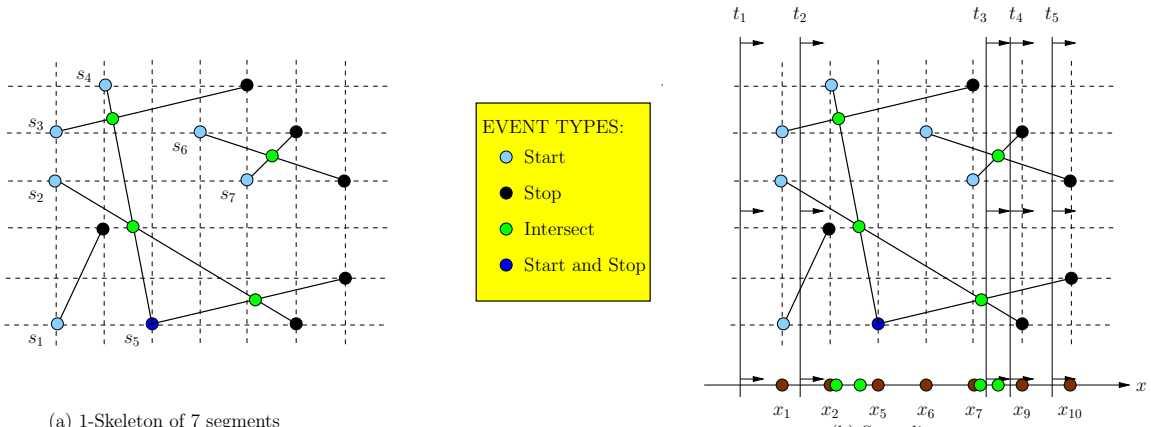
We consider another fundamental problem in computational geometry: given a set S of line segments (or "segments" for short) in the plane, we want to compute their arrangement. This is a very basic problem that arises in VLSI circuit design, hidden line elimination, clipping and windowing, physical simulations, etc. The arrangement of S is a **1-skeleton** (or network) determined by the segments and their intersections: a 1-skeleton is basically a graph that is embedded in the plane: its vertices are the endpoints of segments and the intersection points. Its edges are the open portions of segments bounded by two vertices. Figure 3(a) illustrates the 1-skeleton of a set $\{s_1, \ldots, s_7\}$ of segments.

There are several algorithms for this problem. The asyptotically best algorithm is from Chazelle and Edelsbrunner, with running time $O(k + n \log n)$ where k is the number of intersection points and n is the number of segments. Here k is the output parameter and ranges from 0 to n^2. More practical algorithms based on randomization have been given Clarkson and also Mulmuley. Algorithms that takes into account robustness issues have been studied by Greene and Yao [4], Milenkovic [15], and Sugihara [21]. The approach of Hobby [7], Guibas and Marimont [5] and Goodrich, et al [3] are based on the concept of "snap rounding", which we will treat in detail.

Recently, Preparata et al considered another issue, namely modifying the primitives using in these algorithms so that the algebraic degree is as small as possible. Although this clearly can help FP computations, it is useful even in AP computation.

¶2. Bentley-Ottman Algorithm. The classic algorithm for computing the 1-skeleton of a set of segments is from Bentley and Ottmann. This algorithm takes time $O((n+k) \log n)$ where n is the number of segments and k the number of pairwise intersections. It is quite easy to describe. Later we will implement Hobby's snap rounding using this algorithm as its basis [7].

Let us briefly review the Bentley-Ottmann algorithm. We assume a vertical sweepline L parametrized by a real number t: let $L(t)$ denote the **sweepline at time** t, when it has the equation $x = t$. See Figure 3(b)



(a) 1-Skeleton of 7 segments

(b) Sweepline events

Figure 3: (a) 1-Skeleton, (b) Sweepline events.

where the position of the sweeplines at times $t = t_1, \ldots, t_5$ are shown. The line sweeps from left to right (so time t increases from $-\infty$ to $+\infty$). We will need two data structures Q and T:

- We need a min-priority queue Q, the **event queue**, initialized to store the events. Intuitively, the end points of each input segment represents an **event**. Each event has an associated point $p = (p.x, p.y)$ which is used as the priority of the event. We compare points p, q using their lexicographic ordering $\leq_{\text{LEX}}$:

  $$p \leq_{\text{LEX}} q$$

  iff $p.x < q.x$ or $(p.x = q.x$ and $p.y \leq q.y)$. If the endpoints of a segment s are p and q, and $p <_{\text{LEX}} q$, then we call p the **start point** and q the **stop point** of s. Initially, all the events in Q are either **start events** or **stop events**, corresponding to the leftmost or rightmost endpoints of a segment. Later we introduce a third kind of event, **intersection events**.

