CS 164 & CS 266: Computational Geometry

Week 1

Lecture 1a: Coordinates and primitives

David Eppstein
University of California, Irvine

Spring Quarter, 2022
General Course Information

Instructor: David Eppstein
Assistants: Daniel Frishberg and Annie Raichev

Course info: Canvas and
https://www.ics.uci.edu/~eppstein/164

Weekly homeworks, midterm, and final exam
Area
A toy example

What is the area of this polygon?

Clever solution:
Cut along the grid lines
Flip the cut-off triangles into the lower left kite
The result exactly covers a square, area 1
Area of polygons

We want a solution without cleverness that a computer can follow

Verte coordinates:
\[\{(0, 0), (1, 2), (1, 1), (2, 1)\}\]

Area: 1

Input: Clockwise sequence of vertices
Each given by integer Cartesian coordinates

Output: A number, the area
Idea: decompose into simpler shapes

Triangles

\[
A = \frac{1}{2} \det \begin{pmatrix}
  x_1 & y_1 & 1 \\
  x_2 & y_2 & 1 \\
  x_3 & y_3 & 1
\end{pmatrix}
\]

Trapezoids

\[
A = \frac{1}{2} (x_2 - x_1)(y_1 + y_2)
\]
Using trapezoids to compute area

Add the areas of trapezoids below each upper edge of the polygon
Subtract the areas of trapezoids below each lower edge

Outside polygon, positive and negative areas cancel
Inside points all covered by one more positive than negative
Shoelace formula for area

\[
A = \sum_{i=0}^{n-1} \frac{1}{2} (x_{i+1} - x_i)(y_i + y_{i+1})
\]

All indexes computed modulo \( n \) so \((x_n, y_n) = (x_0, y_0)\)

Produces a positive number for positive trapezoids, negative for negative trapezoids
Shoelace formula as an algorithm

Summation ⇒ for-loop

```python
def shoelace(P):
    area = 0
    for i in 0, 1, 2, ... n-1:
        j = (i + 1) mod n
        xi,yi = P[i]
        xj,yj = P[j]
        area += (xj-xi)*(yj+yi)/2
    return area
```

Easy, time = $O(n)$
Geometric primitives
What is a primitive?

A constant-time formula/algorith/subroutine for higher-level information than the input coordinates

Example: The area of a triangle

Another example (same subroutine, different interpretation): If you travel from $p$ to $q$, then turn and travel from $q$ to $r$, which way did you turn?

- **right turn:** $\text{area}(p, q, r) > 0$
- **straight:** $\text{area}(p, q, r) = 0$
- **left turn:** $\text{area}(p, q, r) < 0$
Crossing test primitive

Idea: build more complicated primitives from simpler ones

Does line segment $ab$ cross line segment $cd$?

Yes, if: $abc$ turns the opposite way from $abd$, and $cda$ turns the opposite way from $cdb$
Distance from \((x_1, y_1)\) to \((x_2, y_2)\) (length of segment)?

\[
distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}
\]

(apply Pythagoras to right triangle with sides \(|x_1 - x_2|, |y_1 - y_2|\))
Coordinate systems
Motivation

Cartesian coordinates — \((x, y)\) pairs — are familiar and work well for many purposes.

Sometimes other coordinates are easier to use.

Example:

If we represent most lines as \((m, b)\) where \(y = mx + b\), finding the line through two points uses division, problematic if we want integers.

And what about vertical lines \(x = c\)?

Projective geometry eliminates both problems.
Coordinate systems for points in the plane

**Cartesian coordinates** \((x, y)\)
- Simple, familiar
- Generalize to higher dims
- Widely used, familiar

**Polar coordinates** \((r, \theta)\)
- Angle and distance from origin
- Widely known, not as useful

**Complex numbers**
- Built into some programming languages (Python)
- Make certain transformations easy:
  - Translate by \(t\): \(q \mapsto q + t\)
  - Scale by \(s\): \(q \mapsto qs\)
  - Rotate by \(\theta\): \(q \mapsto q(\cos \theta + i \sin \theta)\)
- Not as easy to generalize to higher dimensions
Formulas for coordinate conversion

**Cartesian to polar**

\[ r = \sqrt{x^2 + y^2} = \text{hypot}(x, y) \]
\[ \theta = \text{atan2}(y, x) \]

**Polar to Cartesian**

\[ x = r \cdot \cos \theta \]
\[ y = r \cdot \sin \theta \]

