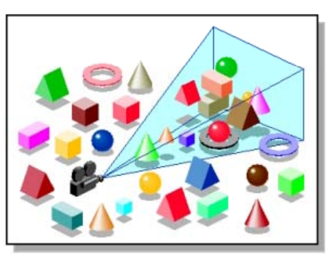
CS 362: Computer Graphics

Clipping

Dr. Samit Bhattacharya
Dept. of Comp. Sc. & Engg.
IIT Guwahati, Assam, India

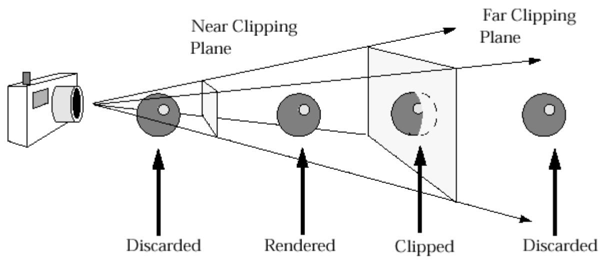
Discard Objects

- Discarding objects which are totally outside view volume
  - Involves comparing an object's bounding box/sphere against the dimensions of the view volume



Clip Objects

- Objects that are partially within the viewing volume need to be clipped



3D Clipping

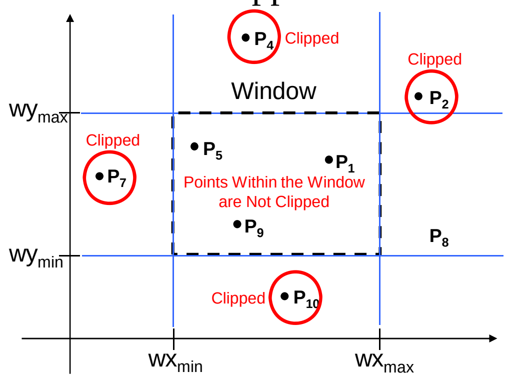
- Many of the algorithms are extension of clipping in 2D
- We will first have a look into the 2D clipping algorithms for
  - Point
  - Line
  - Polygon fill area
- Then discuss about the 3D extensions

Point Clipping

- Easy - a point (x,y) is not clipped if:

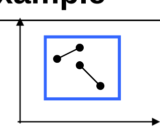
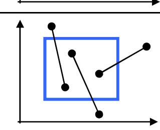
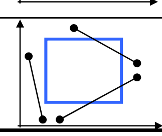
$$wx_{min} \le x \le wx_{max} \text{ AND } wy_{min} \le y \le wy_{max}$$

- Otherwise it is clipped



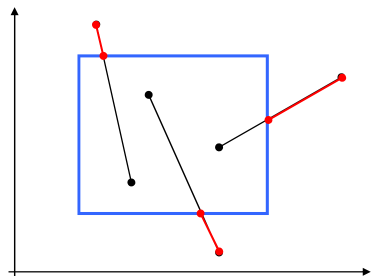
Line Clipping

- Harder - examine the end-points of each line to see if they are in the window or not

Situation	**Solution**	**Example**
Both end-points inside the window	Don’t clip	
One end-point inside the window, one outside	Must clip	
Both end-points outside the window	Don’t know!	

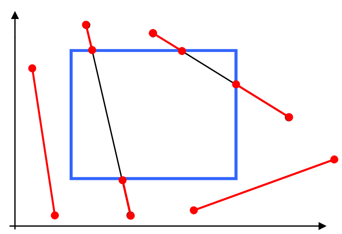
Brute Force Line Clipping

- Brute force line clipping can be performed as follows:
  - Don't clip lines with both end-points within the window
  - For lines with one end-point inside the window and one end-point outside, calculate the intersection point (using the equation of the line) and clip from this point out



