Polygon Clipping
- objective
  - 2D: clip polygon against rectangular window
  - or general convex polygons
  - extensions for non-convex or general polygons
  - 3D: clip polygon against parallelepiped

Why Is Clipping Hard?
- a really tough case:
  - concave polygon to multiple polygons

Polygons Clipping
- classes of polygons
  - triangles
  - convex
  - concave
  - holes and self-intersection

Sutherland-Hodgeman Clipping
- basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped

Project 3 update
- Linux executable reposted
- template update
  - download package again OR
  - just change line 31 of src/main.cpp from
  - int resolution[2];
  - to
  - int resolution[] = {100,100};
  - OR
  - implement resolution parsing

Review: Cohen-Sutherland Line Clipping
- outcodes
  - 4 flags encoding position of a point relative to top, bottom, left, and right boundary
  - \( OC(p1) == 0 \)
  - \( OC(p2) == 0 \)
  - trivial accept
  - \( (OC(p1) \& OC(p2)) != 0 \)
  - trivial reject
  - \( 0 \&\& \overline{OC(p2)} == 0 \)
  - trivial accept
  - \( 0 \&\& \overline{OC(p2)} == 0 \)
  - trivial reject

Review: Clipping Lines To Viewport
- combining trivial accepts/rejects
  - trivially accept lines with both endpoints inside all edges of the viewport
  - trivially reject lines with both endpoints outside the same edge of the viewport
  - otherwise, reduce to trivial cases by splitting into two segments

Sutherland-Hodgeman Clipping
- basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped

Why Is Clipping Hard?
- what happens to a triangle during clipping?
  - some possible outcomes:
    - triangle to triangle
    - triangle to quad
    - triangle to 5-gon
  - how many sides can result from a triangle?
  - seven

Clipping II
- objective
  - analytically calculating the portions of primitives within the viewport

Polygon Clipping
- not just clipping all boundary lines
- may have to introduce new line segments
Sutherland-Hodgeman Clipping
• basic idea:
  • consider each edge of the viewport individually
  • clip the polygon against the edge equation
  • after doing all edges, the polygon is fully clipped

Sutherland-Hodgeman Clipping
• basic idea:
  • consider each edge of the viewport individually
  • clip the polygon against the edge equation
  • after doing all edges, the polygon is fully clipped

Occlusion
• for most interesting scenes, some polygons overlap
• to render the correct image, we need to determine which polygons occlude which

Hidden Surface Removal

Analytic Visibility Algorithms
• early visibility algorithms computed the set of visible polygon fragments directly, then rendered the fragments to a display:

Painter's Algorithm
• simple: render the polygons from back to front, "painting over" previous polygons
  • draw blue, then green, then orange
  • will this work in the general case?

Painter's Algorithm: Problems
• intersecting polygons present a problem
  • even non-intersecting polygons can form a cycle with no valid visibility order:

Analytic Visibility Algorithms
• early visibility algorithms computed the set of visible polygon fragments directly, then rendered the fragments to a display:
Analytic Visibility Algorithms

• what is the minimum worst-case cost of computing the fragments for a scene composed of n polygons?
• answer: $O(n^2)$

Binary Space Partition Trees (1979)

• BSP Tree: partition space with binary tree of planes
• idea: divide space recursively into half-spaces by choosing splitting planes that separate objects in scene
• preprocessing: create binary tree of planes
• runtime: correctly traversing this tree enumerates objects from back to front

Creating BSP Trees: Objects

Splitting Objects

• no bunnies were harmed in previous example
• but what if a splitting plane passes through an object?
• split the object; give half to each node

Traversing BSP Trees

• tree creation independent of viewpoint
• preprocessing step
• tree traversal uses viewpoint
• runtime, happens for many different viewpoints
• each plane divides world into near and far
• for given viewpoint, decide which side is near and which is far
• check which side of plane viewpoint is on independently for each tree vertex
• tree traversal differs depending on viewpoint!
• recursive algorithm
• recurse on far side
• draw object
• recurse on near side

BSP Trees: Viewpoint A

query: given a viewpoint, produce an ordered list of (possibly split) objects from back to front:

renderBSP(BSPtree *T)
BSPtree *near, *far;
if (eye on left side of T->plane)
near = T->left; far = T->right;
else
near = T->right; far = T->left;
renderBSP(far);
if (T is a leaf node)
renderObject(T);
renderBSP(near);
BSP Trees: Viewpoint A

BSP Trees: Viewpoint A

BSP Trees: Viewpoint A

BSP Trees: Viewpoint A

BSP Trees: Viewpoint A

BSP Trees: Viewpoint A

BSP Trees: Viewpoint A

BSP Trees: Viewpoint A

BSP Trees: Viewpoint B

BSP Trees: Viewpoint B

BSP Tree Traversal: Polygons
• split along the plane defined by any polygon from scene
• classify all polygons into positive or negative half-space of the plane
  • if a polygon intersects plane, split polygon into two and classify them both
  • recurse down the negative half-space
  • recurse down the positive half-space

BSP Demo
• useful demo: http://symbolcraft.com/graphics/bsp

Summary: BSP Trees
• pros:
  • simple, elegant scheme
  • correct version of painter’s algorithm back-to-front rendering approach
  • was very popular for video games (but getting less so)
• cons:
  • slow to construct tree: \(O(n \log n)\) to split, sort
  • splitting increases polygon count: \(O(n^2)\) worst-case
  • computationally intense preprocessing stage restricts algorithm to static scenes