Cohen-Sutherland Line Clipping

- **outcodes**: 4 flags encoding position of a point relative to top, bottom, left, and right boundary
  - $OC(p1)=0010$
  - $OC(p2)=0000$
  - $OC(p3)=1001$

- **issues**
  - generation of rays
  - intersection of rays with geometric primitives
  - geometric transformations
  - lighting and shading
  - efficient data structures so we don’t have to test intersection with every object

**Correction: Blinn-Phong Model**

- variation with better physical interpretation
  - Jim Blinn, 1977
  - $h$: halfway vector
  - $h$ must also be explicitly normalized: $h/|h|
  - highlight occurs when $h$ near $n$

**Review: Radiosity**

- capture indirect diffuse-diffuse light exchange
- model light transport as flow with conservation of energy until convergence
  - view-independent, calculate for whole scene then browse from any viewpoint
  - divide surfaces into small patches
  - loop: check for light exchange between all pairs
    - form factor: orientation of one patch vs other patch ($n \times n$ matrix)

**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

**Review: Non-Photorealistic Rendering**

- simulate look of hand-drawn sketches or paintings, using digital models
- www.red3d.com/cwr/npr/

**Review: Non-Photorealistic Shading**

- cool-to-warm shading: $k_s = \frac{l \cdot n}{(l \cdot n)} + (1 - k_s)\cdot c$
- draw silhouettes: if $(e \cdot n_e)(e \cdot n_s) < 0$
- draw creases: if $(e_u \cdot e_v)$ is threshold

**News**

- midterms returned, solutions out
- unscaled average 52, scaled average 62

**P1 Hall of Fame: Honorable Mentions**

- Pierre Jordeau
- Shawn Luo
- David Roodnick

**P1 Hall of Fame: Winner**

- Sung-Hoo Kim

**Correction: Blinn-Phong Model**

- variation with better physical interpretation
  - Jim Blinn, 1977
  - $h$: halfway vector
  - $h$ must also be explicitly normalized: $h/|h|
  - highlight occurs when $h$ near $n$

**Review: Line Clipping**

- line segment: $(p_1, p_2)$
- trivial cases
  - $OC(p1) &= 0 \& OC(p2) = 0$
  - both points inside window, thus line segment completely visible (trivial accept)
  - $(OC(p1) \& OC(p2)) = 0$
    - there is (at least) one boundary for which both points are outside (same flag set in both outcodes)
    - thus line segment completely outside window (trivial reject)

**Review: Ray Tracing**

- capture indirect diffuse-diffuse light exchange
- model light transport as flow with conservation of energy until convergence
  - view-independent, calculate for whole scene then browse from any viewpoint
  - divide surfaces into small patches
  - loop: check for light exchange between all pairs
    - form factor: orientation of one patch vs other patch ($n \times n$ matrix)

**Review: Subsurface Scattering**

- light enters and leaves at different locations on the surface
- bounces around inside
- technical Academy Award, 2003
  - Jensen, Marschner, Hanrahan

**Review: Clipping**

- analytically calculating the portions of primitives within the viewport

**Review: Radiosity**

- capture indirect diffuse-diffuse light exchange
- model light transport as flow with conservation of energy until convergence
  - view-independent, calculate for whole scene then browse from any viewpoint
  - divide surfaces into small patches
  - loop: check for light exchange between all pairs
    - form factor: orientation of one patch vs other patch ($n \times n$ matrix)
Cohen-Sutherland Line Clipping
- intersect line with edge

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

Viewport Intersection Code
- \((x_1, y_1), (x_2, y_2)\) intersect vertical edge at \(x_{right}\)
  - \(y_{intersect} = y_1 + (x_{right} - x_1) \cdot \frac{y_2 - y_1}{x_2 - x_1}\)
  - \(m = \frac{y_2 - y_1}{x_2 - x_1}\)

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

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

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
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

Clipping Against One Edge

- \( p[i] \) inside: 2 cases
  - inside
  - outside

- \( p[i-1] \)

output: \( p[i] \)

- \( p[i] \)

output: nothing

Clipping Against One Edge

- \( p[i] \) outside: 2 cases
  - inside
  - outside

- \( p[i-1] \)

output: \( p[i] \)

- \( p[i] \)

output: \( p, p[i] \)

- \( p[i-1] \)

ClipPolygonToEdge( \( p[n] \), edge ) {
  for( i = 0 ; i < n ; i++ ) {
    if( \( p[i] \) inside edge ) {
      if( \( p[i-1] \) inside edge ) {
        output \( p[i] \);
      } else {
        p = intersect( \( p[i-1] \), \( p[i] \), edge );
        output p, \( p[i] \);
      }
    } else {
      if( \( p[i] \) is outside edge ) {
        p = intersect( \( p[i] \), \( p[i-1] \), edge );
        output p;
      }
    }
  }
}

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?

Analytic Visibility Algorithms

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

Sutherland-Hodgeman Discussion

- similar to Cohen/Sutherland line clipping
- inside/outside tests: outcodes
- intersection of line segment with edge: window-edge coordinates
- clipping against individual edges independent
- great for hardware (pipelining)
- all vertices required in memory at same time
- not so good, but unavoidable
- another reason for using triangles only in hardware rendering

Hidden Surface Removal

- to render the correct image, we need to determine which polygons obscure which

Occlusion

- for most interesting scenes, some polygons overlap

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?

Analytic Visibility Algorithms

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

Sutherland-Hodgeman Example

- input/output for whole algorithm
- input: list of polygon vertices in order
- output: list of clipped polygon vertices consisting of old vertices (maybe) and new vertices (maybe)
- input/output for each step
- input: list of vertices
- output: list of vertices, possibly with changes
- basic routine
  - go around polygon one vertex at a time
  - decide what to do based on 4 possibilities
  - is vertex inside or outside?
  - is previous vertex inside or outside?

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

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?
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

- 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

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 B

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

Clarification: BSP Demo

• order of insertion can affect half-plane extent