Clipping

- Once model-view-projection is performs, the clipping volume is used to determine what is drawn
- Clipping requires not only the coordinates of the vertices, but also the primitive (line, triangle, etc.)
- Performed before rasterization: clipping produces coordinates of vertices of the primitive so that it is fully inside clipping volume
- Focus on two dimensions—three-dimensional extensions possible but may be more tricky

Line Clipping

- Clipping rectangle is bounded by $x_1 \leq x \leq x_2$, $y_1 \leq y \leq y_2$.
- Line defined by the two end points (p_x, p_y) and (q_x, q_y).
- Possibliities:
  - Line completely inside
  - Line completely outside
  - Partially inside (one end point inside)
  - Partially inside (no end point inside)

Line Clipping: Naïve Algorithm

- If any end point is inside, keep them.
- Each of the four boundaries can be represented by a line segment.
- Perform line segment intersection for each of the four boundaries.
- If there are any intersection points, replace the outside end points by intersection (watch out for multiple intersections at corners).
- This works but not the most efficient (intersection test requires many arithmetic operations).

Line Clipping: Cohen-Sutherland Algorithm

- For each end point, we compute a 4-bit outcode. Each bit corresponds to a boundary, and it is 1 if it is outside according to that boundary (0 otherwise).
- e.g. if $x_1 = 10$, the point $(4, 5)$ will have an outcode of 1??? (if first bit corresponds to left boundary).
- Outcode is 0000 for a point inside.
- Outcodes are very easy to compute for each end point (4 comparisons).
- There are 4 bits but all coordinates are assigned one of 9 possible outcodes. (why not 16?)
- If a certain bit in the two end points are different, it means the line crosses that boundary.

Line Clipping: Cohen-Sutherland Algorithm

- If both points have outcode 0000, entire line is visible. Done.
- If bitwise AND of two outcodes is non-zero, then the two points are both outside with respect to one boundary. So entire line is not visible. Done
- Otherwise, pick one of the bit at which the outcodes differ. The line crosses that boundary. Do line intersection, replace the outside point with intersection.
- Repeat until the entire line is visible or invisible.

Polygon Clipping

- A polygon is defined as a sequence of vertices listed in some orientation (e.g. counter-clockwise)
- List of vertices $v_1, v_2, \ldots, v_n$.
- Convex polygon: clipped polygon is still one polygon
- Concave polygon: clipped polygon can become many dijoint polygons

Polygon Clipping: Sutherland-Hodgman Algorithm

- We assume clipping polygon is convex (even simpler: rectangular)
- Input is a list of vertices for the polygon
- The algorithm clips the polygon against each of the four boundaries one boundary at a time.
- If the polygon is concave, the output may be overlapping edges (on the boundary).

Polygon Clipping: Sutherland-Hodgman Algorithm

To clip against one boundary (e.g. $x = x_1$):

```
output = []
for (i = 0; i < n; i++) {
  curr = v[i];   next = v[(i+1)%n];
  pt = intersect(curr to next, boundary)
  if (next is inside) {
    if (curr not inside)
      append pt to output
    append next to output
  } else if (curr is inside)
    append pt to output
}
```

This is $O(n)$ time, and output can have $O(n)$ points.

Line Clipping in 3D

- Similar to 2D: outcode now has 6 bits
- Lines have to be intersected against planes
- Otherwise it is very similar

Polygon Clipping in 3D

- Clip polygon against bounding plane one plane at a time
- Similar to 2D: replace line intersection with line-plane intersection
- If polygons are restricted to triangles, this can be done very fast