Clipping

Week 8, Wed Mar 7

Reading for This Time

- FCG Chap 12 Graphics Pipeline
  - only 12.1-12.4
News

- Project 3 out
- Homework 3 out
  - both due Mon 19 March
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 wrt other patch (n x 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: 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: \( k_w = \frac{1 + \mathbf{n} \cdot \mathbf{l}}{2}, c = k_w c_w + (1 - k_w) c_c \)
- draw silhouettes: if \((\mathbf{e} \cdot \mathbf{n}_0)(\mathbf{e} \cdot \mathbf{n}_1) \leq 0\), \(\mathbf{e}\)=edge-eye vector
- draw creases: if \((\mathbf{n}_0 \cdot \mathbf{n}_1) \leq \text{threshold}\)

\[ k_w = 1 + \mathbf{n} \cdot \mathbf{l} \]
\[ c = k_w c_w + (1 - k_w) c_c \]

Review: Image-Based Modelling / Rendering

- store and access only pixels
  - no geometry, no light simulation, ...
  - input: set of images
  - output: image from new viewpoint
    - surprisingly large set of possible new viewpoints
- display time not tied to scene complexity
  - expensive rendering or real photographs
- convergence of graphics, vision, photography
  - computational photography
Clipping
Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer
Next Topic: Clipping

• we’ve been assuming that all primitives (lines, triangles, polygons) lie entirely within the viewport
  • in general, this assumption will not hold:
Clipping

• analytically calculating the portions of primitives within the viewport
Why Clip?

• bad idea to rasterize outside of framebuffer bounds
• also, don’t waste time scan converting pixels outside window
  • could be billions of pixels for very close objects!
Line Clipping

• 2D
  • determine portion of line inside an axis-aligned rectangle (screen or window)

• 3D
  • determine portion of line inside axis-aligned parallelepiped (viewing frustum in NDC)
  • simple extension to 2D algorithms
Clipping

• naïve approach to clipping lines:
  for each line segment
    for each edge of viewport
      find intersection point
      pick “nearest” point
    if anything is left, draw it

• what do we mean by “nearest”?  
• how can we optimize this?
Trivial Accepts

• big optimization: trivial accept/rejects
  • Q: how can we quickly determine whether a line segment is entirely inside the viewport?
  • A: test both endpoints
Trivial Rejects

- Q: how can we know a line is outside viewport?
- A: if both endpoints on wrong side of same edge, can trivially reject line
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
Cohen-Sutherland Line Clipping

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

• assign outcode to each vertex of line to test
  • line segment: \((p_1, p_2)\)

• trivial cases
  • \(\text{OC}(p_1) == 0 \&\& \text{OC}(p_2) == 0\)
    • both points inside window, thus line segment completely visible
      (trivial accept)
  • \((\text{OC}(p_1) \& \text{OC}(p_2)) != 0\)
    • 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)
Cohen-Sutherland Line Clipping

- if line cannot be trivially accepted or rejected, subdivide so that one or both segments can be discarded
- pick an edge that the line crosses (*how?*)
- intersect line with edge (*how?*)
- discard portion on wrong side of edge and assign outcode to new vertex
- apply trivial accept/reject tests; repeat if necessary
Cohen-Sutherland Line Clipping

• if line cannot be trivially accepted or rejected, subdivide so that one or both segments can be discarded
• pick an edge that the line crosses
  • check against edges in same order each time
    • for example: top, bottom, right, left
Cohen-Sutherland Line Clipping

- intersect line with edge
Cohen-Sutherland Line Clipping

- discard portion on wrong side of edge and assign outcode to new vertex

- apply trivial accept/reject tests and repeat if necessary
Viewport Intersection Code

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

- \((x_1, y_1), (x_2, y_2)\) intersect horizontal edge at \(y_{\text{bottom}}\)
  - \(x_{\text{intersect}} = x_1 + \frac{(y_{\text{bottom}} - y_1)}{m}\)
  - \(m = \frac{y_2 - y_1}{x_2 - x_1}\)
Cohen-Sutherland Discussion

• key concepts
  • use opcodes to quickly eliminate/include lines
    • best algorithm when trivial accepts/rejects are common
  • must compute viewport clipping of remaining lines
    • non-trivial clipping cost
    • redundant clipping of some lines
• basic idea, more efficient algorithms exist
Line Clipping in 3D

• approach
  • clip against parallelepiped in NDC
    • after perspective transform
  • means that clipping volume always the same
    • xmin=ymin= -1, xmax=ymax= 1 in OpenGL

• boundary lines become boundary planes
  • but outcodes still work the same way
  • additional front and back clipping plane
    • zmin = -1, zmax = 1 in OpenGL
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
Polygon Clipping

- not just clipping all boundary lines
- may have to introduce new line segments
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
Why Is Clipping Hard?

• a really tough case:

concave polygon to multiple polygons
Polygon 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
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
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
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 Algorithm

• 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?
Clipping Against One Edge

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

- For \( p[i] \) inside:
  - **Inside:** \( p[i] \)
  - **Outside:** \( p[i-1] \)
  - output: \( p[i] \)

- For \( p[i] \) outside:
  - **Inside:** \( p[i] \)
  - **Outside:** \( p, p[i-1] \)
  - output: \( p, p[i] \)
Clipping Against One Edge

- $p[i]$ outside: 2 cases

output: $p$  

output: nothing
Clipping Against One Edge

```c
clipPolygonToEdge( p[n], edge ) {
    for( i= 0 ; i< n ; i++ ) {
        if( p[i] inside edge ) {
            if( p[i-1] inside edge ) output p[i];   // p[-1]= p[n-1]
            else {
                p= intersect( p[i-1], p[i], edge ); output p, p[i];
            }
        } else {                                     // p[i] is outside edge
            if( p[i-1] inside edge ) {
                p= intersect(p[i-1], p[l], edge ); output p;
            }
        }
    }
}
```
Sutherland-Hodgeman Example
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