Lecture 2: 9 September 1997

Administrative:

Sign up (if you did not do so Thursday)
Reminder: on-line calendar (Assts, X-refs)
Asst 1, Fusion: **bear with us!**
- Only Assts 1, 2 are experimental
- Assts 3-5 based on C++, OpenInventor

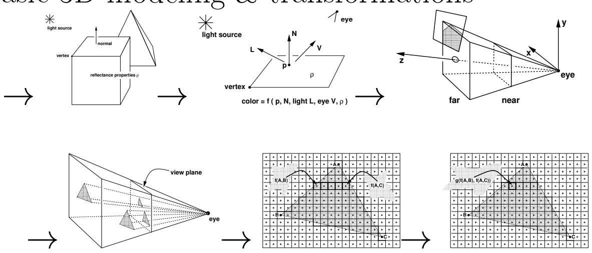
Linear Algebra review: W 7-9 in 34-101
Office Hours (Prof. T): Tuesday 4-on
TA Office Hours:
- David T 4-6
- Jon W 12-on, R 4-on
- Luke W 2-4

Today:

Overview of first part of course
2D Scan Conversion, cont. (H&B S3.11)
2D Clipping (H&B S6.5 – 6.7)
Asst 1 (Signup, Bresenham) due this Friday
Asst 2 (Clipping) due next Friday

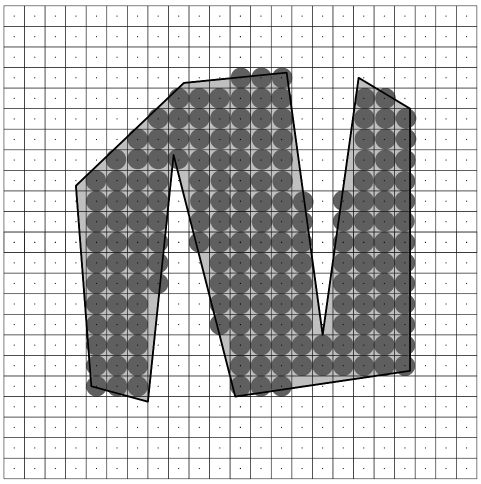
Early overview

At start: "bottom-up" coverage:
- Basic 2D rendering (lines, polygons, clipping)
- 2D Coordinate transformations
- Basic 3D modeling & transformations



Eventually, "top-down" view of rendering
- Classical rendering pipeline
- Alternative methods (e.g., ray-tracing)

Scan Converting Filled 2D Polygons

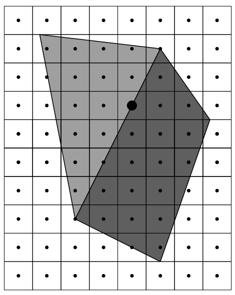


Today, consider both convex and concave polygons

Pixel Ownership

Each polygon should *own* certain pixels:
- don't want pixels owned by multiple polygons
- don't want to "drop" pixels
- polygon owns all pixels centered in poly interior
- polygon owns no pixels centered outside poly interior

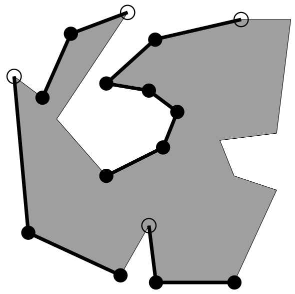
Issue: what about pixels centered on boundaries?



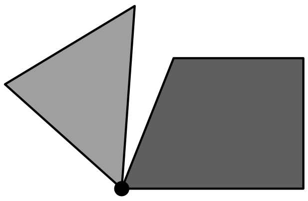
Center might be in two or more polygons
Why not write these pixels twice?

Shadow Rule

"Shadow" Rule: polygon owns boundary pixel unless:
  Edge is horizontal, and on "top" of poly
  Edge is right-facing
  Pixel at upper extremum of polygon

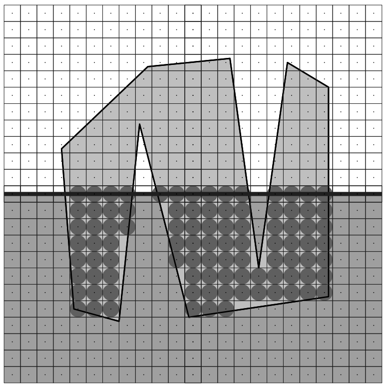


Still must handle special case of shared vertices!



