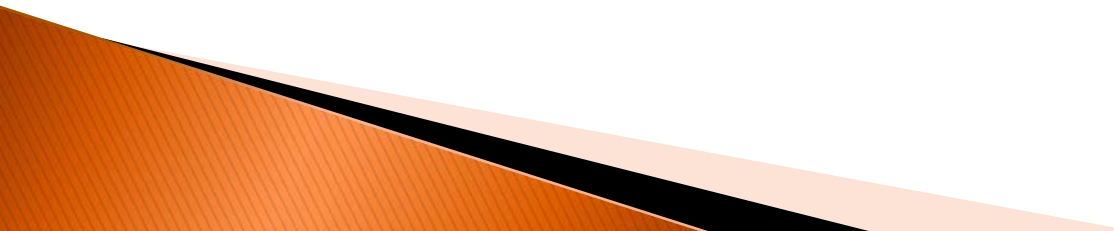


CSE 3200 Micro-Computer Graphics

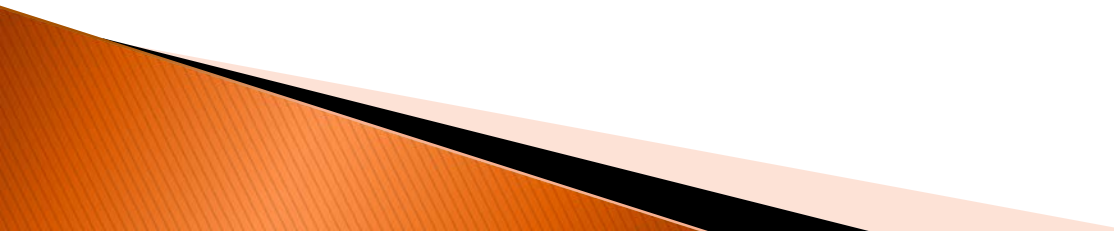
Matrices in CG

Presenter: Girendra Persaud
University of Guyana

Outline

- ▶ What is a Matrix?
 - ▶ Matrices in CG
 - ▶ Matrix Vector Transformation
 - ▶ Matrix Multiplication
 - ▶ Identity Matrix
 - ▶ Transformation Matrices
 - ▶ Translation
 - ▶ Scaling
 - ▶ Rotation
 - ▶ Perspective Matrix
 - ▶ Questions?
 - ▶ Review Questions
- 

What is a Matrix?

- ▶ A 2 dimensional array of numeric data where each row or column consist of a numeric values
 - ▶ A way to describe n-dimensional transformations (a qualitative change) in one convenient package
 - ▶ OpenGL uses 4x4 matrices
- 

Matrices in CG?

$$\begin{pmatrix} x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- ▶ Matrices and vectors in CG are in a form that allow for the various equations necessary for 3D manipulation of points
- ▶ The vector type has four values x , y , z and w .
 - The w value is used to transform world coordinates onto screen coordinates for output onto a 2D screen and it is initially set to 1
- ▶ The matrices used to transform points in 3D space are of size 4×4 , with each row representing a homogenous vector (x, y, z, w) .
- ▶ The first three rows contain the world space coordinates of the local x , y , and z axes and the forth generally contains $(0,0,0,1)$

Matrix Vector Transformation

- ▶ To transform a vector by a matrix you multiply the vector by the matrix to produce the new vector

$$\begin{pmatrix} x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_v \\ y_v \\ z_v \\ w_v \end{pmatrix} = \begin{pmatrix} x_1 * x_v + y_1 * y_v + z_1 * z_v + w_1 * w_v \\ x_2 * x_v + y_2 * y_v + z_2 * z_v + w_2 * w_v \\ x_3 * x_v + y_2 * y_v + z_3 * z_v + w_3 * w_v \\ 0 * x_v + 0 * y_v + 0 * z_v + 1 * w_v \end{pmatrix} = \begin{pmatrix} x'_v \\ y'_v \\ z'_v \\ w_v \end{pmatrix}$$

Matrix Multiplication

- ▶ Frequently, a set of points will have to be transformed by a series of matrix transformations, for example
 - scale a point by a half in the z plane,
 - move it by 20 units in the x plane
 - rotating it 30 degrees in the y plane
- ▶ The matrix for the different transformations are multiplied together to form a single matrix reducing the overall work significantly [combining more than one matrix]