**Cartesian to complex**

\[ q = x + i \cdot y = x + 1j \cdot y \]

**Complex to Cartesian**

\[ x = \Re(q) = q.\text{real} \]
\[ y = \Im(q) = q.\text{imag} \]
Projective geometry
Equivalence between 2d points and 3d lines

Map plane into 3d by \((x, y) \mapsto (x, y, 1)\), view from \((0, 0, 0)\)

Would get same view for all scaled embeddings \((ax, ay, a)\)
All points \((ax, ay, a)\) on a line through \((0, 0, 0)\) look the same!
Projective geometry

New meaning for the words “point” and “line”:

- **“Point”:** 3d line through $(0, 0, 0)$
  
  Represent by 3d coordinates of any nonzero point on the line
  
  Many different triples all represent the same “point”:
  
  $(x, y, z) = (0.5x, 0.5y, 0.5z) = (3x, 3y, 3z) = \cdots$

- **“Line”:** 3d plane through $(0, 0, 0)$
  
  Contains all points and lines of the Euclidean plane
  
  $(x, y) \mapsto (x, y, 1) \quad (x, y, z) \mapsto (x/z, y/z)$

  But it also contains extra “points” $(x, y, z)$ with $z = 0$, called “points at infinity” (although $x, y, z$ are all finite numbers)
Lines in projective geometry

Line: set of points \((x, y, z)\) obeying an equation \(ax + by + cz = 0\)

Coordinates of the line: the numbers \(a, b, c\)

Many different triples all represent the same line:
\((a, b, c) = (0.5a, 0.5b, 0.5c) = (3a, 3b, 3c) = \cdots\)
Converting between coordinates

Euclidean to projective

Point \((x, y) \mapsto (x, y, 1)\)

Line \((m, b) \mapsto (m, -1, b)\)

\[y = mx + b \mapsto mx - y + bz = 0\]

Vertical line \(c \mapsto (1, 0, -c)\)

\[x = c \mapsto x + 0y - cz = 0\]

Projective to Euclidean

Point \((a, b, c) \mapsto (a/c, b/c)\)

Line \((a, b, c) \mapsto (-a/b, -c/b)\)

\[ax + by + cz = 0 \mapsto y = -(ax/b - c/b)\]

Vertical line \((a, 0, c) \mapsto -c/a\)

\[ax + cz = 0 \mapsto x = -c/a\]

But points at infinity \((x, y, 0)\) and line at infinity \((0, 0, 1)\) do not correspond to anything in Euclidean geometry!
What are these extra points?

We can think of:

- The plane as a surface that we’re viewing from slightly above it
- The line at infinity is the horizon
- The points at infinity are the **vanishing points** where parallel lines meet

CC-BY-SA image “Railroad in Northumberland County, Pennsylvania, southeast of Turbotville” by Jakec from https://commons.wikimedia.org/wiki/File:Railroad_in_Northumberland_County,_Pennsylvania.JPG
To test if point \((x, y, z)\) is on line \((a, b, c)\):
compute dot product \((x, y, z) \cdot (a, b, c)\), check if zero

Unaffected by multiplying the coordinates of either by a scalar

Unaffected by which triple is a “point” and which we call a “line”

So if we reinterpret all lines in projective geometry as being points,
and all points as being lines, it doesn’t affect any properties
defined using point–line intersection tests!

The space of points and lines formed by renaming the lines and
points of projective geometry in this way is the \text{projective dual}
Algorithmic applications of projective duality

Whenever we have a valid mathematical statement about points, lines, and point-line incidence in projective geometry, we can change all points to lines and all lines to points in the statement and get another valid statement.

Example 1: every two points have a line that touches both of them
Dual 1: every two lines have a point that touches both of them
Not true of Euclidean parallel lines!

Example 2: f is a subroutine that takes as input two points and outputs the line that touches them
Dual 2: f is a subroutine that takes as input two lines and outputs the point that touches them
The same subroutine does two different things!
To find line through two points / point on two lines

Math version:

Line through \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) is the set of points \((x, y, z)\) for which

\[
\det \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x & y & z \end{pmatrix} = 0
\]

Because this equation is linear in \(x, y, z\) and is true of the two given points

Computer science version:

```python
def thru2(p, q):
    x1, y1, z1 = p
    x2, y2, z2 = q
    x = y1*z2-y2*z1
    y = z1*x2-z2*x1
    z = x1*y2-x2*y1
    return (x, y, z)
```