Brute Force Line Clipping

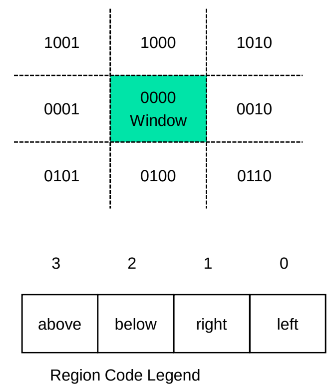
  - For lines with both end-points outside the window, test the line for intersection with all of the window boundaries, and clip appropriately



- Calculating line intersections is computationally expensive
  - Because a scene can contain so many lines, the brute force approach to clipping is much too slow

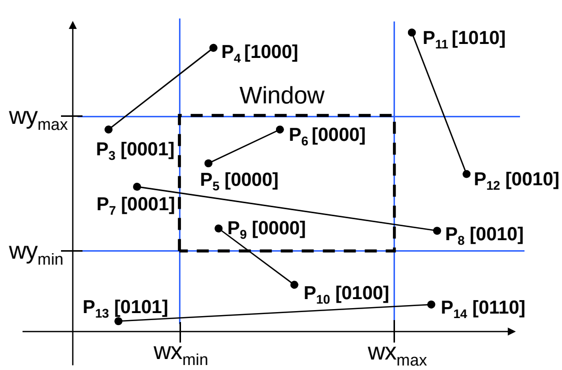
Cohen-Sutherland Clipping Algorithm

- An efficient line clipping algorithm
- Key advantage: vastly reduces the number of line intersection calculation
- World space is divided into regions based on the window boundaries
  - Each region has a unique four bit region code
  - Region codes indicate the position of the regions with respect to the window



Cohen-Sutherland: Labeling

- Every end-point is labeled with the appropriate region code



Region Code Assignment

- Bit 3 = sign (y-y_max)
- Bit 2 = sign (y_min-y)
- Bit 1 = sign (x-x_max)
- Bit 0 = sign (x_min-x)

- Sign(a)=1 if a is positive, 0 otherwise

Cohen-Sutherland: Steps

- Both endpoint region code 0000, line is completely inside window – retain it
- Logical AND of both endpoints ≠ 0000, line is completely outside – discard it entirely

Cohen-Sutherland: Steps

- For all other cases, do the following
  - Calculate line intersection point with window boundaries (follow some order for checking, e.g. Left, Right, Bottom, Top)
    - Line intersects a boundary if the corresponding bit value in the two region codes are not the same
    - Intersection points with the window boundaries are calculated using the line-equation

Cohen-Sutherland: Steps

- For all other cases, do the following
  - Assign region code to the intersection point and discard the line segment “outside” (w.r.t. the particular boundary)
  - Repeat till both endpoints are completely inside or completely outside of the window

Calculating Line Intersections

- Consider a line with the end-points (x_1, y_1) and (x_2, y_2)
  - The y-coordinate of an intersection with a vertical window boundary can be calculated using:

    $$y = y_1 + m\,(x_{boundary} - x_1)$$

    where $x_{boundary}$ can be set to either wx_{min} or wx_{max}
  - The x-coordinate of an intersection with a horizontal window boundary can be calculated using:

    $$x = x_1 + (y_{\mathrm{boundary}} - y_1) / m$$

    where $y_{boundary}$ can be set to either wy_{min} or wy_{max}
  - m is the slope = $(y_2 - y_1) / (x_2 - x_1)$

Cohen-Sutherland Algorithm

- Better than brute force, but not the best
- Works well when number of lines, which can be clipped without further processing, is large compared to the size of the input set
  - Still checks for some lines that are completely outside
- Liang-Barsky algorithm
  - Parametric line-clipping algorithm
  - Reduces intersection calculation further than Cohen-Sutherland

Liang-Barsky Line Clipping

- For a line segment with endpoints (x_0, y_0) and (x_{end}, y_{end}), we can describe the line in parametric form:

$$x = x_0 + u\Delta x \qquad \Delta x = x_{end} - x_0$$
$$y = y_0 + u\Delta y \qquad 0 \le u \le 1 \qquad \Delta y = y_{end} - y_0$$

