Lecture 7 of 41

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

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

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
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
Wayback Machine archive of Brown CS123 slides: http://bit.ly/gAhJbh

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

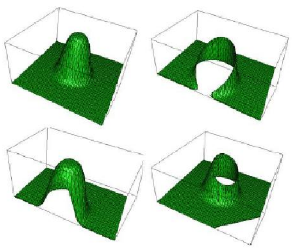
Where We Are

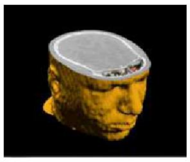
Lecture	Topic	Primary Source(s)
0	Course Overview	Chapter 1, Eberly 2e
1	**CG Basics: Transformation Matrices; Lab 0**	**Sections (§) 2.1, 2.2**
2	Viewing 1: Overview, Projections	§ 2.2.3 – 2.2.4, 2.8
3	Viewing 2: Viewing Transformation	§ 2.3 esp. 2.3.4; FVFH slides
4	**Lab 1a: Flash & OpenGL Basics**	**Ch. 2, 16[1], Angel *Primer***
5	Viewing 3: Graphics Pipeline	§ 2.3 esp. 2.3.7; 2.6, 2.7
6	Scan Conversion 1: Lines, Midpoint Algorithm	§ 2.5.1, 3.1; FVFH slides
7	**Viewing 4: Clipping & Culling; Lab 1b**	**§ 2.3.5, 2.4, 3.1.3**
8	Scan Conversion 2: Polygons, Clipping Intro	§ 2.4, 2.5 esp. 2.5.4, 3.1.6
9	Surface Detail 1: Illumination & Shading	§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2
10	**Lab 2a: Direct3D / DirectX Intro**	**§ 2.7, Direct3D handout**
11	Surface Detail 2: Textures; OpenGL Shading	§ 2.6.3, 20.3 – 20.4, *Primer*
12	Surface Detail 3: Mappings; OpenGL Textures	§ 20.5 – 20.13
13	**Surface Detail 4: Pixel/Vertex Shad.; Lab 2b**	**§ 3.1**
14	Surface Detail 5: Direct3D Shading; OGLSL	§ 3.2 – 3.4, Direct3D handout
15	Demos 1: CGA, Fun; Scene Graphs: State	§ 4.1 – 4.3, **CGA handout**
16	**Lab 3a: Shading & Transparency**	**§ 2.6, 20.1, *Primer***
17	**Animation 1: Basics, Keyframes; HW/Exam**	**§ 5.1 – 5.2**
	Exam 1 review; Hour Exam 1 (evening)	**Chapters 1 – 4, 20**
18	**Scene Graphs: Rendering; Lab 3b: Shader**	**§ 4.4 – 4.7**
19	**Demos 2: SFX; Skinning, Morphing**	**§ 5.3 – 5.5, CGA handout**
20	Demos 3: Surfaces; B-reps/Volume Graphics	§ 10.4, 12.7, **Mesh handout**

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

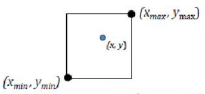




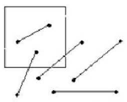
Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

Line Clipping

- Clipping endpoints



  - $x_{min} < x < x_{max}$ and $y_{min} < y < y_{max}$ ⇒ 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



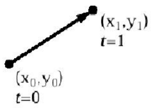
- Brute force clip: solve simultaneous equations using $y = mx + b$ for line and four clip edges
  - slope-intercept formula handles infinite lines only
  - doesn't handle vertical lines

Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

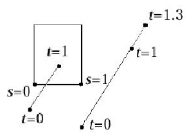
Parametric Line Formulation For Clipping

- Parametric form for line segment

$$X = x_0 + t(x_1 - x_0) \quad 0 \le t \le 1$$
$$Y = y_0 + t(y_1 - y_0)$$
$$P(t) = P_0 + t(P_1 - P_0)$$



- "true," i.e., interior intersection, if s_{edge} and t_{line} in [0,1]
  - (hard to compute)

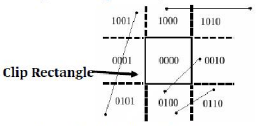


Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

7

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
  - ymax – y; y – ymin; xmax – x; x - xmin
  - 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*: OC0 AND OC1 ≠ 0 (i.e., they share an "outside" bit)

8

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

Back plane	Front plane	Top plane
000000 (in front)	010000 (in front)	001000 (above)
100000 (behind)	000000 (behind)	000000 (below)

Bottom plane	Right plane	Left plane
000000 (above)	000000 (to left of)	000001 (to left of)
000100 (below)	000010 (to right of)	000000 (to right of)

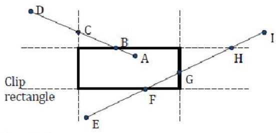
First bit: outside back plane, behind back plane
Second bit: outside front plane, in front of front plane
Third bit: outside top plane, above top plane
Fourth bit: outside bottom plane, below bottom plane
Fifth bit: outside right plane, to right of right plane
Sixth bit: outside left plane, to left of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

9

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)

10

Cohen-Sutherland Algorithm [2]

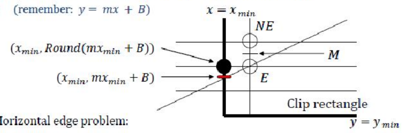
- y = y0 + slope*(x - x0) and x = x0 + (1/slope)*(y - y0)
- **Algorithm:**

```
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
```

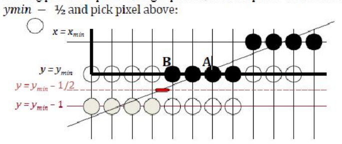

11

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: $y = mx + B$)

