Viewing 4 of 4: Culling and Clipping
Lab 1b: Flash Intro

William H. Hsu
Department of Computing and Information Sciences, KSU

Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Today: Sections 2.3.5, 2.4, 3.1.3, Eberly 2e – see http://bit.ly/ieUq45
Next class: Sections 2.4, 2.5, 3.1.6, Eberly 2e
Brown CS123 slides on Clipping – http://bit.ly/eWU7i1
Lecture Outline

- Reading for Last Class: Sections 2.5.1, 3.1 Eberly 2e
- Reading for Today: §2.3.5, 2.4, 3.1.3, Eberly 2e
- Reading for Next Class: §2.4, 2.5 (Especially 2.5.4), 3.1.6, Eberly 2e
- Last Time: Scan Conversion (aka Rasterization) of Lines
  - Incremental algorithm
  - Bresenham’s algorithm & midpoint line algorithm
  - Preview: Circles and Ellipses (Lecture 8)
- Today: Intro to Clipping and Culling
  - Clipping
    - 2-D derivation: clip edges
    - Algorithms: Cohen-Sutherland, Liang-Barsky/Cyrus-Beck
    - 3-D derivation: clip faces
  - Culling
    - Back face culling
    - Occlusion culling
# Where We Are

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Primary Source(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Course Overview</td>
<td>Chapter 1, Eberly 2nd</td>
</tr>
<tr>
<td>1</td>
<td>CG Basics: Transformation Matrices; Lab 0</td>
<td>Sections (§) 2.1, 2.2</td>
</tr>
<tr>
<td>2</td>
<td>Viewing 1: Overview, Projections</td>
<td>§2.3 – 2.4, 2.8</td>
</tr>
<tr>
<td>3</td>
<td>Viewing 2: Viewing Transformation</td>
<td>§2.3 esp. 2.3.4; FVH slides</td>
</tr>
<tr>
<td>4</td>
<td>Lab 1a: Flash &amp; OpenGL Basics</td>
<td>Ch. 2, 1.2, Angel Primer</td>
</tr>
<tr>
<td>5</td>
<td>Viewing 3: Graphics Pipeline</td>
<td>§2.3 esp. 2.3.7, 2.6, 2.7</td>
</tr>
<tr>
<td>6</td>
<td>Scan Conversion 1: Lines, Midpoint Algorithm</td>
<td>§2.5.1, 3.1, FVH slides</td>
</tr>
<tr>
<td>7</td>
<td>Viewing 4: Clipping &amp; Culling; Lab 1b</td>
<td>§2.3.5, 2.4, 3.1.3</td>
</tr>
<tr>
<td>8</td>
<td>Scan Conversion 2: Polygons, Clipping Intro</td>
<td>§2.4, 2.6 esp. 2.3.4, 3.1.6</td>
</tr>
<tr>
<td>9</td>
<td>Surface Detail 1: Illumination &amp; Shading</td>
<td>§2.5, 2.6.1 – 2.6.2, 4.3.1.2, 20.2</td>
</tr>
<tr>
<td>10</td>
<td>Lab 2a: Direct3D / DirectX Intro</td>
<td>§2.7, Direct3D handout</td>
</tr>
<tr>
<td>11</td>
<td>Surface Detail 2: Textures, OpenGL Shading</td>
<td>§2.6.3, 20.3 – 20.4, Primer</td>
</tr>
<tr>
<td>12</td>
<td>Surface Detail 3: Mappings, OpenGL Textures</td>
<td>§20.5 – 20.13</td>
</tr>
<tr>
<td>13</td>
<td>Surface Detail 4: Pixel/Vertex Shad.; Lab 2b</td>
<td>§3.1</td>
</tr>
<tr>
<td>14</td>
<td>Surface Detail 5: Direct3D Shading, OGLSL</td>
<td>§3.2 – 3.4, Direct3D handout</td>
</tr>
<tr>
<td>15</td>
<td>Demos 1: CGA, Fun; Scene Graphs: State</td>
<td>§4.1 – 4.3, CGA handout</td>
</tr>
<tr>
<td>16</td>
<td>Lab 3a: Shading &amp; Transparency</td>
<td>§2.6, 20.1, Primer</td>
</tr>
<tr>
<td>17</td>
<td>Animation 1: Basics, Keyframes; HW/Exam</td>
<td>§§5.1 – 5.2</td>
</tr>
<tr>
<td>18</td>
<td>Exam 1 review: Hour Exam 1 (evening)</td>
<td>Chapters 1 – 4, 20</td>
</tr>
<tr>
<td>19</td>
<td>Scene Graphs: Rendering; Lab 3b: Shader</td>
<td>§4.4 – 4.7</td>
</tr>
<tr>
<td>20</td>
<td>Demos 2: SFV: Skinning, Morphing</td>
<td>§§5.3 – 5.8, CGA handout</td>
</tr>
<tr>
<td>21</td>
<td>Demos 3: Surfaces, B-reps/Volume Graphics</td>
<td>§§10.4, 12.7, Mesh handout</td>
</tr>
</tbody>
</table>