- In order to retain the line, we should have

$$xw_{\min} \le x_0 + u\Delta x \le xw_{\max}$$
$$yw_{\min} \le y_0 + u\Delta y \le yw_{\max}$$

Liang-Barsky Line Clipping

- Which can be rewritten as:

$$u\ p_k \le q_k \qquad k = 1, 2, 3, 4$$

$$p_1 = -\Delta x, \qquad q_1 = x_0 - xw_{\min}$$
$$p_2 = \Delta x, \qquad q_2 = xw_{\max} - x_0$$
$$p_3 = -\Delta y, \qquad q_3 = y_0 - yw_{\min}$$
$$p_4 = \Delta y, \qquad q_4 = yw_{\max} - y_0$$

k = 1, 2, 3, 4 correspond to Left, Right, Bottom and Top window boundaries

Liang-Barsky Line Clipping

- If $p_k = 0$ and $q_k < 0$ for any k
  - Discard the line and stop (the line is completely outside)
- Calculate parameters u_1 and u_2, that defines the part of the line that lies within the clip window

Liang-Barsky Line Clipping

- u_1: calculate for all those edges for which $p_k < 0$: $r_k = q_k / p_k$

  $u_1 = \max\{0, r_k\}$

- u_2: calculate for all those edges for which $p_k > 0$: $r_k = q_k / p_k$

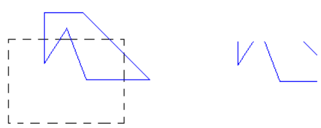
  $u_2 = \min\{1, r_k\}$

Liang-Barsky Line Clipping

- If $u_1 > u_2$, the line is completely outside, discard it
- Otherwise, the endpoints of the clipped line are calculated from the two values of *u*
  - $u_1 = 0$, one intersection point - (x1,y1)
    - $x1 = x0 + u_2.\Delta x$; $y1 = yo + u_2.\Delta y$
  - Otherwise two intersection points - (x1,y1), (x2,y2)
    - $x1 = x0 + u_1.\Delta x$; $y1 = yo + u_1.\Delta y$
    - $x2 = x0 + u_2.\Delta x$; $y2 = yo + u_2.\Delta y$

Fill-Area Clipping

- To clip a polygon fill area, we cannot directly apply a line-clipping method to the individual polygon edges
  - Line clipping may not produce a closed polyline



- Other efficient algorithms are available
  - Sutherland-Hodgman
  - Weiler-Atherton

Sutherland-Hodgman Polygon Clipping

- Basic idea
  - Four “clippers” - each corresponding to one of the clipping edges (window boundaries)
    - Left, Right, Bottom, Top
  - Each clipper takes as input a list of ordered pairs of vertices (edges) – produces another list of vertices as output

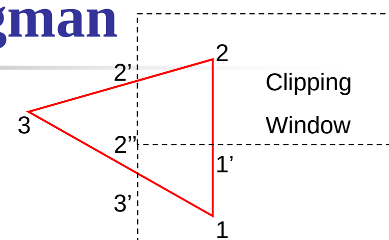
Sutherland-Hodgman Polygon Clipping

- Basic idea
  - The original polygon vertices are given as input to the first clipper (usually Left)
    - Follow some clipper order for checking, e.g. Left → Right → Bottom → Top
    - Follow some vertex naming convention (clockwise/anti-clockwise)

Sutherland-Hodgman

- For each clipper, the output is generated in the following way
  - Do for each input edge (vertex pair vi, vj)
    - vi = inside, vj = outside; *return* intersection point
    - vi = inside, vj = inside; *return* vj
    - vi = outside, vj = inside; *return* intersection point and vj
    - vi = outside, vj = outside; *return* NULL