  See Figure 3(b) which illustrates the various kinds of events. More precisely, an event is a point p with some associated segments: a start (resp., stop) event is $e = (p, s)$ where p is the start (resp., stop) point of s, and an intersection event is $e = (p, s, s')$ where $p = s \cap s'$, and p is in the relative interior of s or the relative interior of s'. We call p the **event point** and $p.x$ the **event time** of e. Any given point p may be associated with more than one event (e.g., the blue point in Figure 3).

  > In the following, we shall assume that each point is associated with a unique event. The general case will be addressed in the Exercises.

- As L sweeps from left to right, we pause at each event time. Suppose all the event points are globally sorted as $p_1 <_{\text{LEX}} p_2 <_{\text{LEX}} \cdots$. Thus, we obtain

  $$x_1 \leq x_2 \leq \cdots$$

  where $x_i = p_i.x$ is the ith event time. By our assumption, there is a unique event e_i whose event point is p_i. However, this does not rule out the possibility that $x_i = x_{i+1}$. Of course, this global information is not known in advance. Nevertheless, we will have "just in time" information: our algorithm ensures that, after e_i has been processed, then the minimum priority event in Q will be e_{i+1}.

  Conceptually, at any moment t, we keep track of the set $S(t)$ of segments that intersect $L(t)$. The set $S(t)$ for $t = x_i$ is in transition, and so we only focus on the time between two consecutive event times, $x_i < t < x_{i+1}$. These segments in $S(t)$ are sorted by the y-coordinates of their intersection with $L(t)$. Although the actual y-coordinate depend on the value of t, their relative ordering is invariant between

two events. Suppose $S(t) = \{s_1, \ldots, s_k\}$ for some $k = k(t)$, and let segment s_j intersect the line $L(t)$ at the y-coordinate $y_j(t)$. We sort the $y_j(t)$'s so that

$$y_1(t) < y_2(t) < \cdots y_k(t). \tag{1}$$

Clearly, $k(t)$ and the set $S(t)$ is invariant for all $t \in (x_i, x_{i+1})$. Therefore, we may write k_i and S_i for $k(t)$ and $S(t)$. Likewise, the ordering (1) is invariant and it induces an ordering on the set S_i, which we denote by

$$s_1 <_i s_2 <_i \cdots <_i s_k. \tag{2}$$

- This total ordering (2) is maintained by our second data structure T, which is a balanced binary tree. The following invariant will be observed for T:

> $INVARIANT:$ (3)
>
> *For each pair adjacent segments $s_j < s_{j+1}$ in $T(t)$, if s_j, s_{j+1} properly intersect at a point q to the right of $L(t)$, then the intersection event (q, s_j, s_{j+1}) is in $Q(t)$.*

- Here is how we update T and Q with each event e:
  - If $e = (q, s_j)$ is a stop event, we delete s_j from T.
    To maintain the invariant (3): if s_j is not the first or last segment in T, then there are two segments s_{j-1}, s_{j+1} in T that are now adjacent after we delete s_j. We check if they intersect at some point q' to the right of the line $L(q.x)$; if so, put a new intersection event (q', s_{i-1}, s_{i+1}) into Q.
  - If $e = (q, s)$ is start event, we insert s into T.
    To maintain the invariant (3): suppose s is inserted between s_j and s_{j+1}. Then we must check if (s_j, s) and (s, s_{j+1}) generates any intersection events to the right of $L(q.x)$. If so, these are inserted into Q.
  - If $e = (q, s_j, s_{j+1})$ is an intersection event, then we can swap the relative ordering of s_j and s_{j+1} in T.
    To maintain the invariant (3): If $j > 1$, then we must check if (s_{j-1}, s_{j+1}) generates an intersection event to the right of $L(q.x)$. If so, we insert it into Q. If $j + 1 < k$, then we must similarly check if (s_j, s_{j+2}) generates an intersection event to the right of $L(q.x)$. If so, we insert it into Q.

This completes our algorithm. In the exercises, we ask you to work our the primitives that must be implemented, the degeneracies, and also the possibly errors that can occur because of numerical errors.