Lightly-shaded entries denote the due date of a written problem set, heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review, and the green-shaded entry, that of the term project.

Green, blue and red letters denote exam review, exam, and exam solution review dates.
Clipping

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Line Clipping

- Clipping endpoints
  - \( x_{\text{min}} < x < x_{\text{max}} \) \( \text{and} \) \( y_{\text{min}} < y < y_{\text{max}} \) \( \rightarrow \) point inside

- Endpoint analysis for lines:
  - if both endpoints in, do "trivial acceptance"
  - if one endpoint inside, one outside, must clip
  - if both endpoints out, don't know

- Brute force clip: solve simultaneous equations using \( y = mx + b \) for line and four clip edges
  - slope-intercept formula handles infinite lines only
  - doesn't handle vertical lines

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Parametric Line Formulation For Clipping

Parametric form for line segment

\[ X = x_0 + t(x_1 - x_0) \quad 0 \leq t \leq 1 \]
\[ Y = y_0 + t(y_1 - y_0) \]
\[ P(t) = P_0 + t(P_1 - P_0) \]

“true,” i.e., interior intersection, if \( s_{\text{edge}} \) and \( t_{\text{line}} \) in \([0, 1]\)

(hard to compute)

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Cohen-Sutherland 2-D Clipping:
Outcodes [1]

- Divide plane into 9 regions
- Compute the sign bit of 4 comparisons between a vertex and an edge
  - $y_{max} - y; y - y_{min}; x_{max} - x; x - x_{min}$
  - point lies inside only if all four sign bits are 0, otherwise exceeds edge

- 4 bit outcode records results of four bounds tests:
  - First bit: outside halfplane of top edge, above top edge
  - Second bit: outside halfplane of bottom bottom edge
  - Third bit: outside halfplane of right edge, to edge, below right of right edge
  - Fourth bit: outside halfplane of left edge, to left of left edge

- Compute outcodes for both vertices of each edge (denoted $OC_0$ and $OC_1$)
- Lines with $OC_0 = 0$ and $OC_1 = 0$ can be trivially accepted (i.e., outcode 0000)
- Lines lying entirely in a half plane outside an edge can be trivially rejected: $OC_0$ AND $OC_1 \neq 0$ (i.e., they share an “outside” bit)

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Cohen-Sutherland 2-D Clipping: Outcodes [2]

- Very similar to 2D
- Divide volume into 27 regions (Picture a Rubik’s cube)
- 6-bit outcode records results of 6 bounds tests

<table>
<thead>
<tr>
<th>Back plane</th>
<th>Front plane</th>
<th>Top plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000 (in front)</td>
<td>010000 (in front)</td>
<td>001000 (above)</td>
</tr>
<tr>
<td>100000 (behind)</td>
<td>000000 (behind)</td>
<td>000000 (below)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bottom plane</th>
<th>Right plane</th>
<th>Left plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000 (above)</td>
<td>000000 (to left of)</td>
<td>0000001 (to left of)</td>
</tr>
<tr>
<td>000100 (below)</td>
<td>000010 (to right of)</td>
<td>000000 (to right of)</td>
</tr>
</tbody>
</table>

