Today’s class

- Clipping
- Parametric and point-normal form of lines
- Intersecting lines and planes
Normalized device coordinates

- Graphics imaging is device dependent
- Image development is device independent
- To simplify the process images are frequently generated using *normalized device coordinates* (NDC), where $x$ and $y$ values are between -1.0 and 1.0 (for OpenGL; other systems may use 0.0 to 1.0)
Converting NDC to DC

- To display an image a transformation from NDC to device coordinates (DC) needs to take place.
- To convert NDC to DC:
  - multiply by $\frac{1}{2}$ the number of pixels in the dimension
  - add $\frac{1}{2}$ the number of pixels in the dimension
Viewing transformation

- It is usually inconvenient to represent an image in DC or NDC
- Want to do it in *world coordinates* (WC)
- Setting the window and the viewport accomplishes the transformation from WC to DC, thus providing the *viewing transformation*
Clipping volume

- The combination of setting the window and the viewport defines the *clipping volume*.
- It is possible some primitives lie wholly or partially outside this volume.
- Need to insure these items are not displayed.
- This process is known as *clipping*.
Clipping points

- Clipping points is trivial
- A point is visible if
  - \( x_{\text{min}} \leq x \leq x_{\text{max}} \)
  - \( y_{\text{min}} \leq y \leq y_{\text{max}} \)
Clipping lines

- Lines lie:
  - wholly inside window
  - wholly outside window
  - partially inside and partially outside window

- Define a 4-bit *outcode* \((b_0b_1b_2b_3)\) describing relationship of line segment to window
  - \(b_0 = 1\) if \(y > y_{\text{max}}\) (point above window)
  - \(b_1 = 1\) if \(y < y_{\text{min}}\) (point below window)
  - \(b_2 = 1\) if \(x > x_{\text{max}}\) (point to right of window)
  - \(b_3 = 1\) if \(x < x_{\text{min}}\) (point to left of window)
Cohen-Sutherland Outcode Algorithm

- code₁ ← outcode (p₁); code₂ ← outcode (p₂);
- loop
  - if code₁ = 0 and code₂ = 0 then
    ▪ display line segment; exit loop;
  - if code₁ & code₂ ≠ 0 then
    ▪ reject line segment; exit loop;
  - if code₁ = 0 then
    ▪ swap (p₁, p₂); swap (code₁, code₂);
  - find a nonzero bit in code₁
  - find intersection of line with corresponding window boundary;
  - p₁ ← intersection point; code₁ ← outcode (p₁);
Observations

- Clipping of one line segment may occur more than once
- Don’t want to change the slope of the line
- Therefore, do the work in floating point numbers
- Works well when explicit form of line equation is used
Vertical lines

- Vertical lines have an infinite slope
- Will this cause a problem for the Cohen-Sutherland algorithm?
- Pair off and determine the answer to the above question.
  - If no, great!
  - If yes, how can you overcome the problem?
Vertical lines and Cohen-Sutherland clipping

For a vertical line $\Delta x=0$, but you’ll never divide by it!
Parametric form of a line

- Describes a “travelling” motion along a line or line segment
- The x and y coordinates of points on the line are described by linear equations involving a parameter variable (frequently t, u, or v)
- Example:
  \[
  \begin{align*}
  x &= t \\
  y &= 2t + 1
  \end{align*}
  \]
General form

- In general, the parametric form for the equation of a line is

  \[ x = at + b \]
  \[ y = ct + d \]
Line segments

- Frequently, in computer graphics we restrict t by $0 \leq t \leq 1$
- This generates a line segment
Finding the parametric form of a line segment

- Given two endpoints, \((x_1, y_1)\) and \((x_2, y_2)\)
- Let \(t = 0\) be at \((x_1, y_1)\) and \(t = 1\) be at \((x_2, y_2)\)
- Then:
  \[x = x_1 + (x_2 - x_1)t\]
  \[y = y_1 + (y_2 - y_1)t\]
Vector form of parametric equation

- Let the two endpoints of a line segment be denoted \( \mathbf{A} \) and \( \mathbf{B} \)
- The line between them is described by
  \[
  \mathbf{P}(t) = \mathbf{A} + (\mathbf{B} - \mathbf{A})t \quad 0 \leq t \leq 1
  \]
Slope vector