Matrix Multiplication

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{pmatrix} * \begin{pmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{32} & y_{33} & y_{34} \\ y_{41} & y_{42} & y_{43} & y_{44} \end{pmatrix} = \begin{pmatrix} y'_{11} & y'_{12} & y'_{13} & y'_{14} \\ y'_{21} & y'_{22} & y'_{23} & y'_{24} \\ y'_{31} & y'_{32} & y'_{33} & y'_{34} \\ y'_{41} & y'_{42} & y'_{43} & y'_{44} \end{pmatrix}$$

$$\begin{pmatrix} y'_{11} \\ y'_{21} \\ y'_{31} \\ y'_{41} \end{pmatrix} = \begin{pmatrix} x_{11} * y_{11} + x_{12} * y_{21} + x_{13} * y_{31} + x_{14} * y_{41} \\ x_{21} * y_{11} + x_{22} * y_{21} + x_{23} * y_{31} + x_{24} * y_{41} \\ x_{31} * y_{11} + x_{32} * y_{21} + x_{33} * y_{31} + x_{34} * y_{41} \\ x_{41} * y_{11} + x_{42} * y_{21} + x_{43} * y_{31} + x_{44} * y_{41} \end{pmatrix}$$

$$\begin{pmatrix} y'_{12} \\ y'_{22} \\ y'_{32} \\ y'_{42} \end{pmatrix} = \begin{pmatrix} x_{11} * y_{12} + x_{12} * y_{22} + x_{13} * y_{32} + x_{14} * y_{42} \\ x_{21} * y_{12} + x_{22} * y_{22} + x_{23} * y_{32} + x_{24} * y_{42} \\ x_{31} * y_{12} + x_{32} * y_{22} + x_{33} * y_{32} + x_{34} * y_{42} \\ x_{41} * y_{12} + x_{42} * y_{22} + x_{43} * y_{32} + x_{44} * y_{42} \end{pmatrix}$$

$$\begin{pmatrix} y'_{13} \\ y'_{23} \\ y'_{33} \\ y'_{43} \end{pmatrix} = \begin{pmatrix} x_{11} * y_{13} + x_{12} * y_{23} + x_{13} * y_{33} + x_{14} * y_{43} \\ x_{21} * y_{13} + x_{22} * y_{23} + x_{23} * y_{33} + x_{24} * y_{43} \\ x_{31} * y_{13} + x_{32} * y_{23} + x_{33} * y_{33} + x_{34} * y_{43} \\ x_{41} * y_{13} + x_{42} * y_{23} + x_{43} * y_{33} + x_{44} * y_{43} \end{pmatrix}$$

$$\begin{pmatrix} y'_{14} \\ y'_{24} \\ y'_{34} \\ y'_{44} \end{pmatrix} = \begin{pmatrix} x_{11} * y_{14} + x_{12} * y_{24} + x_{13} * y_{34} + x_{14} * y_{44} \\ x_{21} * y_{14} + x_{22} * y_{24} + x_{23} * y_{34} + x_{24} * y_{44} \\ x_{31} * y_{14} + x_{32} * y_{24} + x_{33} * y_{34} + x_{34} * y_{44} \\ x_{41} * y_{14} + x_{42} * y_{24} + x_{43} * y_{34} + x_{44} * y_{44} \end{pmatrix}$$

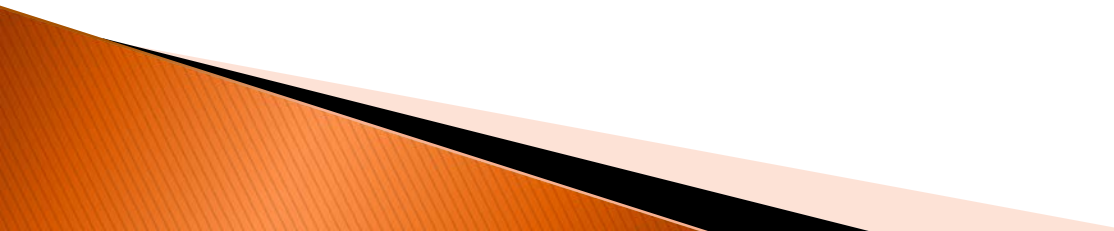
Matrix Multiplication

- ▶ $M \times N$ not equal to $N \times M$ so order is important.
- ▶ Suppose we have
 - a rotation in the x axis R_X
 - followed by a rotation in the z axis R_Z to combine
 - you do a $R_Z R_X$, where R_X is the multiplicand and R_Z is the multiplier.
- ▶ Multiplication is done from right to left $a \times b \times (c \times d)$.