Sutherland-Hodgman

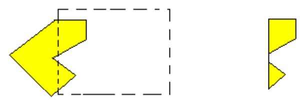


Input Edge → Left Clipper → Right Clipper → Bottom Clipper → Top Clipper

{1,2}:	(in-in)->{2}			
{2,3}:	(in-out)->{2’}	{2,2’}:(in-in)->{2’}	{2’,3’}:(in-out)->{2”}	
{3,1}:	(out-in)->{3’,1}	{2’,3’}:(in-in)->{3’}	{3’,1}:(out-out)->{}	
		{3’,1}:(in-in)->{1}	{1,2}:(out-in)->{1’,2}	{2”,1’}:(in-in)->{1’}
		{1,2}:(in-in)->{2}	{2,2’}:(in-in)->{2’}	{1’,2}:(in-in)->{2}
				{2,2’}:(in-in)->{2’}
				{2’,2”}:(in-in)->{2”}

Sutherland-Hodgman

- When a concave polygon is clipped with the Sutherland-Hodgman algorithm, extraneous lines may be displayed



- Since there is only one output vertex list, the last vertex in the list is always joined to the first vertex

Weiler-Atherton Polygon Clipping

- Can be used to clip a fill area that is either a convex polygon or a concave polygon
- Basic idea - instead of always proceeding around polygon edges as vertices are processed, sometimes follow window boundaries
  - A boundary is followed whenever a polygon edge crosses to the outside of that boundary

Weiler-Atherton Polygon Clipping

- Two rules
  - For an outside-to-inside vertex pair, follow polygon edges
  - For an inside-to-outside vertex pair, follow window boundary
- The direction of vertex traversal and window boundary traversal should be the same – clockwise/counter-clockwise
  - Linked to vertex naming convention

Weiler-Atherton Polygon Clipping

- Steps (assume anti-clockwise traversal) – Repeat till all the vertices are processed
  - Process vertices in anti-clockwise order *until* an inside-outside pair of vertices is encountered for one of the clipping boundaries
  - Follow window boundaries in anti-clockwise direction from the exit-intersection point to another intersection point already *seen*
    - Form the vertex list for this section of the clipped fill area

Weiler-Atherton Polygon Clipping

- Steps (assume anti-clockwise traversal) – Repeat till all the vertices are processed
  - Return to the exit-intersection point and continue processing the polygon edges in anti-clockwise order

3D Clipping

- Clipping is done after the geometric and viewing (including normalization) transformations are complete
  - So, we have the following

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = M \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Clipping procedures applied to these homogeneous coordinates

3D Clipping

- Clipping is done against the symmetric normalized view volume
  - Normalized cube with x, y, z in [-1,1]
- Point clipping – trivial, as in 2D
- Line and polygon clipping - extension of 2D algorithms
  - Cohen-Sutherland
  - Liang-Barsky
  - Sutherland-Hodgman
  - Weiler-Atherton

Point Clipping

- Because we have a normalised clipping volume, don't clip a point $P(x_h, y_h, z_h, h)$ if

$$-1 \le \frac{x_h}{h} \le 1 \qquad -1 \le \frac{y_h}{h} \le 1 \qquad -1 \le \frac{z_h}{h} \le 1$$

- Rearranging these we get

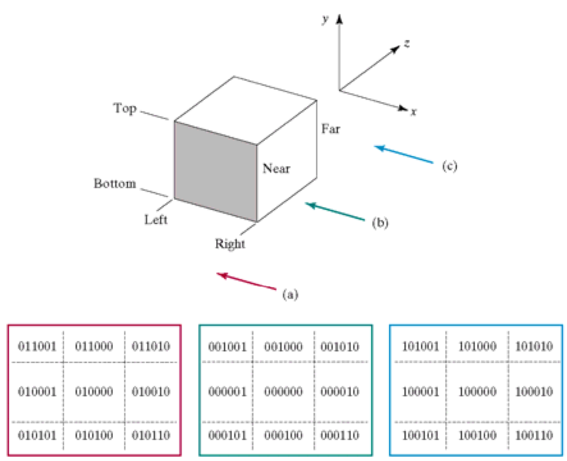
$$-h \le x_h \le h \quad -h \le y_h \le h \quad -h \le z_h \le h \quad if\ \ h > 0$$
$$h \le x_h \le -h \quad h \le y_h \le -h \quad h \le z_h \le -h \quad if\ \ h < 0$$