¶3. Snap Rounding on a Grid. We consider the following finite-precision model of geometry. We first begin with 1-dimension: any finite sequence (or set) of numbers

$$G : a_0 < a_1 < \cdots < a_m$$

is called a 1-**grid** on the interval $[a_0, a_m]$, also known as the **domain** of G. Each a_i is a **grid point**, and $\#(G) := m$ is the **size** of G. In the simplest case, we have the **uniform grid** of width $w > 0$, where $a_i - a_{i-1} = w$ for $i = 1, \ldots, m$. In illustrations, we always use uniform grids. A finite integer grid is the case where $w = 1$. However, a very important non-uniform example is where $G = F(2, t) \cap [a_0, a_m]$ is a set of floating point numbers.

Let us extend the above definitions to 2-dimensional grids: if G_i is a 1-grid on $[\ell_i, u_i]$ (for $i = 1, 2$), then the set $G = G_1 \times G_2$ is a 2-grid on the **domain** $[\ell_1, u_1] \times [\ell_2, u_2] \subseteq \mathbb{R}^2$. If a_i, b_i $(i = 1, 2)$ are adjacent points in G_i then the square $[a_1, b_1] \times [a_2, b_2]$ is called a **grid square**. The size of G is $\#(G) = \#(G_1)\#(G_2)$. Clearly, we can extend this to an d-grid in $\mathbb{R}^d$ $(d \geq 3)$.

Let us consider the geometry of points and line segments on the 2-grid G. Recall in the introduction to this Lecture, we discussed digital geometry or the computer graphics viewpoint – a point is a grid square, and

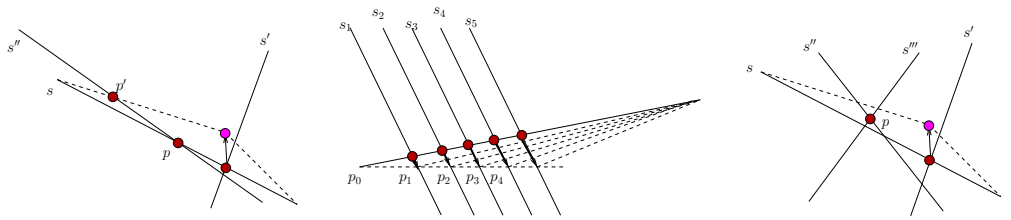
a line is a Bresenham set of grid squares. In the present discussion, especially since grid squares may have non-uniform sizes, we take a different approach. We assume that "points" are dimensionless, as standard Euclidean geometry. However, points must be "representable" — this means they must be grid points. Next, what are "representable" (line) segments? Initially, we may think of segments as defined by pairs to grid points. But when we consider the 1-skeleton of a set of segments, we see that our concept of segments must be generalized.

Suppose $s = [a, b]$ and $s' = [a', b']$ are two representable segments. Let s intersect s' in a point p; say the intersection is proper, i.e., p is in the relative interior of both s and s'. If p is not a grid point, we will "round" p to some nearby grid point q. In Lecture 1, we discuss a fundamental identity,

$$OnLine(Intersect(s, s'), s) \equiv TRUE.$$

To preserve this identity, we must next replace s (resp., s') by the "polysegment" $s = [a, q, b]$ (resp., $s' = [a', q, b']$).

More generally, a **polysegment** is a finite polygonal path $[q_0, q_1, \ldots, q_k]$ where each q_i is a grid point. We call $[q_{i-1}, q_i]$ $(i = 1, \ldots, k)$ a **subsegment** of the polysegment. If a polysegment $s = [\ldots, q_i, q_{i+1}, \ldots]$ passes sufficiently close to a grid point q in the sense that there is some point $p \in [q_i, q_{i+1}]$ which can can round to q, then we may modify s to the polysegment $[\ldots, q_i, q, q_{i+1}, \ldots]$, by inserting q into s. We say that s is **snapped** to q. This defines our basic model for segment geometry on the grid. There is also a naive method to compute the 1-skeleton of a set of segments: we successively find intersection points p between polysegments, round p to some grid point q, and then snap the underlying polysegments to q. This was first studied by Greene and Yao, who pointed several issues with the naive solution.



