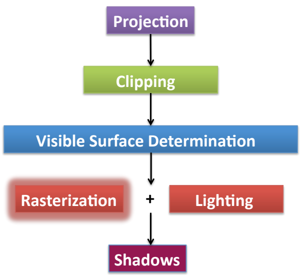
Rasterising Polygons

Rasterization Pipeline Reminder

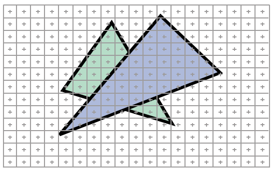


Overview

- What's Inside a Polygon
- Coherence
- Active Edge Tables
- Illumination Across Polygons

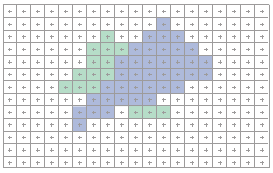
2D Scan Conversion

- Primitives are continuous; screen is discrete
  - Well, triangles are described by a discrete set of vertices
  - But they describe a continuous area on screen



2D Scan Conversion

- Solution: compute discrete approximation
- Scan Conversion (Rasterization): algorithms for efficient generation of the samples comprising this approximation



Naïve Filling Algorithm

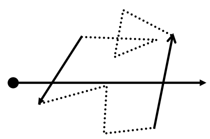
- Find a point inside the polygon
- Do a **flood fill:**
  - Keep a stack of points to be tested
  - When the stack none empty
    - Pop the top point (Q)
    - Test if Q is inside or outside
      - If Inside, colour Q, push neighbours of Q if not already tested
      - It outside discard
    - Mark Q as tested

Critique

- Horribly slow
  - Explicit in/out test at every point
  - But still very common in paint packages!
- Stack might be very deep
- Need to exploit **TWO** types of coherency
  - Point coherency
  - Scan-line coherency

Recall Infinite Ray Test

- Shoot infinite ray from point
- Count the number of intersections with the boundary

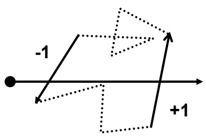


Counting Boundaries

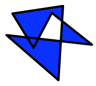
- If the shape is convex
  - Just count total number of intersections:
    - 0 or 2 (outside)
    - 1 (inside)
- Concave: Non-Zero Rule
  - Any number of intersections is possible, but if you just count the total you can not tell if you are inside or outside
    - Count +/-1 and use either the odd-even or non-zero rule

Infinite Ray Test - Rules

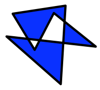
- Draw a line from the test point to the outside:
  - +1 if you cross anti-clockwise
  - -1 if you cross clockwise



Non-zero

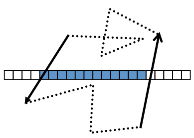


Odd-Even



Point Coherency

- Ray shooting is fast, but note that for every point on one scan line the intersection points are the same
- Why not find the actual span for each line from the intersection points?



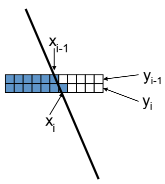
Scan-Line Coherency

- Intersection points of polygon edges with scan lines change little on a line by line basis

$$y_i = ax_i + b$$

$$y_{i-1} = ax_{i-1} + b$$

$$x_i = x_{i-1} + \frac{1}{a}$$



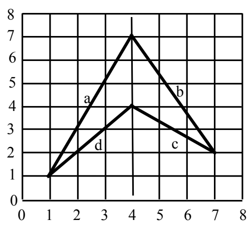
Overview of Active Edge Table

- For each scan-line in a polygon only certain edges need considering
- Keep an **ACTIVE** edge table
  - Update this edge table based upon the vertical extent of the edges
- From the AET extract the required spans

Setting Up

- “fix” edges
  - make sure y1<y2 for each (x1,y1) (x2,y2)
- Form an ET
  - Bucket sort all edges on minimum y value
  - 1 bucket might contain several edges
  - Each edge element contains
    - (max Y, start X, X increment)
    - X increment = (x2-x1)/(y2-y1)

