Questions

1. Cohen-Sutherland xy outcodes have 4 bits. This suggests $2^4 = 16$ codes. Yet only 9 codes are used? What happened to the other 7 ?

2. Suppose you have a line segment joining two points (a, b, c, d) and (a', b', c', d') in clip coordinates and you wish to clip the line segment to a boundary plane, say $x = 1$ of the normalized view volume. Show how you can do this directly in clip coordinates. You do *not* need to do perspective division to find the intersection.

   Hint: This one is more challenging. It will test your linear algebra skills.

3. In the solution to the previous question, I argued it is *as cheap* [**EDIT: Feb. 17**] to do Cohen-Sutherland clipping in clip coordinates: because both require six divisions to find intersections with the clipping planes.

   But taking this a step back, you may ask, why not try to do the clipping way back in camera coordinates? Wouldn't this be cheaper? After all, computing clipping coordinates itself requires multiplication with a 4×4 matrix (e.g. `GL_PROJECTION` in the case of OpenGL) which seem like an unnecessary expense in the case the line gets thrown away anyhow.

4. How do you do line clipping if one of the vertices has $w < 0$? Are any special considerations required or can you just go ahead and use the method of Q2 above?

   Hint: there are issues. What are they? Look at the ABCD...JK figure from lecture 5 and think about what happens if you clip.

5. Write out the matrix for doing the window-to-viewport transformation that corresponds to:

   ```
   glViewport( x0, y0, width, height)
   ```

   Just deal with the (x, y) transformation here, not the depth z.

6. The one detail which I did not mention in the lecture is that in OpenGL, the (0,0) pixel is at the upper-left corner, not the lower left corner, and so the y variable increases from the top of the image to the bottom. *If you use the mouse and print out the mouse position in the window, you will indeed see that y increases from top to bottom.* Modify your solution from the previous question to account for this flip.

Answers

1. Some outcodes make no sense, for example $b_1b_0 = 11$ is impossible, since we cannot have x being simultaneously greater than 1 and less than -1. Similarly, $b_3b_2 = 11$ is impossible. There are $7 = 2 * 4 - 1$ ways in which either of those two conditions can happen, that is, 4 ways for the first and 4 for the second, minus 1 for doublecounting in case of outcode 1111.

2. Let's reason geometrically about the line joining the two 4D points. The two 4D points correspond to two 3D points in the normalized view volume. The key property to use here is that each of these two 3D points corresponds to a line in clip coordinates (4D): each line contains one of the original 4D points and each line passes through the origin. These two 4D lines span a 2D hyperplane π in 4D space and this hyperplane π contains the origin. In particular, the 4D parametric line

$$t(a, b, c, d) + (1 - t)(a', b', c', d') \qquad (*)$$

lies in this 2D hyperplane as well.

I will first show that this line (*) in 4D defines a line in 3D. (In my opinion, this is not totally obvious since perspective division is not linear.) Here's my argument: Let (x, y, z, w) be the four coordinates of clip space. If we consider the intersection of the hyperplane π with the plane $w = 1$, we get a line l in 4D such that the 4th coordinate of all points on the line is always 1 (and so we can think of the line l as being either in 4D or 3D). This 4D line represents a line in 3D space. For any t, (*) defines a point on this line and there is a 4D line from the origin through that point. That 4D line lies in the hyperplane π and all points on that 4D line represent the same 3D point. In particular, the 3D point represented by that line must lie on the 3D line that we called l. But the points $(a/d, b/d, c/d)$ and $(a'/d', b'/d', c'/d')$ lie on the line l for the same reason. Thus, for any t, the point $(*)$ lies on the line joining those two 3D points.

Next, consider the intersection of line l with the plane $x = 1$. We want to compute it in clip coordinates only. We can solve for t using:

$$t(a, b, c, d) + (1 - t)(a', b', c', d') = (w, \_\_, \_\_, w)$$

which gives us two equations for t and w, namely $ta + (1 - t)a' = w$ and $td + (1 - t)d' = w$ or $(a - a')t + (d - d')(1 - t) = 0$ and hence

$$t = \frac{a' - d'}{(a - a') - (d - d')}$$

We then substitute into (*) to get the 4D point. This gives the same result as if we had done the perspective division and then intersected with $x = 1$ in 3D.

