Brezenham’s Algorithm (and variants)

- Consider only 1 octant (get others by symmetry)
  \(-0 \leq m \leq 1\)
- Loop over \(x\) pixels
  \(-\) Guarantees 1 per column
- For each pixel, either move up 1 or not
  \(-\) If you plotted \(x,y\) then choose either \(x+1,y\) or \(x+1,y+1\)
  \(-\) Trick: how to decide which one easily
  \(-\) Same method works for circles (just need different test)
- Decision variable
  \(-\) Implicit equation for line \((d=0\) means on the line\)

Midpoint method

\[\begin{align*}
\Delta d &= d_1 - d_2 \\
\Delta d &= (y-y_k) - (y_{k+1}-y) \\
y &= m(x_{k+1}) + b \\
\Delta d &= 2(m(x_{k+1}) + b) - 2y_k - 1 \\
m &= \frac{\Delta y}{\Delta x} \\
\end{align*}\]


Incremental Algorithm

- Suppose we know \(p_k\) – what is \(p_{k+1}\)?
  - \(p_{k+1} = p_k + 2\Delta y - 2\Delta x (y_{k+1} - y_k)\)
    \(-\) Since \(x_{k+1} = x_k + 1\)
  - And \(y_{k+1} - y_k\) is either 1 or 0, depending on \(p_k\)

Brezenham’s Algorithm

\[\begin{align*}
P_{k+1} &= 2\Delta y + x \\
Y &= y_1 \\
\text{For } X = x_1 \text{ to } x_2 \\
\& \text{ Set } X,Y \\
\& \text{ If } P_k > 0 \\
\& \quad Y += 1 \\
\& \quad P_k += 2 \Delta y \quad 2 \Delta x \\
\& \text{ Else: } P_k += 2 \Delta y \\
\end{align*}\]

Why is this cool?

- No division!
- No floating point!
- No gaps!
- Extends to circles
- But…
  - Jaggies
  - Lines get thinner as they approach 45 degrees
  - Can’t do thick primitives

Triangles?

- Old way: Scan conversion
  - Start at top
  - Brezenham’s algorithm gives left/right sides
  - Draw horizontal scans
- New Way: point in triangle tests
  - Generate sets of points that might be in triangle
  - Do half-plane tests to see if inside
- Tricky part: edges
  - Need to decide which triangle draws shared edges

General Polygons?

- Inside / Outside not obvious for general polygons
- Usually require simple polygons
  - Convex (easy to break into triangles)
- For general case, three common rules:
  - Non-exterior rule: A point is inside if every ray to infinity intersects the polygon
  - Non-zero winding number rule: trace around the polygon, count the number of times the point is circled (+1 for clockwise, -1 for counter clockwise). Odd winding counts = inside (note: I got this wrong in class)
  - Parity rule: Draw a ray to infinity and count the number or edges that cross it. If even, the point is outside, if odd, it’s inside (ray can’t go through a vertex)

Parity

- Any point, take any ray (that doesn’t go through a vertex)
- Odd number of crossings = inside
- Even number of crossings = outside

Power Point uses this rule!

Winding Numbers

- Count the number of times a point is circled counter clockwise
  - Clockwise counts negative
- Can pick any ray from point and count left/right
  - Right (relative to away direction) = CCW = +1
  - Left = CW = -1

Non-Zero Winding Rule

- Any non-zero winding is “inside”
- What Adobe Illustrator does
- Odd Winding Rule / Positive Winding Rule / ….
Inside/Outside Rules

Coordinate Systems

- Tells us how to interpret positions (coordinates)
- In graphics we deal with many coordinate systems and move between them
  - Use what is convenient for what we’re doing
- Examples
  - Chalkboard as coordinate system
  - One panel of chalkboard as coordinate system
  - Monitor as coordinate system

What is a coordinate system

- Position of the zero point
- Directions for each axis
  - Represent points as a linear combination of vectors
  - Vectors (basis) are axes
  - Scale of vectors matter (what is “1 unit”) 
  - Directions matter (which way is up)
  - Doesn’t need to be perpendicular (just can’t be parallel)