(a) Unbounded shift of p to p' (b) Drifting (c) Topological inversion of s relative to p

Figure 4: Issues in Snap Rounding

- Unbounded shift: when we snap a polysegment s, its intersection with another polysegment might move an unbounded distance, even though the snap distance is bounded. This is illustrated in Figure 4(a): the original intersection p of s and s'' is moved arbitrarily far to a new position p'.
- Drifting: Each time we modify a polysegment, the result may drift further and further away from its original position. Is there a bound on this drift? See Figure 4(a) for an illustration of drifting.
- Topological changes: See Figure 4(b) illustrates the possibility of a polysegment moving past an intersection point. This is an "inversion". A milder form of topological change is when two points collapse into one (this is called "degeneration").
- Cascaded and new intersections: when we snap a point, we may generate new intersections. If we now snap these new intersections points, we may obtain other new ones. Thus we have a cascading effect. How often do we need to do this? Does this terminate?
- Braiding: Two polysegments may intersect in a single connected component which need not be a single point (this can already happen even for segments). However, if the polysegments intersect in more than one connected component, we call this "braiding". Even if there were no inversion, braiding can happen.

  It is instructive to see how braiding arises as we successively snap our intersection points. In Figure 5(a), we have see two line segments s_1, s_2. Each s_i $(i = 1, 2)$ intersects other segments in three other nongrid

points. In Figure 5(b), we round two of these intersection points and s_i is snapped to these two rounded points. As a result of snapping, the third intersection point is shifted slightly; it is then rounded to a different point q_i that it would have originally been rounded to. In Figure 5(c), we further snap s_i to q_i. The resultant polysegments s_1, s_2 are now braided: they intersect in two isolated points q_1, q_2.

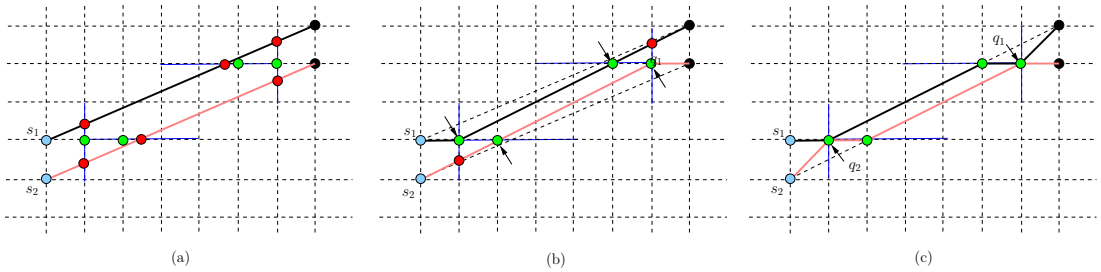


Figure 5: Braiding Behavior in Snap Rounding

This list of problems seems formidable [4]. So it is somewhat surprising to see the simple solution described next.

¶4. What is an acceptable topological change? Topological changes are inevitable in finite precision computation. But we propose to distinguish between **degeneration** (which is acceptable) versus **inversion** (which is not). We accept degeneration because when we round, we expect to loose details and close features might be expected to merge. For instance, if an edge in a planar arrangement of segments is collapsed to a point and it does not affect other relations (except for the obvious ones forced by this collapse), this is an acceptable "degeneration". On the other hand, if a vertex inside a cell is modify to lie outside the cell, this is an unacceptable "inversion".

In the presence of degeneration, there is a trivial solution for any segment arrangement: collapse everything to a point. So, our requirements ought to be supplemented by some metric properties such as a bound of the amount of degeneration. Greene and Yao proposed a solution for snap rounding that imposed a very strong requirement – the transformation from a segment into a polysegment must not cross any grid points. The resulting polysegments has nice properties related to the continued fraction process. Their method is elegant but do not extend to non-uniform grids. We will study another approach from Hobby which we extend to non-uniform grids.

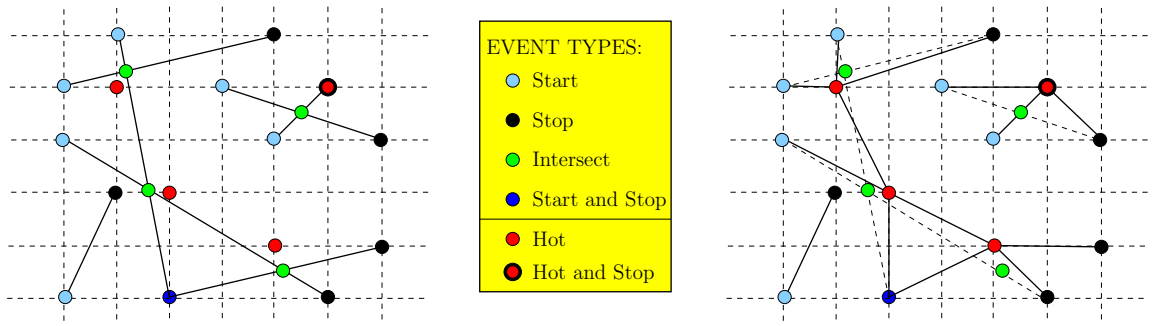
¶5. Hot Point Rounding. Hobby's proposal for snap rounding is based on the concept of "hot points". Given a set of segments, a grid point is hot if it is the rounded value of an endpoint or an intersection point of the segments. Then we just snap all the segments to these hot points to create our 1-skeleton. This solution is simple and has many nice properties (avoiding most of the anomalies of arbitrary snap rounding listed above). For instance, braiding (cf. Figure 5) does not occur in this approach. We show that Hobby's method will work on any rounding grids, not just uniform ones.