- First bit: outside back plane, behind back plane
- Second bit: outside front plane, in front of front plane
- Third bit: outside top plane, above top plane
- Fourth bit: outside bottom plane, below bottom plane
- Fifth bit: outside right plane, to right of right plane
- Sixth bit: outside left plane, to left of left plane

- Again, Lines with $OC_0 = 0$ and $OC_1 = 0$ can be trivially accepted
- Lines lying entirely in a volume on outside of a plane can be trivially rejected: $OC_0 \ AND \ OC_1 \neq 0$ (i.e., they share an “outside” bit)

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Cohen-Sutherland Algorithm [1]

- If we can neither trivially accept/reject (T/A, T/R), divide and conquer
- Subdivide line into two segments; then T/A or T/R one or both segments:

- use a clip edge to cut line
- use outcodes to choose edge that is crossed
  - edges where the two outcodes differ at that particular bit are crossed
- pick an order for checking edges: top – bottom – right – left
- compute the intersection point
  - the clip edge fixes either x or y
  - can substitute into the line equation
- iterate for the newly shortened line, “extra” clips may happen (e.g., E-I at H)

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Cohen-Sutherland Algorithm [2]

- \( y = y_0 + \text{slope} \times (x - x_0) \) and \( x = x_0 + (1/\text{slope}) \times (y - y_0) \)

- Algorithm:

```plaintext
ComputeOutCode(x0, y0, outcode0);
ComputeOutCode(x1, y1, outcode1);
repeat
  check for trivial reject or trivial accept
  pick the point that is outside the clip rectangle
  if TOP then
    x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0); y = ymax;
  else if BOTTOM then
    x = x0 + (x1 - x0) * (ymin - y0)/(y1 - y0); y = ymin;
  else if RIGHT then
    y = y0 + (y1 - y0) * (xmax - x0)/(x1 - x0); x = xmax;
  else if LEFT then
    y = y0 + (y1 - y0) * (xmin - x0)/(x1 - x0); x = xmin;
  if (x0, y0 is the outer point) then
    x0 = x; y0 = y; ComputeOutCode(x0, y0, outcode0)
  else
    x1 = x; y1 = y; ComputeOutCode(x1, y1, outcode1)
until done
```

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Scan Conversion after Clipping

- Don’t round and then scan convert, because the line will have the wrong slope: calculate decision variable based on pixel chosen on left edge
  - (remember: $y = mx + B$)
  - $x = x_{min}$
  - $(x_{min}, \text{Round}(mx_{min} + B))$
  - $(x_{min}, mx_{min} + B)$

- Horizontal edge problem:
  - clipping/rounding produces pixel $A$; to get pixel $B$, round up $x$ of the intersection of line with $y = y_{min} - \frac{1}{2}$ and pick pixel above:

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Sutherland-Hodgman Polygon Clipping

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Cyrus-Beck / Liang-Barsky

Parametric Line Clipping [1]

- Use parametric line formulation: \( P(t) = P_0 + (P_1 - P_0)t \)
- Determine where line intersects the infinite line formed by each clip rectangle edge
  - solve for \( t \) multiple times depending on the number of clip edges crossed
  - decide which of these intersections actually occur on the rectangle

Outside of clip region

Inside of clip rectangle

\[ N_i \cdot |P(t) - P_{E_i}| > 0 \]

\[ N_i \cdot |P(t) - P_{E_i}| < 0 \]

For any point \( P_{E_i} \) on edge \( E_i \)

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Parametric Line Clipping [2]

- Now solve for the value of $t$ at the intersection of $P_0P_1$ with the edge $E_i$:
  
  $N_i \bullet [P(t) - P_{E_i}] = 0$

- First, substitute for $P(t)$:
  
  $N_i \bullet [P_0 + (P_1 - P_0)t - P_{E_i}] = 0$

- Next, group terms and distribute dot product:
  
  $N_i \bullet [P_0 - P_{E_i}] + N_i \bullet [P_1 - P_0]t = 0$

- Let $D$ be the vector from $P_0$ to $P_1 = (P_1 - P_0)$, and solve for $t$:
  
  $t = \frac{N_i \bullet [P_0 - P_{E_i}]}{-N_i \bullet D}$

- Note that this gives a valid value of $t$ only if the denominator of the expression is nonzero.