Identity Matrix

- ▶ The identity matrix, when used as a multiplier or the multiplicand the result is the matrix is it multiplied with.

Transformation Matrices

- ▶ Translation Matrix
 - ▶ Scale Matrix
 - ▶ Rotation Matrix
 - ▶ Perspective Matrix
- 

Translation Matrix

- ▶ Used to move a point in 3D space relative to the world coordinates. Changes in x, y and z values.
- ▶ OpenGL call is `glTranslate(x,y,z);`

$$\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Usage glTranslate(x,y,z);

- ▶ Call to the matrix: glTranslate(x,y,z);
- ▶ Call to the object

```
80 //right arm
81 glPushMatrix();
82         glTranslatef(-2,0,0);
83         glScalef(0.5, 0.5, 0.5);
84         drawDECube();
85 glPopMatrix();
```

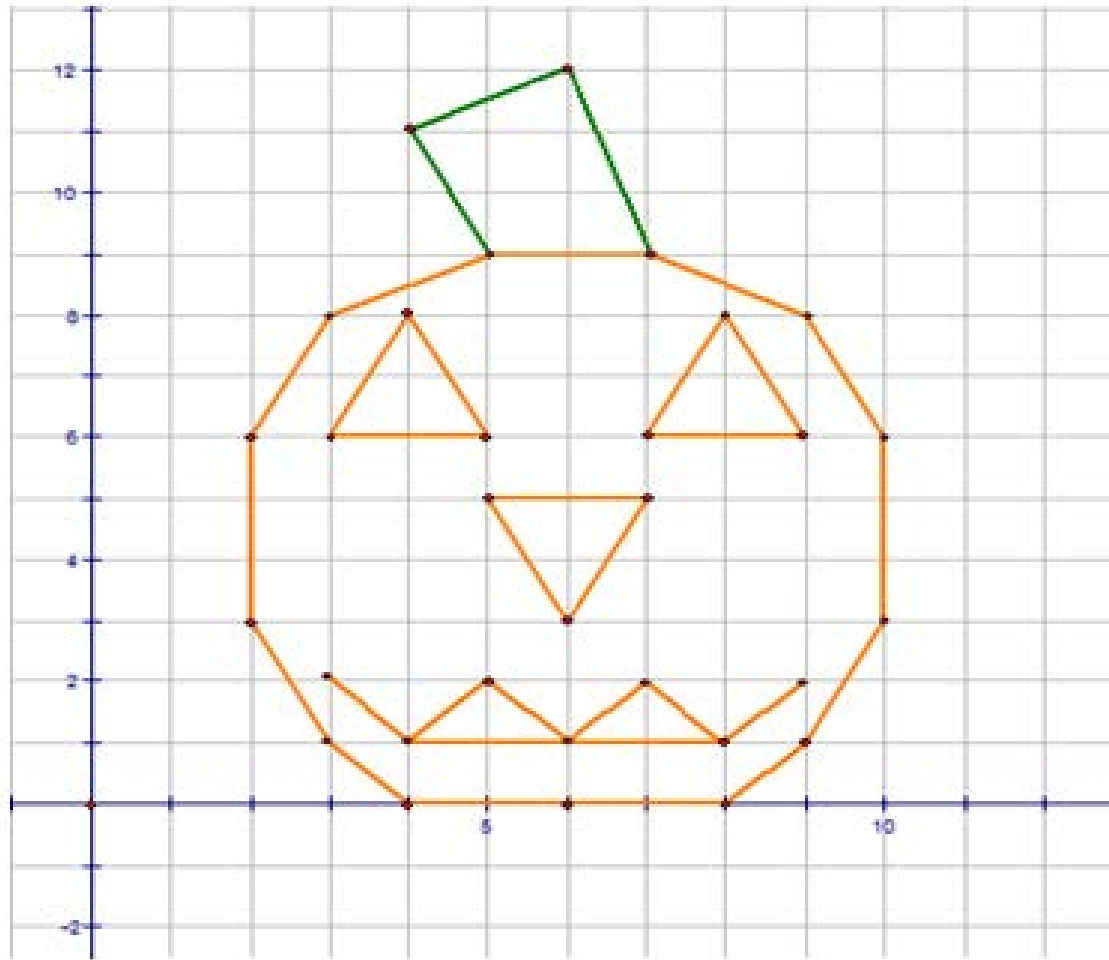
Scaling Matrix

- ▶ Scales an object relative to the origin of the world coordinates
- ▶ When you scale an object,
 - it must first be translated to the origin,
 - scaled
 - and then returned to the original position