First, let us make precise the notion of rounding. If G_1 is a 1-grid with domain $[a_0, a_m]$ and $x \in [a_0, a_m]$, then the **floor** of x (mod G_1) is the largest grid point $\lfloor x \rfloor_G$ that is less than or equal to x. Also, $\lceil x \rceil_G$ denotes the grid point that is nearest to x; in case of a tie, we choose $\lceil x \rceil_G := \lfloor x \rfloor_G$.

We extend these notions to the 2-grid $G = G_1 \times G_2$ with domain D. If $p = (x, y) \in D$, then the **rounding** of p is the grid point $\lfloor p \rceil_G := (\lfloor x \rceil_{G_1}, \lfloor y \rceil_{G_2})$. If G is understood, we may simply write $\lfloor p \rceil$.

Let S be a set of segments, where each pair of segments intersect transversally (we do not allow overlap). Define H to be the set comprising all the end points of segments in S and also all intersection points. So H is finite. Let the rounded point set be $\lfloor H \rceil := \{\lfloor p \rceil : p \in H\}$. Following Hobby, we call the rounded points the **hot points**.

¶6. Properties of Hot Point Rounding. So far, we have just described polysegments to be any polygonal path $s = [q_0, \ldots, q_k]$. To ensure that polysegments are "generalizations" of segments, we impose a



(a) Original Input (b) Hot Snapped Output

Figure 6: Snap Rounding Based on Hot Points

simple restriction: a polysegment must be **monotone**. This means that either $q_{i-1} < q_i$ for all $i = 1, \ldots, k$, or $q_{i-1} > q_i$ for all $i = 1, \ldots, k$. Here, "$p < q$" means $p \leq q$ and $p \neq q$, where "$p \leq q$" means $p.x \leq q.x$ and $p.y \leq q.y$. Note that the $\leq$ ordering is not a total ordering on points, unlike the lexicographic ordering $\leq_{lex}$ used in our priority queue Q.

As a special case, a **staircase** is a monotone polysegment $s = [q_0, \ldots, q_k]$ where each $[q_{i-1}, q_i]$ $(i = 1, \ldots, k)$ is either a vertical or a horizontal segment.

We regard the polysegment $[q_0, q_1, \ldots, q_k]$ and $[q_k, , \ldots, q_1, q_0]$ as equivalent representations. To make a canonical choice among these two representations, we assume $q_0 \leq_{\texttt{LEX}} q_1$. This means $q_0.x$ is minimum among all the q_i's, and in case of ties, $q_0.y$ is minimum. Call this choice of q_0 the **start point** of the polysegment; then the other end point is the **end point**. Clearly, our sweepline L will encounter q_0 first among all vertices of the polysegment. However, this does not mean that the sweepline will encounter q_i before q_{i+1} for all $i \geq 1$ (how not?). There are two basic types of monotone polysegments: either the slope is non-decreasing, or it is non-increasing (or course, a perfectly horizontal polysegment has both classifications).

A polysegment $s = [\ldots, q_i, q_{i+1}, \ldots]$ is **near** a grid point $q \in G$ if there some point $p \in s$ such that $\lfloor p \rceil_G = q$. Provided q is not one of the vertices of s, recall that we can snap s to such a q. If $s = [\ldots, q_i, q, q_{i+1}, \ldots]$, the result of snapping is $s' = [\ldots, q_i, q, q_{i+1}, \ldots]$. Remark: even if $p = \lfloor p \rceil_G = q$, this snapping operation is meaningful, and we have extended the length of s by 1. Suppose that we consider the set of all grid points that s is near to, and snap s to all these points. Let $\overline{s}$ be the resulting polysegment. This line can be regarded as the analogue of the Bresenham line.