Example



Setup

- Edges are

Edge Label	Coordinates	y1	Structure
a	(1,1) to (4,7)	1	(7,1, 0.5)
b	(7,2) to (4,7)	2	(7,7, -0.6)
c	(7,2) to (4,4)	2	(4,7, -1.5)
d	(1,1) to (4,4)	1	(4,1, 1)

- Edge Table Contains

Reminder: ET element = (max Y, start X, X increment)

y1	Sequence of Edges
1	(7,1,0.5), (4, 1, 1)
2	(7,7,-0.6), (4, 7,-1.5)

Maintaining the AET

- For each scan line
  - Remove all edges whose y2 is equal to current line
  - Update the x value for each remaining edge
  - Add all edges whose y1 is equal to current line

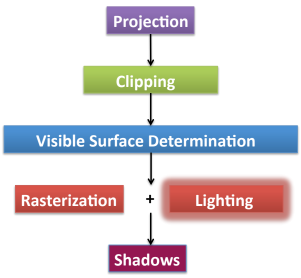
Drawing the AET

- Sort the active edges on x intersection
- Pairs of edges are the spans we require
- Caveats
  - Don't consider horizontal lines
  - Maximum vertices are not drawn
  - Plenty of special cases when polygons share edges

On Each Line

Line	Active Edge Table	Spans
0	empty	
1	(7,1,0.5), (4,1,1)	1 to 1
2	(7,1.5,0.5), (4,2,1), (4,7,-1.5), (7,7,-0.6)	1.5 to 2, 7 to 7
3	(7,2.0,0.5), (4,3,1), (4,5.5,-1.5), (7,6.4,-0.6)	2.0 to 3, 5.5 to 6.4
4	(7,2.5,0.5), (7,5.8,-0.6)	2.5 to 5.8
5	(7,3.0,0.5), (7,5.2,-0.6)	3.0 to 5.2
6	(7,3.5,0.5), (7,4.6,-0.6)	3.5 to 4.6
7	empty	
8	empty	

Rasterization Pipeline Reminder



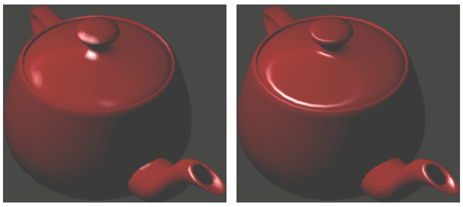
Gouraud Shading

- Recall simple model for local diffuse reflection

$$I = k_a I_a + k_d \sum_{i=1}^{N} I_{pi} \cdot (n \cdot l_i)$$

- Gouraud interpolates this colour down edges and across scan-lines (using barycentric combination)

Gouraud Shading Problems



Gouraud **Correct (Phong)**

Gouraud Details

- ET now contains
  - (y2, x1,dx, z1,dz, r1,dr, g1,dg, b1,db)
    - (we are running out of registers!)

$$dr = \frac{r_2 - r_1}{x_2 - x_1} \qquad dr = \frac{r_2 - r_1}{y_2 - y_1}$$

- Problems
  - not constant colour on rotation of points
  - misses specular highlights

Phong Shading

- Phong lighting model:

$$I = k_a I_a + \sum_{i=1}^{N} I_{pi} \cdot \left((n \cdot l_i) k_d + (h_i \cdot n)^m k_s \right)$$

  - Include specular component

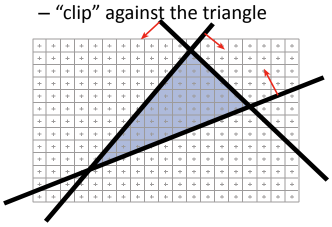
- Phong shading:
  - Interpolate normals across the scan-line instead of colours
  - Recaptures highlights in the centre of polygons

Is this really done in practise?