Describing Coordinate systems

- Need to have some “reference” 
  - Where we will measure from
- Give origin, vectors
- Once we have 1 system, can define others
- Can move points by changing their coordinate system
  - Piece of paper is a coordinate system
  - Move piece of paper around
  - If it were a rubber sheet could stretch it as well

Changing Coordinate Systems

- Changing coordinate systems allows us to change large numbers of points all at once
- Need to move points between coordinate systems
  - A coordinate system transforms points to a more canonical coordinate system
  - Can define coordinate systems by transformations between coordinate systems

Transformations

- Something that changes points
  - \( y' = f(x,y) \quad f \in \mathbb{R}^2 \rightarrow \mathbb{R}^2 \)
- Coordinate systems are a special case
- Other examples
  - \( F(x,y) = x+2, y+3 \)
  - \( F(x,y) = -y, x \)
  - \( F(x,y) = x^2, y \)
- Easy way to effect large numbers of points
Interpreting Transformations

- Can be viewed as a change of coordinates
  - What happens to a piece of graph paper?
  - Just sometimes to a stretchy piece of paper
- View as a function applied to points
- Function composition
  - \( F(g(h(x))) \) (note order)

\[ X \rightarrow h \rightarrow g \rightarrow f \rightarrow X' \]

Linear Transformations

- Important special case – linear functions
- Can be written as a matrix \( x' = M \times x \) (\( x \) is a vector)
- Good points
  - Many useful transformations are of this form
  - Composition by matrix multiply
  - Easy analysis
  - Straight lines stay straight lines
  - Inverses by inverting the matrix
- Note: linear operators preserve zero!

Example Linear Operators

- Uniform Scale
  \[ scale(s) = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \]
- Non-Uniform Scale
  \[ nonscale(s,t) = \begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \]
- Reflect
  \[ reflect(a,t) = \begin{bmatrix} -a & 0 \\ 0 & 1 \end{bmatrix} \]
- Skew
  \[ skew(a) = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \]

More linear operators

- Rotate
  \[ rotate(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \]
- Note: all of this keeps zero
- All linear operations are around the origin (?)

Understanding linear operators

\[ Mx = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \]

- This is POST-Multiply (vector on the right)
  - Pre-multiply convention works too
  - All the matrices get transposed
- What does each element do?
  - Left column – where does \( X \) axis go (put in unit \( X \) vector)
  - Right column – where does \( Y \) axis go
- Can’t do anything about origin!

Affine Transformations

- Translation = move all points the same (vector +)
- Affine = Linear operations plus translation
- Cannot be encoded in a 2x2 matrix (for 2d)
  - Need six numbers for 2d
  - Could be a 3x2 matrix – but then no more multiplies
- Rather than treat as a special case, improve our coordinates a bit
Homogeneous Coordinates

- Big idea for graphics – really important
  - Will be used for several things – translation is just 1
- Basic idea: add an extra coordinate
  - 2D becomes 3D (3x3 matrices)
  - 3D becomes 4D (4x4 matrices)
- Convert "back" from homogeneous coordinates by division
  - (x, y) -> (x, y, 1)
  - (x, y, w) -> (x/w, y/w)
- Projection
  - Many points in higher dim space = 1 point in lower dim space
- For now, just make w=1

Translation in Homogeneous Coords

- Translate in 2D = Skew in 3D
  - Deck of cards

\[
\begin{bmatrix}
1 & 0 & x \\
0 & 1 & y \\
0 & 0 & 1
\end{bmatrix}
\]

What about other linear ops

- Just add an extra coordinate
- Don’t change w (unless you know what you’re doing)

\[
\text{scale}(s) = \begin{bmatrix}
s & 0 & 0 \\
0 & s & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\text{rotate}(\theta) = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Matrices as Coordinate Systems

- Where does X axis go?
- Where does Y axis go?
- Where does origin go?

- Assumes that bottom row is [0 0 1]
- Can you scale by changing w?
  - Yes, but often we prefer to renormalize so bottom right number is 1