Let us define a **Bresenham path** in the 2-grid $G = G_1 \times G_2$ is a monotone polysegment $[q_0, \ldots, q_k]$ such that if $q_i = (a_i, b_i)$, then the sequence $(a_0, \ldots, a_k)$ are consecutive grid points of G_1 and the sequence $(b_0, \ldots, b_k)$ are consecutive grid points of G_2.

Bresenham paths are generally staircases: this means that for all $i = 1, \ldots, k$, either $a_{i-1} = a_i$ or $b_{i-1} = b_i$. However, there are exceptions: we might have $a_{i-1} < a_i$ *and* $b_{i-1} < b_i$. In this case, $[a_{i-1}, b_{i-1}] \times [a_i, b_i]$ is a grid square of G. Thus $[q_{i-1}, q_i]$ is a diagonal of this square. Such a situation may arise when the segment s passes through a midpoint of $[a_{i-1}, b_{i-1}] \times [a_i, b_i]$.

The following lemma is immediate (draw a picture):

Lemma 1. *If s is a monotone polysegment, then $\overline{s}$ is a Bresenham path.*

Given a segment $s \subseteq \mathbb{R}^2$, define the set $\text{Snap}(s) = \text{Snap}_G(s)$ of all the segments $[a, b]$ such that there exists $[a', b'] \subseteq s$ such that:
(1) $\lfloor a' \rceil = a$, $\lfloor b' \rceil = b$, and
(2) for all $c \in [a', b']$, $\lfloor c \rceil \in \{a, b\}$.

Note that a, b must be grid points. If $[a', b']$ is a maximal segment with this property, we call $[a', b']$ the "preimage" of $[a, b]$. Hence, each segment in $D_G(s)$ has at least one preimage in s. Below we will prove that these preimages are very well-behaved (in particular, there is a unique preimage for each segment in $D_G(s)$).

REMARK: the rest of this section has not been converted to the non-uniform case. The arguments here are from John Hobby.

A collection S of segments are **essentially disjoint** if any pair are disjoint or intersect at their end points only. We also write $\text{Snap}(S)$ for the set union of $\text{Snap}(s)$, ranging over all $s \in S$.

Lemma 2. *If S have endpoints in G and are essentially disjoint, then the set $\text{Snap}_G(S)$ are also essentially disjoint.*

The transformation is a rotation, *i.e.*, $\det M = 1$ and the inverse M^{-1} is given by its transpose M^T. E.g., if $\delta = 1$, the points $(\pm 1, 1)$ map to $(1/\sqrt{2}, 0)$ and $(0, 1/\sqrt{2})$.

Lemma 3. *The set of these preimages forms a cover of s. Each segment $D_T(s)$ has a unique preimage.*

Proof. The fact that they form a cover of s follows from the fact that if any maximal open subsegment $(a, b) \subseteq s$ that is dijoint from any preimages, then there must be a preimage of the form $[b, c] \subseteq s$ (for some c). This implies that $[a, c]$ is contained in a preimage, contradicting the assumption that $[b, c]$ is maximal. Uniqueness is trivial by the maximality requirement. **Q.E.D.**

Lemma 4. *If the endpoints in $D_T(S)$ maps to $q_1 = (\xi_1, \eta_1), \ldots, q_m = (\xi_m, \eta_m)$, and $\xi_1 < \xi_2 < \cdots < \xi_m$, then they describe a monotonic piecewise linear function F on the interval $[\xi_1, \xi_m]$ given by*

$$F(\xi) = \frac{\eta_i(\xi_{i+1} - \xi) + \eta_{i+1}(\xi - \xi_i)}{\xi_{i+1} - \xi_i}$$

when $\xi_i \le \xi \le \xi_{i+1}$.

Proof. This follows from the fact that the preimages of $[q_i, q_{i+1}]$ are ordered along the segment s in the expected way. [Incomplete] **Q.E.D.**

If $s' \in S$ is a distinct segment, then a similar function F' based on $D_T(s')$ can be defined.

Note that the slope of s' may be different from that of s. This will lead to different coordinate systems for F and F'. To fix this, we may first rotate the coordinate system so that s and s' have both positive slopes or both negative slopes. It is easy to see this is always possible.

These functions approximate the lines ℓ and ℓ' through s and s' (respectively).

Since s, s' are essentially disjoint, we may assume THAT IN THE (ξ, η)-coordinate system, wlog that s is below s' whenever they intersect a common vertical line $\xi = \xi_0$. So it suffices to show that

$$F(\xi) \le F'(\xi) \tag{4}$$

for all $\xi \in \{\xi_1, \ldots, \xi_m\} \cup \{\xi'_1, \ldots, \xi'_{m'}\}$.

