CS 370 - Assignment 2

1. Show the following sequences commute:

A rotation R_z and a uniform scaling (same scale factor β in all directions) S are given by

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad S = \begin{bmatrix} \beta & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forming the matrix products gives

$$R_z S = \begin{bmatrix} \beta\cos\theta & -\beta\sin\theta & 0 & 0 \\ \beta\sin\theta & \beta\cos\theta & 0 & 0 \\ 0 & 0 & \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad SR_z = \begin{bmatrix} \beta\cos\theta & -\beta\sin\theta & 0 & 0 \\ \beta\sin\theta & \beta\cos\theta & 0 & 0 \\ 0 & 0 & \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hence the operations commute. A similar relationship holds for R_y and R_x.

2. Despite the fact that all affine transformations are defined by 12 parameters which can be generated using appropriate translations, rotations, and scalings, we *cannot* generate all objects by using *any* order of transformations. Each transformation order (e.g. TRS) will produce a different set (though not mutually exclusive) of objects, i.e. the transformations do not necessarily commute. Consider a rotation and non-uniform scaling given by R_z and S:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad S = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forming the matrix products gives:

$$R_z S = \begin{bmatrix} \beta_x\cos\theta & -\beta_y\sin\theta & 0 & 0 \\ \beta_x\sin\theta & \beta_y\cos\theta & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad SR_z = \begin{bmatrix} \beta_x\cos\theta & -\beta_x\sin\theta & 0 & 0 \\ \beta_y\sin\theta & \beta_y\cos\theta & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. This is known as an oblique projection which takes the 3D coordinates and projects them into 2D with the z-axis at a $-135°$ angle. Hence we want a projection matrix which transforms the coordinate axes as:

$$x:\ (1,0,0) \to (1,0,0) \quad y:\ (0,1,0) \to (0,1,0) \quad z:\ (0,0,1) \to (-\tfrac{1}{\sqrt{2}},-\tfrac{1}{\sqrt{2}},0)$$

Assuming the form of the homogeneous projection matrix

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & 0 \\ p_{21} & p_{22} & p_{23} & 0 \\ p_{31} & p_{32} & p_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The x-axis transformation gives

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & 0 \\ p_{21} & p_{22} & p_{23} & 0 \\ p_{31} & p_{32} & p_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow p_{11} = 1 \quad p_{21} = p_{31} = 0$$

The y-axis transformation gives

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & p_{12} & p_{13} & 0 \\ 0 & p_{22} & p_{23} & 0 \\ 0 & p_{32} & p_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow p_{22} = 1 \quad p_{12} = p_{32} = 0$$

Finally the z-axis transformation gives

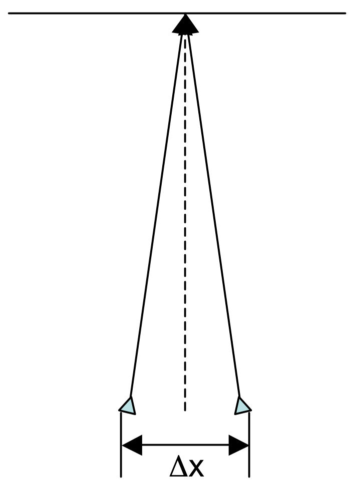
$$\begin{bmatrix} -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & p_{13} & 0 \\ 0 & 1 & p_{23} & 0 \\ 0 & 0 & p_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$\Rightarrow p_{13} = p_{23} = -\tfrac{1}{\sqrt{2}} \quad p_{33} = 0$$

Hence the final oblique projection matrix is

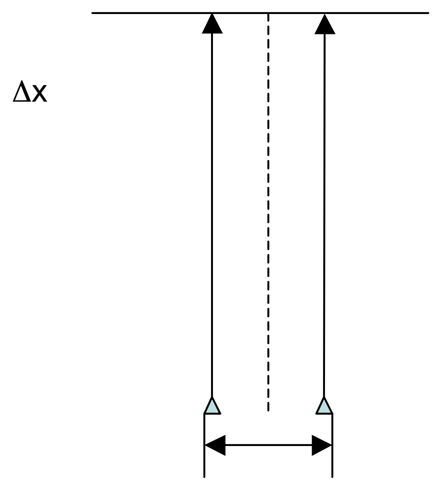
$$P = \begin{bmatrix} 1 & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 1 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. Ah, my favorite question… There are two schools of thought on creating stereo views, but both involve generating two images by offsetting the camera by $\pm\frac{\Delta x}{2}$ from a nominal center point (i.e. standard camera rendering position). The first is known as the toe-in method where the cameras are pointed at a common point (e.g. the origin). This would be done using



Left image => `gluLookAt(x-dx/2,y,z,0,0,0,0,1,0);`
`// Render scene`
Right image => `gluLookAt(x+dx/2,y,z,0,0,0,0,1,0);`
`// Render scene`

This method, while simple, produces vertical parallax issues (come discuss it with me for more details). A better method is known as parallel parallax which has the two cameras pointing in the same parallel direction. This would be done using



Left image => `gluLookAt(x-dx/2,y,z,x-dx/2,0,0,0,1,0);`
`// Render scene`
Right image => `gluLookAt(x+dx/2,y,z,x+dx/2,0,0,0,1,0);`
`// Render scene`