▶ `glScale(x,y,z);`

$$\begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Usage glScale(x,y,z)



Rotation Matrix

- ▶ Positive rotation is clockwise and negative rotation is anti-clockwise – i.e if you look down the positive direction of any axis
- ▶ Rotations are non-commutative, rotating an object in the x before you rotate it in the y is not the same the other way around

Rotation Matrix

- ▶ Similar to scaling, rotation of an object is done in relation to the origin
 - You must first move the object to the origin,
 - rotate
 - then translate it back to its original position
- ▶ example `glRotatef(45,1,0,0);`

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos x & -\sin x & 0 \\ 0 & \sin x & \cos x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} \cos y & 0 & \sin y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin y & 0 & \cos y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} \cos z & -\sin z & 0 & 0 \\ \sin z & \cos z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about the X-Axis

Rotation about the Y-Axis

Rotation about the Z-Axis

Perspective Matrix

- ▶ used to transform objects into a 3D view relative to their distance from the origin
- ▶ It takes objects from being flat with a size irrespective of distance along the z-axis and transforms them into realistic, distance dependent objects

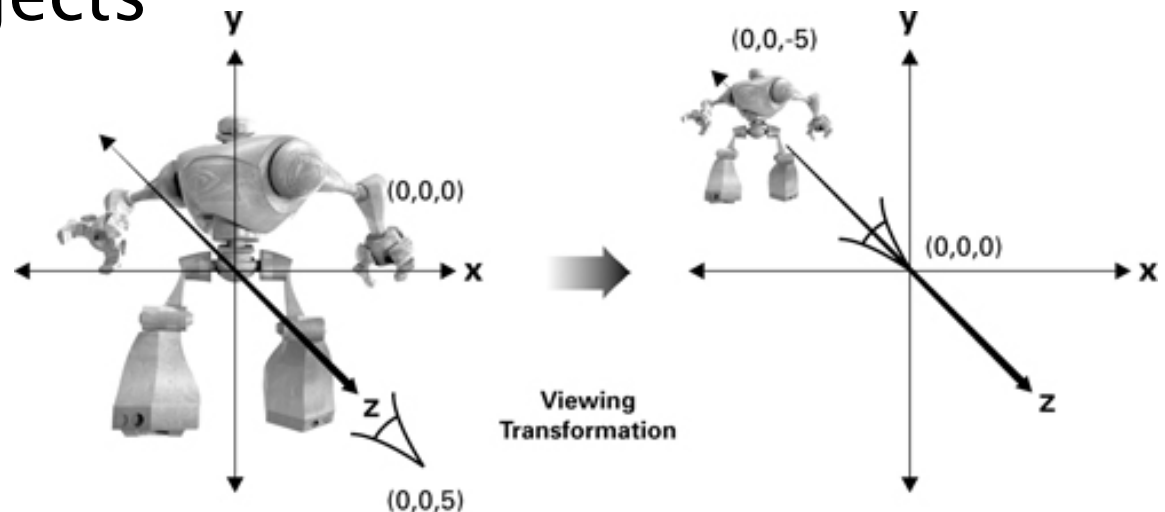


IMAGE:

http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter04.html

Perspective Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}$$

- ▶ The value of d is the distance between
 - the point the viewer is looking from (the center of projection)
 - and the plane (the vertical plane in front of the viewer which separates the display world from the undisplayed world; all objects between the viewer and the view of the plane are not displayed as if they were behind the viewer).

Questions?

Review Questions

- ▶ Create a single matrix which will rotate an object about the x-axis by 45 degrees before translating by (30, 40, 0).
- ▶ Rotate a point (10, 20, 15) by -30 degrees around the z-axis followed by a 90 degrees in the x-axis.