- The slope of a line is \( \frac{y_2-y_1}{x_2-x_1} \)
- The slope vector is defined as the vector \( (x_2-x_1, y_2-y_1) \)
- Note that the slope vector’s components are the coefficients of the linear terms of the parametric equations
Perpendicular lines

- Slopes are negative reciprocals of each other
- Given a line with a slope of \((y_2-y_1)/(x_2-x_1)\), which implies a slope vector of \((x_2-x_1, y_2-y_1)\), a perpendicular line will have slope \(-(x_2-x_1)/(y_2-y_1)\) and a slope vector \(-(y_2-y_1), (x_2-x_1)\))
- To get the equation of the perpendicular line choose the midpoint \((m_x, m_y)\) of the line to base it from
- Parametric form is now
  \[
  x(t) = m_x - (y_2 - y_1)t \\
y(t) = m_y + (x_2 - x_1)t
  \]
Point normal form

\[ \mathbf{n} \cdot \mathbf{r} = D, \] where \( \mathbf{n} \) is the normal to the line, \( \mathbf{r} \) represents all points on the line, and \( D = \mathbf{n} \cdot \mathbf{A} \), where \( \mathbf{A} \) is any point on the line

\[ \mathbf{n} \cdot (\mathbf{R} - \mathbf{A}) = 0 \]
Half spaces

- A line divides space into two halves, an outside half space and an inside half space.
- Given a line through point A and outward normal $\mathbf{n}$, then any point Q lies
  - in the outside half space if $(Q-A) \cdot \mathbf{n} > 0$
  - on the line if $(Q-A) \cdot \mathbf{n} = 0$
  - in the inside half space if $(Q-A) \cdot \mathbf{n} < 0$
Intersections of lines with lines and planes

- Let a ray be represented by $B + ct$
- Both lines and planes are represented in point normal form by $n \cdot (R - A) = 0$ or $n \cdot R = D$
- To find the intersection, substitute the ray equation into the point normal form for $R$:
  $$n \cdot (B + ct) = D$$
- Solve for $t$, which we will designate $t_{hit}$, the time at which the ray hits the line or plane:
  $$t_{hit} = \frac{(D - n \cdot B)}{(n \cdot C)}$$
Intersecting a ray with a polygon

- We frequently need to answer the questions
  - Where does a ray hit an object?
  - What part of a line is inside an object?
- We can solve these problems by determining the intersections of a ray with a polygon
Two intersections

- Let the ray in question be denoted $B + ct$
- It will hit the boundary of a convex polygon at most twice, once upon entering the polygon and once upon leaving
- Call these times $t_{in}$ and $t_{out}$
- Need to find intersection of ray with each edge and determine portion within polygon
Entering or leaving?

- When the ray intersects an edge, we need to decide if it is entering or leaving the polygon.
- Let \( \mathbf{n} \) represent the normal to an edge.
- The following conditions hold:
  - If \( \mathbf{n} \cdot \mathbf{c} < 0 \) the ray is entering.
  - If \( \mathbf{n} \cdot \mathbf{c} = 0 \) the ray is parallel.
  - If \( \mathbf{n} \cdot \mathbf{c} > 0 \) the ray is leaving.
Times to keep

- For entering times, want to keep the largest intersection time (or 0 if the ray starts inside the polygon)
- For leaving times, want to keep the smallest intersection time
- How would this be modified if we consider a line segment intersecting the polygon rather than a ray?
What’s left?