Parametrize the lines ℓ so that $(\xi, \ell(\xi)) \in \ell$ for all ξ. Similarly, assume $(\xi, \ell'(\xi)) \in \ell'$. Since the segment in $D_T(s) \subseteq s_j + R_2$, and similar for s', the difference $F(\xi) - \ell(\xi)$ is limited to the range of the η coordinates in R_2. This ranges over the intervals $(-\frac{1}{2}, \frac{1}{2})$ or $[-\frac{1}{2}, \frac{1}{2})$. [CHECK]

Then our assumptions that $\ell(\xi) \le \ell'(\xi)$ for all ξ is the range of interest implies that

$$F'(\xi_i) \ge \eta_i, \qquad \text{if } \ \ell'(\xi_i) \ge \eta_i + \frac{1}{2}.$$

Otherwise, $\ell'(\xi_i) \in \eta_i + R_1$ and the definition of D_T forces $F'(\xi_i) \ge \eta_i$. incomplete

Similarly, we argue that $F(\xi'_j) \le \eta'_j$. This proves equation (4).

¶7. Snap Rounding Based on Bentley-Ottmann's Algorithm. For simplicity, we may assume that the input segments are already representable.

STEP 1. We modify the Bentley-Ottman algorithm as follows. We first run the original algorithm as a "first pass". The intersection points are symbolically represented (by the pair of defining segments). Let H be the set of all endpoints and the intersection points found in this way.

STEP 2. Compute the set of hot points: $\lfloor H \rceil$.

STEP 3. Now we want to snap each segment $s \in S$ to all the hot points p that it is near to. More precisely, for each s we compute a set of pairs $\{(q_i, p_i) : i = 1, \ldots, k\}$ (for some k) such that p_i is a hot point that s is near to, and q_i is the point in s closest to p_i. Then we snap q_i to p_i, in an natural order

determined by the q_i's along s. But how do we detect these p_i's? We can do this by modifying the same sweepline algorithm of Bentley-Ottman: for each hot point p, we add the four unit segments corresponding to the boundary of $p + \overline{R}_2$ to the input set. Thus the segment s is near to p iff it intersects one of these unit segments of p.

We can take advantage of the fact that these unit segments are horizontal or vertical and in special positions relative to the grid G_2. An efficient implementation (using the idea of batching, and keeping the unit segments separate from the original segments) is given in the original paper of Hobby.

Exercises

Exercise 2.1: Consider the Bentley-Ottmann's algorithm.
(i) Show examples where the algorithm will fail because of numerical errors.
(ii) Give details of the primitives in this algorithm.
(iii) Implement the algorithm. ♢

Exercise 2.2: Implement Hobby's hot snap rounding algorithm using the IEEE floating point grid. ♢

End Exercises

§3. Consistency Approach

A general approach to robust algorithms can be formulated around the concept of consistency. Consistency in its most general form is a purely logical concept; in the context of geometric algorithms, consistency is often called "topological consistency" [22]. The rationale for this is clear – topological properties are abstractions of the specific geometry that arises from specific numerical data. For instance, the Voronoi diagrams of a planar point set is combinatorially a planar graph. Then we *define* a Voronoi diagram algorithm to be "consistent" if its output a Voronoi diagram is a planar graph. But it is clear that consistency alone is not enough: imagine an algorithm that ignores the actual input but always output the correct answer to some pre-selected input. This algorithm is surely consistent. But how shall we require the algorithm to depend on the input? For instance, we would like the output to be not only self-consistent, but consistent for some ε-perturbation of the input. This is a form of backward-error analysis. We could proceed as follows. Fortune defines a **parsimonious program** as one that never makes a comparison if the outcome of that comparison can be logically deduced from the outcomes of previous comparisons. A parsimonious algorithm that ignores its input can simply toss a coin at each comparison and use this as the result of the comparison. To ensure that the algorithm depends on the input, we can require the algorithm to give the correct result to a comparison $x : y$ whenever $|x - y| > \varepsilon$. The attractiveness of this view is that it is a widely applicable paradigm, independent of the underlying geometry or topology.

In this section, we examine the ideas of Schorn [17] for consistent computation.

References

[1] G. Bohlender, C. Ullrich, J. W. von Gudenberg, and L. B. Rall. *Pascal-SC*, volume 17 of *Perspectives in Computing*. Academic Press, Boston-San Diego-New York, 1990.