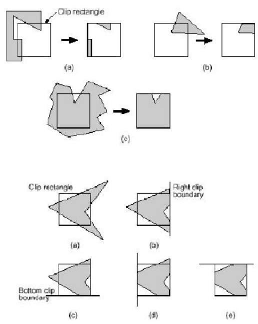


- Horizontal edge problem:
- clipping/rounding produces pixel A; to get pixel B, round up x of the intersection of line with $y = ymin - ½$ and pick pixel above:



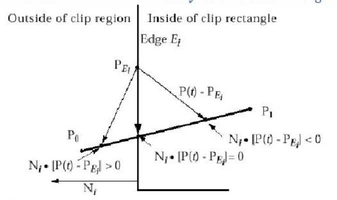
12

Sutherland-Hodgman Polygon Clipping



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



- For any point P_{E_i} on edge E_i

Cyrus-Beck / Liang-Barsky Parametric Line Clipping [2]

- Now solve for the value of *t* at the intersection of $P_0 P_1$ 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:

$$N_i \bullet [P_0 - P_{E_i}] + N_i \bullet [P_1 - P_0]t = 0$$

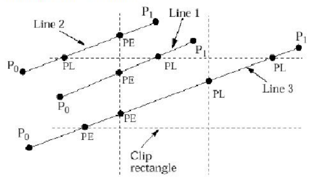
- Let *D* be the vector from P_0 to $P_1 = (P_1 - P_0)$, and solve for *t*:

$$t = \frac{N_i \bullet [P_0 - P_{E_i}]}{-N_i \bullet D}$$

  - 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.

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 $N_i \bullet D < 0 \implies$ Potentially Entering (PE), if $N_i \bullet D > 0 \implies$ 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

Cyrus-Beck / Liang-Barsky Line Clipping Algorithm

```
Pre-calculate N_i and select P_Ei for each edge;
for each line segment to be clipped
 if P_1 = P_0 then line is degenerate so clip as a point;
 else
  begin
   t_E = 0; t_L = 1;
   for each candidate intersection with a  clip edge
    if Ni • D ≠ 0 then {Ignore edges parallel to line}
     begin
      calculate t; {of line and clip edge intersection}
      use sign of N_i • D to categorize as PE or PL;
      if PE then t_E = max(t_E,t);
      if PL then t_L = min(t_L,t);
     end
   if t_E > t_L then return  nil
   else return P(t_E) and P(t_L) as true clip intersections
 end
```


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

Clip Edge$_i$	Normal N_i	P_{E_i}	P_0-P_{E_i}	$t = \frac{N_i \bullet (P_0 - P_{E_i})}{-N_i \bullet D}$
left: $x = x_{min}$	(-1,0)	(x_{min}, y)	$(x_0 - x_{min}, y_0 - y)$	$\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$
right: $x = x_{max}$	(1,0)	(x_{max}, y)	$(x_0 - x_{max}, y_0 - y)$	$\frac{-(x_0 - x_{max})}{(x_1 - x_0)}$
bottom: $y = y_{min}$	(0,-1)	(x, y_{min})	$(x_0 - x, y_0 - y_{min})$	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
top: $y = y_{max}$	(0,1)	(x, y_{max})	$(x_0 - x, y_0 - y_{max})$	$\frac{-(y_0 - y_{max})}{(y_1 - y_0)}$

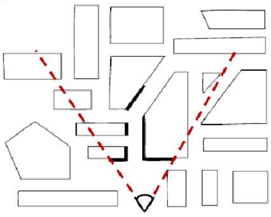
Parametric Line Clipping For Upright Clip Rectangle [2]

- 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***

19

Culling: A Form of Visible Surface Determination

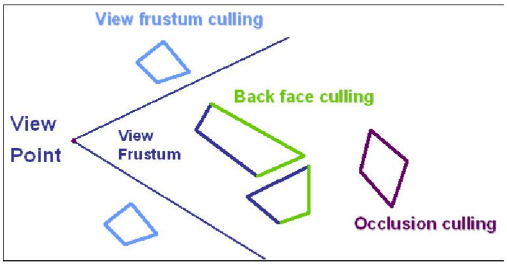
- Given a set of 3-D objects and a view specification (camera), determine which lines or surfaces of the object are visible
  - why might objects not be visible? *occlusion vs. clipping*
  - clipping is one object at a time, while *occlusion is global*
- Also called Hidden Surface Removal (HSR)
- We begin with some history of previously used VSD algorithms



Adapted from slides © 1997 – 2010 van Dam *et al.*, Brown University
http://bit.ly/hiSt0f Reused with permission.

20

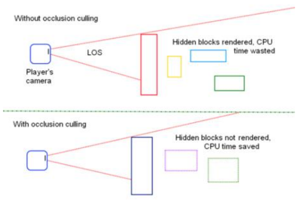
Visibility Culling: View Frustum, Back Face, Occlusion



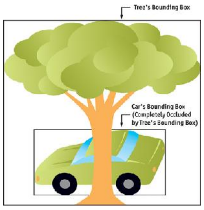
© 1998 – 2004 Kim *et al.*, KAIST VR Lab
http://bit.ly/e3wRRN

21

Occlusion Culling



© 2010 Kwoon, J. (Jakkor), Roblox.com
http://bit.ly/hAL7U5



© 2004 Sekulic, D. Chapter 29: Efficient Occlusion Culling. In Fernando, R., *ed., GPU Gems.* Reading, MA: Addison-Wesley.
http://bit.ly/edQt9N

22

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**
  - **Watch Senocular.com tutorial(s) as needed (http://bit.ly/hhlgtk)**
  - **Turn in**
    - **ActionScript source code**
    - **Screenshot(s) as instructed in Lab 1 handout**

23

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)**

24

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**