A Scan Line Algorithm

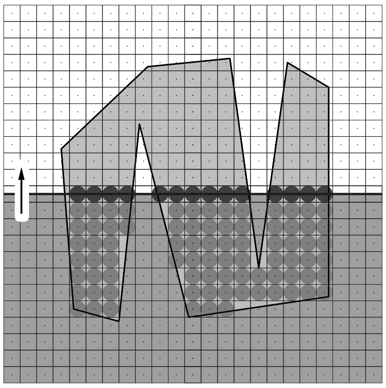
Sweep *horizontal* scan line from bottom to top



Scan line intersects polygon in horizontal *spans*
Span endpoints lie on polygon edges
Each span implicates some set of pixels to fill

Span Generation

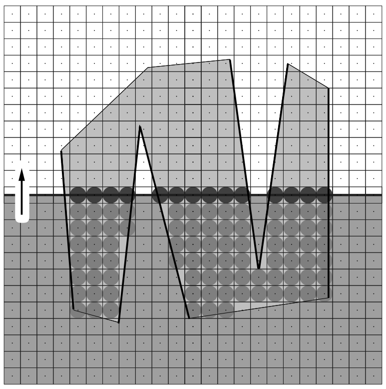
Idea: generate output spans once per scan line



Span coherence: spans include many pixels
Edge coherence: incremental edge updates
Careful to respect polygon "ownership" rules:
  Edge is active if intersected by scan line, except
  horizontal edges or edges intersected at top endpoint

Active Edge List

Maintain *Active Edge List* (AEL), sorted by x:
  those edges intersected by current scan line

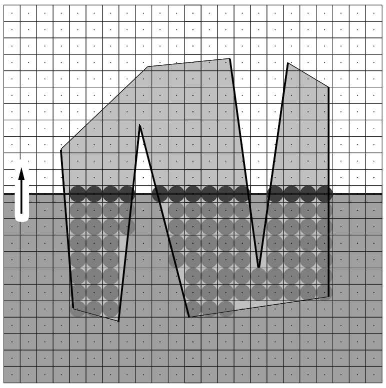


Output which spans? Traverse AEL, use **parity rule**:

- start with parity 0 (= outside polygon)
- reverse parity with each edge (1 = inside polygon)
- setting parity to 1 marks left endpoint of span
- setting parity to 0 marks right endpoint of span

Scan Conversion of Spans

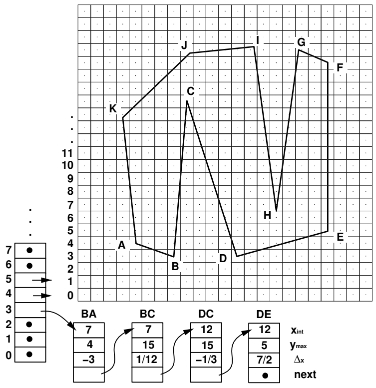
Simple, but careful of ownership, shadow rules:



- generate pixels left-to-right
- start with first integer at or past left endpoint
- end with last integer *before* right endpoint
- output single pixel if left == right at integer x

Initialization

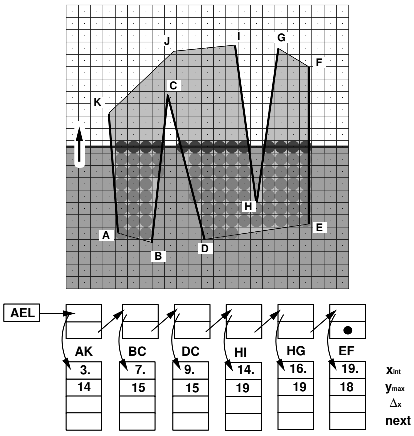
Precompute: Edge Table (ET), one entry per scan line



Each entry is a linked list of edges, sorted by x_{int}:
  x_{int}: x of (current) intersection with scan line
  y_{max}: y of top edge endpoint
  Δ_x: x increment with scan line advance
  next: pointer to next record, or NULL
At BOT, insert all non-horizontal edges
Edge buckets to first (lowest) intersecting scan line
Note: every edge represented exactly once!

Maintaining the AEL

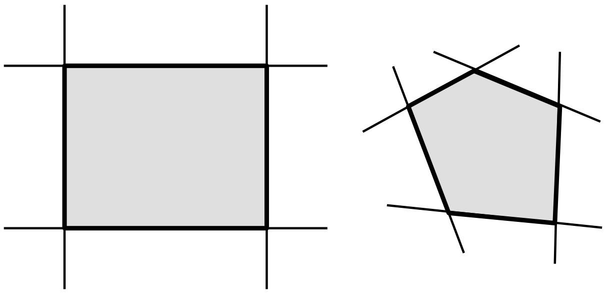
At each y value:
  1) Insert edges starting at this y
  2) Scan convert AEL to spans
  3) Then, **update** AEL:



   3a) Delete completed edges
   3b) Adjust active intersections
   3c) Re-sort AEL if necessary
Nice property: works for general polygons!