[2] S. Figueroa. *A Rigorous Framework for Fully Supporting the IEEE Standard for Floating-Point Arithmetic in High-Level Programming Languages*. Ph.D. thesis, New York University, 1999.

[3] M. Goodrich, L. J. Guibas, J. Hershberger, and P. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th ACM Symp. on Comp. Geometry*, pages 284–293, 1997.

[4] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. *IEEE Foundations of Computer Sci.*, 27:143–152, 1986.

[5] L. Guibas and D. Marimont. Rounding arrangements dynamically. In *Proc. 11th ACM Symp. Computational Geom.*, pages 190–199, 1995.

[6] L. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. *ACM Symp. on Comp. Geometry*, 5:208–217, 1989.

[7] J. D. Hobby. Practical segment intersection with finite precision output. *Comput. Geometry: Theory and Appl.*, 13:199–214, 1999.

[8] Holt, Matthews, Rosselet, and Cordy. *The Turing Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[9] T. H. Hopp. Computational metrology. In *Proc. 1993 International Forum on Dimensional Tolerancing and Metrology*, pages 207–217, New York, NY, 1993. The American Society of Mechanical Engineers. CRTD-Vol.27.

[10] T. Hull, A. Abraham, M. Cohen, A. Curley, C. Hall, D. Penny, and J. Sawchuk. Numerical Turing. *ACM SIGNUM newsletter*, 20(3):26–34, July, 1985.

[11] T. Hull, M. Cohen, J. Sawchuk, and D. Wortman. Exception handling in scientific computing. *ACM Trans. on Math. Software*, 14(3):201–217, 1988.

[12] D. Jackson. Boundary representation modeling with local tolerances. In *Proc. Symp. on Solid Modeling Foundations and CAD/CAM applications*, pages 247–253, 1995.

[13] N. Juster. Modeling and representation of dimensions and tolerances: a survey. *CAD*, 24(1):3–17, 1992.

[14] K. Mehlhorn, T. Shermer, and C. Yap. A complete roundness classification procedure. In *13th ACM Symp. on Comp. Geometry*, pages 129–138, 1997.

[15] V. Milenkovic. Double precision geometry: a general technique for calculating line and segment intersections using rounded arithmetic. In *Proc. 30th Annual Symp. on Foundations of Comp.Sci.*, pages 500–506. IEEE Computer Society, 1989.

[16] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *Proc. 3rd ACM Sympos. Comput. Geom.*, pages 119–125, 1987.

[17] P. Schorn. An axiomatic approach to robust geometric programs. *J. of Symbolic Computation*, 16:155–165, 1993.

[18] M. G. Segal. Programming language support for geometric computations. Report csd-90-557, Dept. Elect. Engrg. Comput. Sci., Univ. California Berkeley, Berkeley, CA, 1990.

[19] M. G. Segal and C. H. Sequin. Consistent calculations for solids modelling. In *Proc. 1st ACM Sympos. Comput. Geom.*, pages 29–38, 1985.

[20] K. Sugihara. On finite-precision representations of geometric objects. *J. of Computer and System Sciences*, 39:236–247, 1989.

[21] K. Sugihara. An intersection algorithm based on Delaunay triangulation. *IEEE Computer Graphics Appl.*, 12(2):59–67, 1992.

[22] K. Sugihara and M. Iri. A solid modeling system free from topological inconsistency. *J.Information Processing, Information Processing Society of Japan*, 12(4):380–393, 1989.

[23] K. Sugihara and M. Iri. Two design principles of geometric algorithms in finite precision arithmetic. *Applied Mathematics Letters*, 2:203–206, 1989.

[24] K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-oriented implementation – an approach to robust geometric algorithms. *Algorithmica*, 27:5–20, 2000.

[25] H. B. Voelcker. A current perspective on tolerancing and metrology. In *Proc. 1993 International Forum on Dimensional Tolerancing and Metrology*, pages 49–60, New York, NY, 1993. The American Society of Mechanical Engineers. CRTD-Vol.27.

[26] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. Chapman & Hall/CRC, Boca Raton, FL, 2nd edition, 2004.

[27] C. K. Yap and E.-C. Chang. Issues in the metrology of geometric tolerancing. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robot Motion Planning and Manipulation*, pages 393–400, Wellesley, Massachusetts, 1997. A.K. Peters. 2nd Workshop on Algorithmic Foundations of Robotics (WAFR).