[EDIT: Feb 17] In the original posting of these solutions, I arged that doing it in clip coordinates is cheaper because we use only one division instead of six. However, one should keep in mind that to do clipping one needs to do a similar calculation for $x = -1$ and for $y = \pm 1, z = \pm 1$. So six divisions in total are needed, which is the same as if one does the perspective division.

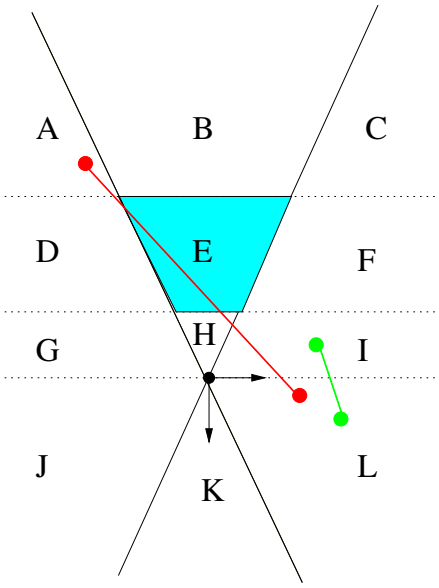
3. The bottleneck is the division, not the multiplication. Division generally is much slower than multiplication. Multiplication is easily parallelizable e.g. Karatsuba's algorithm, and hence fast multiplication is possible in hardware. But fast division is much more difficult; it is inherently serial. (Think long division.) Clipping to all six planes is going require six divisions, not matter which coordinate system you use.

4. To do line clipping using Cohen-Sutherland, we still need to correctly treat points that lie behind the camera. Recall from lecture 5 that such points had $z > 0$ in camera coordinates, and are transformed into clip coordinate 4-tuple (wx, wy, wz, w) with $w < 0$.

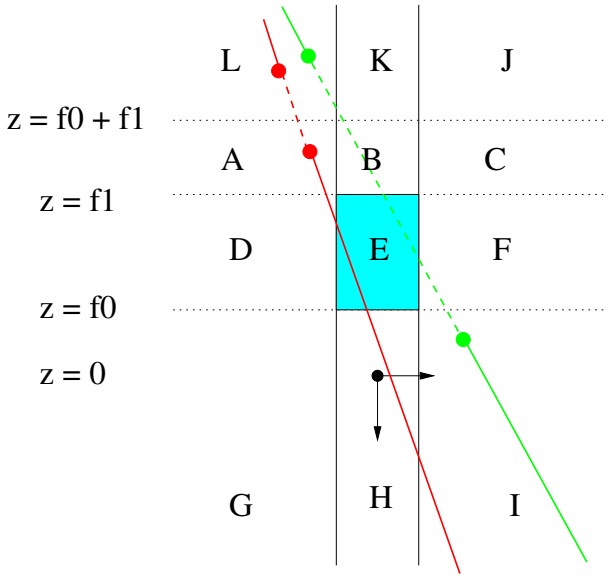
   Reconsider the figure from lecture 5 which illustrated the projective transformation (see below). Note that on the right, I have done the perspective mapping but I haven't done the normalization. That doesn't affect your reasoning in this question though.

   I have added two interesting examples of line segments. For both of these line segments (on the left), the two points straddle the $z = 0$ boundary.

   We know that points on the $z = 0$ boundary are projected to points at infinity (lecture 5), so we note that each of the two line segments in fact maps to a pair of semi-infinite lines (see figure below). What we really need to do, therefore, is to clip each of the semi-infinite lines. Note that the red line should be clipped to region E, and so special care must be taken in clip coordinates to do this. If you just naively consider the line segment between the two red dots on the right, you'll trivially reject it, which would be incorrect. Similary, the green line on the left *should* be trivially rejected, but if the clipping is done by Cohen-Sutherland on the right, then it would be clipped to region E. So in general, when there is a line with clip coordinate $w < 0$, special considerations must be taken. Details omitted.



camera coordinates



(non−normalized) projective coordinates

5. The mapping below takes $[-1,1] \times [-1,1]$ to $[0,0] \times [\texttt{width}, \texttt{height}]$.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \texttt{x0} \\ 0 & 1 & \texttt{y0} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\texttt{width}}{2} & 0 & 0 \\ 0 & \frac{\texttt{height}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

6. To account for the top bottom flip, start by scaling the y value by -1.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \texttt{x0} \\ 0 & 1 & \texttt{y0} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\texttt{width}}{2} & 0 & 0 \\ 0 & \frac{\texttt{height}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$