Region Code

- Similar to the case in two dimensions, we divide the world into regions
- This time we use a 6-bit region code to give us 27 different region codes
- The bits in these regions codes are as follows:

bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
Far	Near	Top	Bottom	Right	Left

Region Code



Region Codes In Front of Near Plane (a)

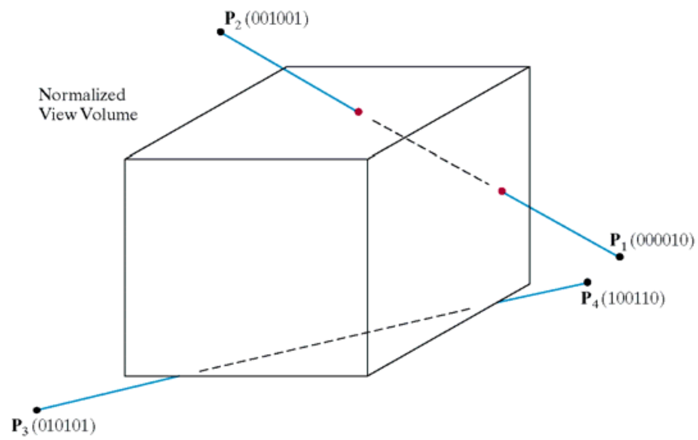
Region Codes Between Near and Far Planes (b)

Region Codes Behind Far Plane (c)

Line Clipping

- Label all end points with the appropriate region codes
- Trivial accept – all lines with both end-points having [000000] region code
- Trivial reject - Logical AND of both endpoints ≠ 000000
  - Example: next slide, the line from P_3[010101] to P_4[100110]

Line Clipping Example



Line Equation for 3D Clipping

- Line segments are given in parametric form
  - Parametric form of a line segment with end points $P_1(x1_h, y1_h, z1_h, h1)$ and $P_2(x2_h, y2_h, z2_h, h2)$

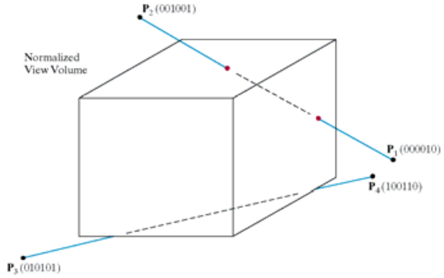
$$P = P_1 + (P_2 - P_1)u \qquad 0 \le u \le 1$$

  - From the parametric equation, equations for the homogeneous coordinates can be generated

$$x_h = x1_h + (x2_h - x1_h)u$$
$$y_h = y1_h + (y2_h - y1_h)u$$
$$z_h = z1_h + (z2_h - z1_h)u$$
$$h = h1 + (h2 - h1)u$$

3D Line Clipping Example

- Consider the line P_1[000010] to P_2[001001]
- The lines have different values in bit 2
  - It crosses the right boundary



3D Line Clipping Example (cont…)

- Right boundary is at $x = 1$, hence the following holds

$$x_p = \frac{x_h}{h} = \frac{x1_h + (x2_h - x1_h)u}{h1 + (h2 - h1)u} = 1$$

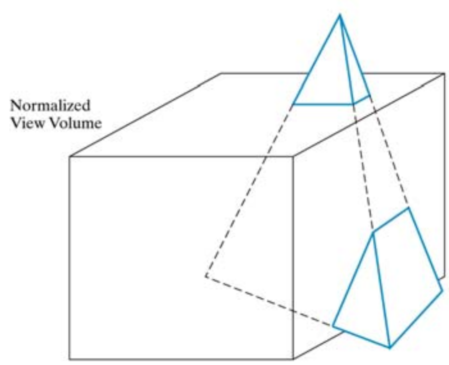
- Solving for u

$$u = \frac{x1_h - h1}{(x1_h - h1) - (x2_h - h2)}$$

- Using u, y_p and z_p can be found out similarly
- Continue the process for other clipping planes
  - Similar to 2D