- Modern rasterisation works quite differently
- Reason:
  - GPU implementation of AET is very tricky
  - **Triangles** are a special case
    - Do not need generality of AET
- Start with a brute-force method and improve it…

Brute Force Solution for Triangles

- For *each* pixel
  - Compute line equations (half-space test) at pixel center
  - "clip" against the triangle



Half-Space Test Reminder

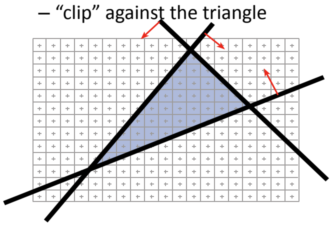
- For each edge compute line equation (analogue to plane equation):

$$L_i(x,y) = a_i x + b_i y + c_i$$

- If $L_i(x,y) > 0$
  - point in **positive** half-space
- If $L_i(x,y) < 0$
  - point in **negative** half-space
- If all $L_{1,2,3}(x,y) >= 0$
  - Point (x,y) is inside triangle!

Brute Force Solution for Triangles

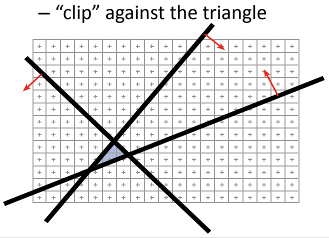
- For *each* pixel
  - Compute line equations at pixel center
  - "clip" against the triangle



Problem?

Brute Force Solution for Triangles

- For each pixel
  - Compute line equations at pixel center
  - "clip" against the triangle

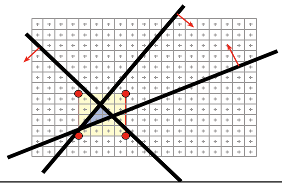


Problem?

If the triangle is small, a lot of useless computation

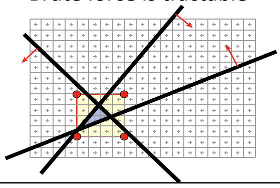
Brute Force Solution for Triangles

- Improvement: Compute only for the *screen bounding box* of the triangle
- How do we get such a bounding box?
  - Xmin, Xmax, Ymin, Ymax of the triangle vertices



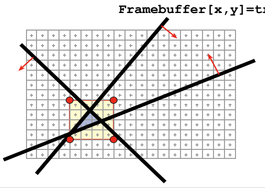
Rasterisation on Graphics Cards

- Triangles are usually very small
  – Setup cost are becoming more troublesome
- Clipping is annoying
- Brute force is tractable



Rasterisation on Graphics Cards

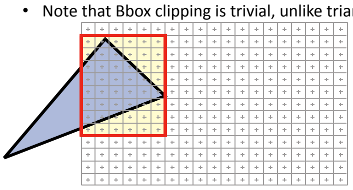
```
For every triangle
    ComputeProjection
    Compute bbox, clip bbox to screen limits
    For all pixels in bbox
        Compute line equations
        If all line equations>0  //pixel [x,y] in triangle
                Framebuffer[x,y]=triangleColor
```



Rasterisation on Graphics Cards

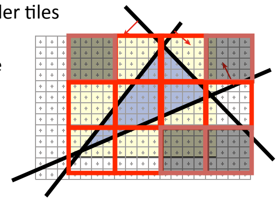
```
For every triangle
    ComputeProjection
    Compute bbox, clip bbox to screen limits
    For all pixels in bbox
        Compute line equations
        If all line equations>0  //pixel [x,y] in triangle
                Framebuffer[x,y]=triangleColor
```

- Note that Bbox clipping is trivial, unlike triangle clipping



Rasterisation on Graphics Cards

- Further tricks:
  - Compute result of line equation incrementally
    - Similar to AET
  - Subdivide BBox into smaller tiles
    - Early rejection of tiles
    - Memory access coherence



Recap

- Active Edge Table Method
  - Implements a scan-line based fill method
  - Exploits point and scan-line coherency
- AET easily extended to support Gouraud and Phong shading
  - (Also visibility, shadows and texture mapping)
- Modern Rasterisation
  - More brute-force, easier to implement in hardware

Exercise

- Given the following triangle, use the half-space-based triangle rasterization to compute whether pixel (2,2), (2,3), (3,2), and (3,3) are inside our outside of the triangle.