- For this to be true, it must be the case that:
  - $N_i \neq 0$ (that is, the normal should not be 0; this could occur only as a mistake)
  - $D \neq 0$ (that is, $P_1 \neq P_0$)
  - $N_i \bullet D \neq 0$ (edge $E_i$ and line $D$ are not parallel; if they are, no intersection).

- The algorithm checks these conditions.

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Cyrus-Beck / Liang-Barsky
Parametric Line Clipping [3]

- Eliminate $t$'s outside $[0,1]$ on the line
- Which remaining $t$'s produce interior intersections?
- Can't just take the innermost $t$ values!

- Move from $P_0$ to $P_1$; for a given edge, just before crossing:
  - If $N_i \cdot D < 0 \iff$ Potentially Entering (PE), if $N_i \cdot D > 0 \iff$ Potentially Leaving (PL)
  - Pick inner PE, PL pair: $t_E$ for $P_{PE}$ with max $t$, $t_L$ for $P_{PL}$ with min $t$, and $t_E > 0$, $t_L < 1$.
  - If $t_L < t_E$, no intersection

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Cyrus-Beck / Liang-Barsky
Line Clipping Algorithm

Pre-calculate $N_i$ and select $P_{E_i}$ for each edge;
for each line segment to be clipped
if $P_L = P_0$ then line is degenerate so clip as a point;
else
begin
  $t_0 = 0$; $t_1 = 1$;
  for each candidate intersection with a clip edge
  if $N_i \cdot D = 0$ then (Ignore edges parallel to line)
  begin
    calculate $t_1$ (of line and clip edge intersection)
    use sign of $N_i \cdot D$ to categorize as PE or PL;
    if PE then $t_0 = \max(t_0, t_1)$;
    if PL then $t_1 = \min(t_0, t_1)$;
  end
  if $t_0 > t_1$ then return nil
  else return $P(t_0)$ and $P(t_1)$ as true clip intersections
end

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
**Parametric Line Clipping**  
**For Upright Clip Rectangle [1]**

- \( D = P_1 - P_0 = (x_1 - x_0, y_1 - y_0) \)
- Leave \( P_{E_i} \) as an arbitrary point on clip edge; it's a free variable and drops out

**Calculations for Parametric Line Clipping Algorithm**

<table>
<thead>
<tr>
<th>Clip Edge</th>
<th>Normal ( N_i )</th>
<th>( P_{E_i} )</th>
<th>( P_0 - P_{E_i} )</th>
<th>( t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>left: ( x = x_{\text{min}} )</td>
<td>((-1,0))</td>
<td>((x_{\text{min}}, y))</td>
<td>((x_0 - x_{\text{min}}y_0, y))</td>
<td>(-\frac{x_0 - x_{\text{min}}}{y_0 - y_{\text{min}}})</td>
</tr>
<tr>
<td>right: ( x = x_{\text{max}} )</td>
<td>((1,0))</td>
<td>((x_{\text{max}}, y))</td>
<td>((x_0 - x_{\text{max}}y_0, y))</td>
<td>(-\frac{x_0 - x_{\text{max}}}{y_0 - y_{\text{max}}})</td>
</tr>
<tr>
<td>bottom: ( y = y_{\text{min}} )</td>
<td>((0,-1))</td>
<td>((x, y_{\text{min}}))</td>
<td>((x_0 - xy_0, y_{\text{min}}))</td>
<td>(-\frac{y_0 - y_{\text{min}}}{x_0 - x_{\text{min}}})</td>
</tr>
<tr>
<td>top: ( y = y_{\text{max}} )</td>
<td>((0,1))</td>
<td>((x, y_{\text{max}}))</td>
<td>((x_0 - xy_0, y_{\text{max}}))</td>
<td>(-\frac{y_0 - y_{\text{max}}}{x_0 - x_{\text{max}}})</td>
</tr>
</tbody>
</table>

Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University  
Parametric Line Clipping
For Upright Clip Rectangle [2]

- Examine $t$:
  - numerator is just the directed distance to an edge; sign corresponds to OC
  - denominator is just the horizontal or vertical projection of the line, $dx$ or $dy$; sign determines PE or PL for a given edge
  - ratio is constant of proportionality: “how far over” from $P_0$ to $P_1$ intersection is relative to $dx$ or $dy$
Culling: A Form of Visible Surface Determination

- Given a set of 3-D objects and a view specification (camera), determine which lines or surfaces of the object are visible
  - why might objects not be visible?
    - occlusion vs. clipping
  - clipping is one object at a time, while occlusion is global

- Also called
  Hidden Surface Removal (HSR)

- We begin with some history of previously used VSD algorithms

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Visibility Culling:
View Frustum, Back Face, Occlusion

View frustum culling

Back face culling

Occlusion culling

View Point

© 1998 – 2004 Kim et al., KAIST VR Lab
http://bit.ly/e3wRRN
Occlusion Culling

Without occlusion culling:
- Player's camera
- LOS
- Hidden blocks rendered, CPU time wasted

With occlusion culling:
- Hidden blocks not rendered, CPU time saved

© 2010 Kwoon, J. (Jakkor), Roblox.com

http://bit.ly/edQi9N
Lab 1B

- **Adobe Flash**
  - Basic 2-D (up to Flash v9)
  - 3-D: Flash 10+
  - Simple Flash animation exercise

- **Animation Ideas**
  - **Animate**: to “bring to life”
  - From still frames to animations
  - Incremental change and smoothness

- **Using Culling**
  - Back faces illustrated
  - What to do besides cull

- **Simple Flash Animation Exercise**
  - Turn in
    - ActionScript source code
    - Screenshot(s) as instructed in Lab 1 handout
Summary

- Last Time: Scan Conversion (aka Rasterization)
  - Lines: incremental algorithm vs. (Bresenham’s) midpoint algorithm
  - Decision variables and forward differences
  - Circles and Ellipses (preview)
- See Also: CG Basics 3 - 4
  - CG Basics 3: Projections and 3-D Viewing (in detail)
  - CG Basics 4: Fixed-Function Graphics Pipeline
- Today: Clipping and Culling
  - What parts of scene to clip: edges vs. polygons of model
  - What parts of viewport to clip against: clip faces vs. clip edges
  - Clipping techniques
    - Cohen-Sutherland: outcodes (quick rejection), test intersections
    - Liang-Barsky / Cyrus-Beck: solve for $t$, find innermost PE/PL
  - Visibility culling: view frustum, back face, occlusion
- Next: More Scan Conversion (Polygons, Scan Line Interpolation)
Terminology

- **Fixed Function Pipeline**
  - Modelview transformation
  - Normalizing transformation (inverse of viewing transformation)

- **Coordinate Spaces**
  - Model space – absolute w.r.t. model
  - World space aka scene space – absolute w.r.t. scene, canonical
  - Camera / Eye / View space – relative, user-defined, arbitrary
  - Clip space – before perspective division
  - Normalized device coordinates – after perspective division

- **Clipping and Culling**
  - Clip faces/edges – clip region (screen, view volume) boundaries
  - Clipping techniques
    - Cohen-Sutherland: outcodes (quick rejection), test intersections
    - Liang-Barsky / Cyrus-Beck: solve for $t$, innermost PE/PL
  - Visibility culling: view frustum, back face, occlusion