- The ray between $t_{\text{in}}$ and $t_{\text{out}}$ is the part inside the polygon
- The rest is clipped
Liang-Barksy clipping

- Useful when clipping against a rectangular window
- Uses parametric form of line equation
  - $x(t) = x_1 + t(x_2-x_1) = x_1 + t\Delta x$
  - $y(t) = y_1 + t(y_2-y_1) = y_1 + t\Delta y$
  - $0 \leq t \leq 1$
- Clipping conditions become
  - $x_{min} \leq x_1 + t\Delta x \leq x_{max}$
  - $y_{min} \leq y_1 + t\Delta y \leq y_{max}$
Inequality form

Each of the four inequalities is of the form $tp_k \leq q_k$, where

- $p_1 = -\Delta x$, $q_1 = x_1 - x_{\text{min}}$
- $p_2 = \Delta x$, $q_2 = x_{\text{max}} - x_1$
- $p_3 = -\Delta y$, $q_3 = y_1 - y_{\text{min}}$
- $p_4 = \Delta y$, $q_4 = y_{\text{max}} - y_1$
Lines parallel to window boundary

- For lines that are parallel to a window boundary, the corresponding $p_k$ will be 0.
- If the corresponding $q_k < 0$ the line is outside the boundary and can be ignored.
Other lines

- If $p_k < 0$ the infinite extension of the line enters the clipping region.
- If $p_k > 0$ the infinite extension of the line leaves the clipping region.
- In either case, the value of $t$ for which the line crosses the boundary is $q_k / p_k$. 
Liang-Barksy algorithm

- Let $t_1$ and $t_2$ define the ends of the clipped line that will be visible
- $t_1$ is the entering time, $t_2$ the leaving time
- Initialize $t_1$ to 0 and $t_2$ to 1
- For each of the four boundaries:
  - if $p_k = 0$ then
    - if $q_k < 0$ then reject line
  - else if $p_k < 0$ then
    - compute entering time
Liang-Barksy algorithm (cont.)

- if entering time > $t_1$ then
  - change $t_1$ to entering time
  - if $t_1 > t_2$ then reject line

- else // $p_k > 0$
  - compute leaving time
  - if leaving time < $t_2$ then
    - change $t_2$ to leaving time
    - if $t_1 > t_2$ then reject line

- If line has not been rejected then draw line between $t_1$ and $t_2$
Problems with polygons

- Edges of polygons are made of line segments.
- Recognize that when we clip a polygon’s edges, some of the edges may be completely removed and others may be partially removed.
- Thus, we may change the shape of the polygon during clipping.
Clipping polygons to a rectangular window

- Number vertices of polygon and window in a clockwise manner
- Clip against one window edge at a time
- Input is a set of polygon vertices
- Output is an updated set of polygon vertices
Sutherland-Hodgman algorithm

- Start with edge from last polygon vertex to first one, and then repeat for each successive edge:
  - Assume beginning vertex has been handled
  - If both vertices are inside window with respect to clipping edge output second vertex
Sutherland-Hodgman algorithm (cont.)

- If first vertex is inside and second vertex is outside, find intersection of polygon edge with clipping edge and output intersection
- If both vertices are outside window output nothing
- If first vertex is outside and second vertex is inside, find intersection of polygon edge with clipping edge and output intersection and then second vertex
Bounding box

- Clipping polygons can be time consuming, especially if it is many sided and lies completely outside the window.
- Use a bounding box (extent) to eliminate these quickly.
What’s visible?

- Once clipped, we now have a list of polygons to display
- We must determine which surfaces are visible and which are hidden
Object-space approaches

- Given a list of \( n \) polygons
- Take one polygon and compare it with the remaining ones to determine the part that is visible; render it
- The above polygon no longer needs to be considered
- Take one of the remaining polygons and repeat the process
- This is \( O(n^2) \) and thus should only be used for scenes with few polygons
Image-space approaches

- Follow ray tracing paradigm
- Consider a ray from viewpoint that passes through a pixel
- Find the closest intersection of the ray with all polygons
- Color pixel by the shade of the face closest to viewpoint
- Repeat for all pixels
- This is $O(n)$ if there are $n$ polygons
- However, they tend to be more jagged since they work at the pixel level
Back face removal

- A back face is one that is turned away from the eye of the camera
- The camera will not see it if the faces in front are opaque
- For a convex polyhedron:
  - every face is either wholly visible or wholly invisible
  - eliminating back faces is equivalent to hidden surface removal
Culling

- The process of removing something is known as **culling**
- In OpenGL, turn on culling of back faces with `glEnable (GL_CULL_FACE);` and then `glCullFace (GL_BACK);`