2D Clipping

So far, assumed all primitives within viewport
But, some vertices might be specified outside
Want geometry confined to *clip region*
  Today: clip region is convex polygon



Most frequent case: clip rectangle

$$x_{min} \leq x \leq x_{max}$$

$$y_{min} \leq y \leq y_{max}$$

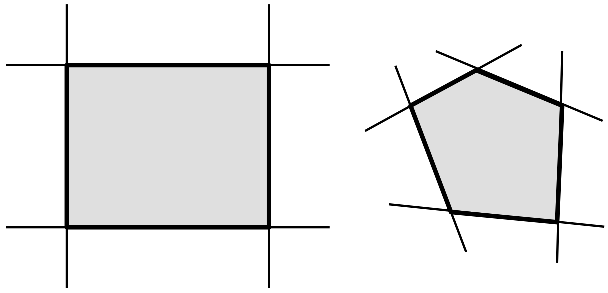
Clipping is the process of pruning geometric primitives to the clip region

Clipping Approaches

1) Scan convert elsewhere; copy bitmap to viewport
2) Scissor: clip on the fly during scan conversion
3) Clip analytically: revise input geometry
Today: analytical clipping of
  Points
  Segments
  Wireframe polygons
  Filled polygons

Point Clipping

Idea: retain point iff it is inside clip region



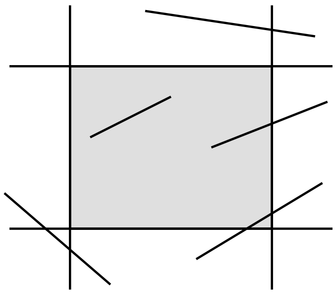
Clip region described by 4 inequalities:

$$x \geq x_{min}$$
$$x \leq x_{max}$$
$$y \geq y_{min}$$
$$y \leq y_{max}$$

Generalization to arbitrary convex regions?

Segment Clipping: Cases

Output must be a single segment, or vacuous (why?)



Both endpoints in clip region? *Trivial accept.*
One endpoint in clip region, one endpoint outside?
  Must exist intersection point p with edge of
    clip region; replace outside endpoint with p
Otherwise, both endpoints outside (two cases):
  No intersection with clip region
    Remove segment (output "clip out")
  Intersections with two edges of clip region
    Output segment between intersection points
    Handle degeneracies (e.g., corner crossing)

Brute-Force Clipping

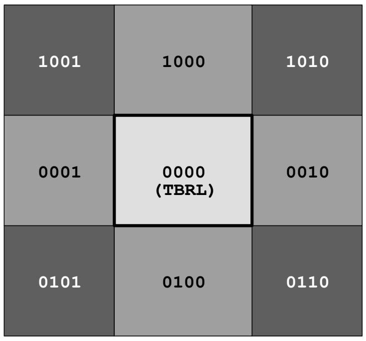
Classify endpoints to identify relevant case
Compute intersection with each clip region edge
  (Identify 0, 1, or 2 clipped points)
How to compute intersection points?
Solve parametric equations for segment ($0 \leq t \leq 1$):

$$x = x_0 + t(x_1 - x_0)$$
$$y = y_0 + t(y_1 - y_0)$$

Plug in to expressions for clip boundaries:
Example: $x = x_{min}$ yields $t = \boxed{\quad\quad}$
Check t to make sure it is inside $[0, 1]$
(Careful: what if segment parallel to boundary?)
Brute force algorithm is expensive

Cohen-Sutherland Algorithm

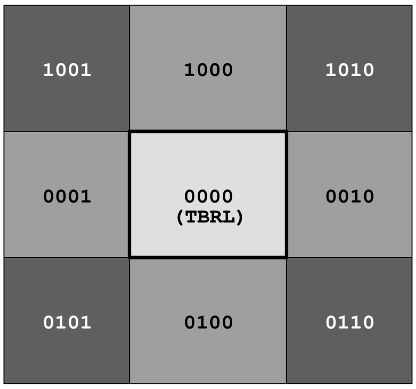
Reduces number of intersection computations
Works for any convex polygonal clip region
Observation: can often **trivially reject** segment (how?)



Strategy: classify each endpoint with respect
to T,B,R,L half-planes (1 bit per plane):
- $[y \geq y_{max}]$ (outside Top)
- $[y \leq y_{min}]$ (outside Bottom)
- $[x \geq x_{max}]$ (outside Right)
- $[x \leq x_{min}]$ (outside left)

Easy: each bit is sign bit of difference

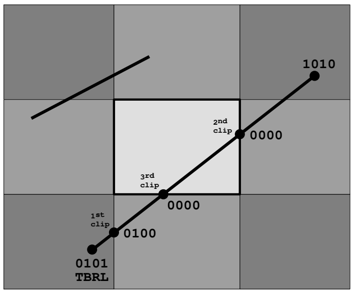
Cohen-Sutherland Algorithm



Concatenate TBRL bits into 4-bit "outcodes"
- Trivial accept iff both outcodes are zero
- Trivial reject iff some bit 1 for both endpoints
- Otherwise? What do we know about segment?

Cohen-Sutherland Algorithm

If neither trivial reject nor accept, then:
- There is at least one 1 bit
- Each 1 has corresponding 0 at other endpoint
  - Thus each 1 implies crossed boundary!
- Thus, if half-plane H's bit = 1 at endpoint A
  - A outside H, other endpoint inside H
  - Clip endpoint A to halfplane H, replace A



Idea: Clip segment in order Left, Right, Bottom, Top
- After each clip, recompute appropriate outcode
- Continue clipping until trivial accept/reject
  - (Must get one or the other, eventually)

Clipping Filled Polygons

Note: segment clipping algorithm directly useable
for clipping **wireframe** polygons
Will same algorithm work for filled polygons?