Homework 4 Solution

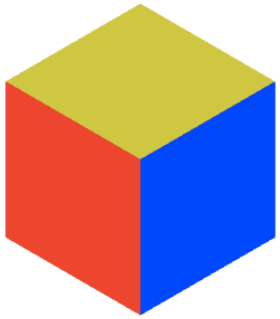
1) Given the 3D cube example in programs: ortho.js and ortho.html (available on the course web page), if the view position and the orthographic viewing volume is change into each of the following situations, how will the final 2D image change from its original image?

   a. mvMatrix=lookAt(vec3(-4, 0, 0), at, up); // pMatrix does not change

      Shows the cyan square, the left face of the cube, because the eye is now to the left of the cube looking to the center of the cube.

   b. mvMatrix=lookAt(vec3(3, 3, 3), at, up); // pMatrix does not change

      Shows the top, right, and front faces of the cube



   c. mvMatrix=lookAt(vec3(3, 3, 3, at, up);
      pMatrix=ortho(-3, 3, -3, 3, -1, 1);

      No display, because the cube is outside of the view volume

   d. pMatrix= ortho(-6, 6, -3, 3, 2, 10); // mvMatrix does not change

      The cube is now inside the view volume, between near plane (-2) and far plane (-10). It appears taller than before. The width of the view volume is doubled and the height of the view volume does not change. Therefore, the height of the back side of the cube looks twice as tall as its wide.

   e. pMatrix=ortho(0, 4, 0, 3, 2, 10); // mvMatrix does not change

Only ¼ of the cube (the upper, right ¼) is inside the view volume and displayed. The width and height ratio of the displayed portion (back face) is 3:4

2) Given: mvMatrix=lookAt(vec3(4, 4, - 4), at, up);
pMatrix=ortho(-2, 2, -4, 4, -10, 10);

show:

- the mvMatrix
- the pMatrix
- the coordinates of point F(1, 1, -1) and B(1, 1, 1) when converted into the final clip coordinates. (show intermediate steps in deriving the results)

n=eye-look=[4, 4, -4]
normalized n: [0.577, 0.577, -0.577]
u=upxn = [0, 1, 0] x [0.577, 0.577, -0.577]
normalized u: [-0.707, 0, -0.707]
v=n x u = [0.577, 0.577, -0.577] x [-0.707, 0, -0.707]
normalized v: [-0.408, 0.816, 0.408]

-dot(n, eye) = -6.928
-dot(u, eye) = 0
-dot(v, eye) = 0

$$\text{view matrix} = \begin{bmatrix} -0.707 & 0 & -0.707 & 0 \\ -0.408 & 0.816 & 0.408 & 0 \\ 0.577 & 0.577 & -0.577 & -6.928 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{projection matrix} = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & -0.1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{M= projection matrix x view matrix=} \begin{bmatrix} -0.35 & 0 & -0.35 & 0 \\ -0.102 & 0.204 & 0.102 & 0 \\ -0.0577 & -0.057 & 0.057 & 0.69 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$F' = M * F = \begin{bmatrix} -0.35 & 0 & -0.35 & 0 \\ -0.102 & 0.204 & 0.102 & 0 \\ -0.0577 & -0.057 & 0.057 & 0.69 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 5.19 \\ 1 \end{bmatrix}$$

3) Changing the orthographic viewing volume in problem 2) to a frustum with left=-2, right=2, bottom=-4, top=4 for the near plane, and the near plane at distance 4 and far plane at distance 10 from the eye/camera. How would you call the perspective function to set up the corresponding pMatrix in the .js program?

Convert Frustum(-2, 2, -4, 4, 4, 10) into perspect
Aspect = (right-left)/(top-bottom) = (2-(-2))/(4-(-4)) = 0.5
viewAngle = 2*arctan(1/2*(top-bottom)/N) = 2*arctan(0.5*(4-(-4))/4)=90 degrees

➔ perspect(90, 0.5, 4, 10)

4) With the perspective viewing volume defined in problem 3), what will be the x and y coordinates of points F(1, 1, -1) and B(1, 1, 1) when projected onto the near plane?

F'x = N*Fx/(-Fz) = 4*1/(1)=4
F'y = N*Fy/(-Fz) = 4*1/(1)=4
F': (4, 4)

B'x = N*Bx/(-Bz) = 4*1/(-1)= -4
B'y = N*By/(-Bz) = 4*1/(-1)= -4
B': (-4, -4)

5) Given a **viewing window** defined with left=50, right=150, bottom=0, and top=100, apply the Cohen-Sutherland Clipping algorithm to determine which segment of a line defined by two end points: P1(20, -10) and P2(200, 50) be displayed on the browser canvas. (The line P1-P2 will be clipped to P1'-P2' by the boundaries of the viewing window. Compute the two points P1' and P2'.)

Outcode for P1 is 0101
Outcode for P2 is 0010
0101 or 0010 is not equal to 0
0101 and 0010 is 0
So, need to perform clipping.

First, compute the slope of the line P1-P2: k=(P1.y-P2.y)/(P1.x – P2.x) = (-10-50)/(20-200) = 1/3

Clip from the left,
A.y = P1.y + k*(W.left – P1.x)
= (-10) + 1/3 * (50 – 20) = 0
So, point A is (50, 0)

Outcode for A is 0000
Outcode for P2 is 0010

0000 or 0010 is not equal to 0
0000 and 0010 is 0
so, require further clipping.

Clip from the right,
A.y = P1.y + k * (W.right – P1.x)
= (-10) + 1/3*(150-20) = 100/3
So, point B is (150, 100/3)

Outcode for A and B are both 0000 ➔ done

6) Suppose the line segment P1'-P2' is to be displayed in a **viewport** defined on the browser canvas by: left=200, right=600, bottom= 100, top=700. Compute the positions of these two points as they appear in the viewport.

Given:

World window (clipping window)
Left = 50, right =150, bottom = 0, top = 100 ➔ XWmin=50, XWmax=150, YWmin=0, YWmax=100

and

Viewport:
Left=200, right=600, bottom=100, top=700 ➔ XVmin=200, XVmax=600, YVmin=100, YVmax=700

Sx=(XVmax-XVmin)/(XWmax – XWmin) = (600-200)/(150-50)=400/100=4
Sy=(YVmax-YVmin)/(YWmax – Ywmin) = (700-100)/(100-0)=6
Tx=(XWmax*XVmin – XWmin*XVmax)/(XWmax - XWmin)=(150*200-50*600)/(150-50)=0
Ty=(YWmax*YVmin - YWmin*YVmax)/(YWmax - YWmin)=(100*100-0)/(100-0)=100

To compute the points as they are transformed in the viewport, apply the following transformation:

Xv=Sx*Xw + Tx
Yv=Sy*YW+ Ty

Xv=4*Xw
Yv=7*Yw+100

For P1' (50, 0) (aka point A in problem 5):
P1'.x = 4 * 50 = 200
P1'.y = 6 * 0 + 100 = 100
So, P1' in viewport is at location (200, 100)

For P2' (150, 100/3) (aka point B in problem 5)
P2'.x = 4 * 150 = 600
P2'.y = 6 * 100/3 + 100 = 300
So, P2' in viewport is at location (600, 300)