3D Polygon Clipping

- The most common case in 3D clipping: clipping of graphics objects made up of polygons



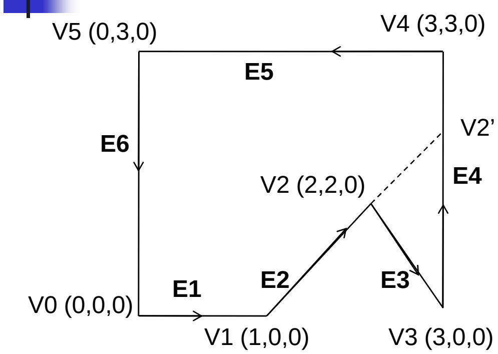
3D Polygon Clipping (cont…)

- First try to eliminate the entire object using its bounding volume
- If that is not possible, perform clipping on the individual polygons (surfaces) using
  - Sutherland-Hodgman (convex polygon)
  - Weiler-Atherton (both concave and convex polygons)

3D Polygon Clipping (cont…)

- Sutherland-Hodgman can be used for concave also
  - Split concave to a set of convex polygons
  - Vector method of splitting
    - Create edge vectors: $\mathbf{E} = \mathbf{V_{k+1} - V_k}$
    - Calculate z component of the cross product of consecutive edges ($z(\mathbf{E_i} \times \mathbf{E_j}) = E_{ix}.E_{jy} - E_{iy}.E_{jx}$)
    - If $z < 0$ for $\mathbf{E_i} \times \mathbf{E_j}$, extend $\mathbf{E_i}$ to split the polygon into two
    - Repeat till all edges are covered

Splitting Method - Example



E1 = **V1** – **V0**
E2 = **V2** – **V1**
E3 = **V3** – **V2**
E4 = **V4** – **V3**
E5 = **V5** – **V4**
E6 = **V0** – **V5**

z(**E2**×**E3**) < 0, hence **E2** extended, resulting in two convex polygons:

1. {V2, V2', V3}
2. {V0, V1, V2, V2', V4, V5}

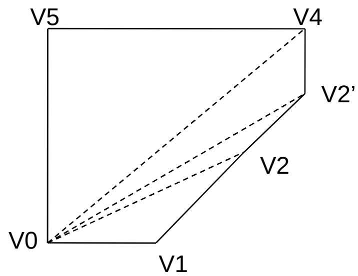
Splitting Method - Note

- Two assumptions
  - Polygon on XY plane (if not, apply transformations to bring it to XY plane)
  - No three consecutive vertices are collinear
- Other methods are also there

3D Polygon Clipping (cont…)

- Sutherland-Hodgman can be used for concave also
  - Split concave to a set of convex polygons
  - Split up a convex polygon into triangular mesh (easier to check triangle-plane intersection)
    - Input: vertex list V = {v0, v1,...,vn}
    - Take first three vertices from V – a triangle
    - Remove the middle of the three from V
    - Repeat till V contains only 3 vertices – the last triangle

Triangle Mesh - Example



Consider the previous example:
Initially,
V = {V0, V1, V2, V2’, V4, V5}
Step 1:
triangle 1: {V0, V1, V2}
V = {V0, V2, V2’, V4, V5}
Step 2:
triangle 2: {V0, V2, V2’}
V = {V0, V2’, V4, V5}
Step 3:
triangle 3: {V0, V2’, V4}
V = {V0, V4, V5}
Step 4:
triangle 4: {V0, V4, V5}
Stop

3D Polygon Clipping (cont…)

- Sutherland-Hodgman can be used for concave also
  - Split concave to a set of convex polygons
  - Split up a convex polygon into triangular mesh
  - Apply the algorithm to determine the intersection points of each triangle
    - with all the six faces of the symmetric normalized cube
  - Do for all triangles that made up the surface