CSCD 396

Beginning Graphics

Today’s topic

- Different kinds of transformation;
- Homogeneous coordinates and Cartesian coordinates;
- Concatenation of transformation;
- Transformation in the application;
- Transformation in the shader;
- Use of **glm** library.

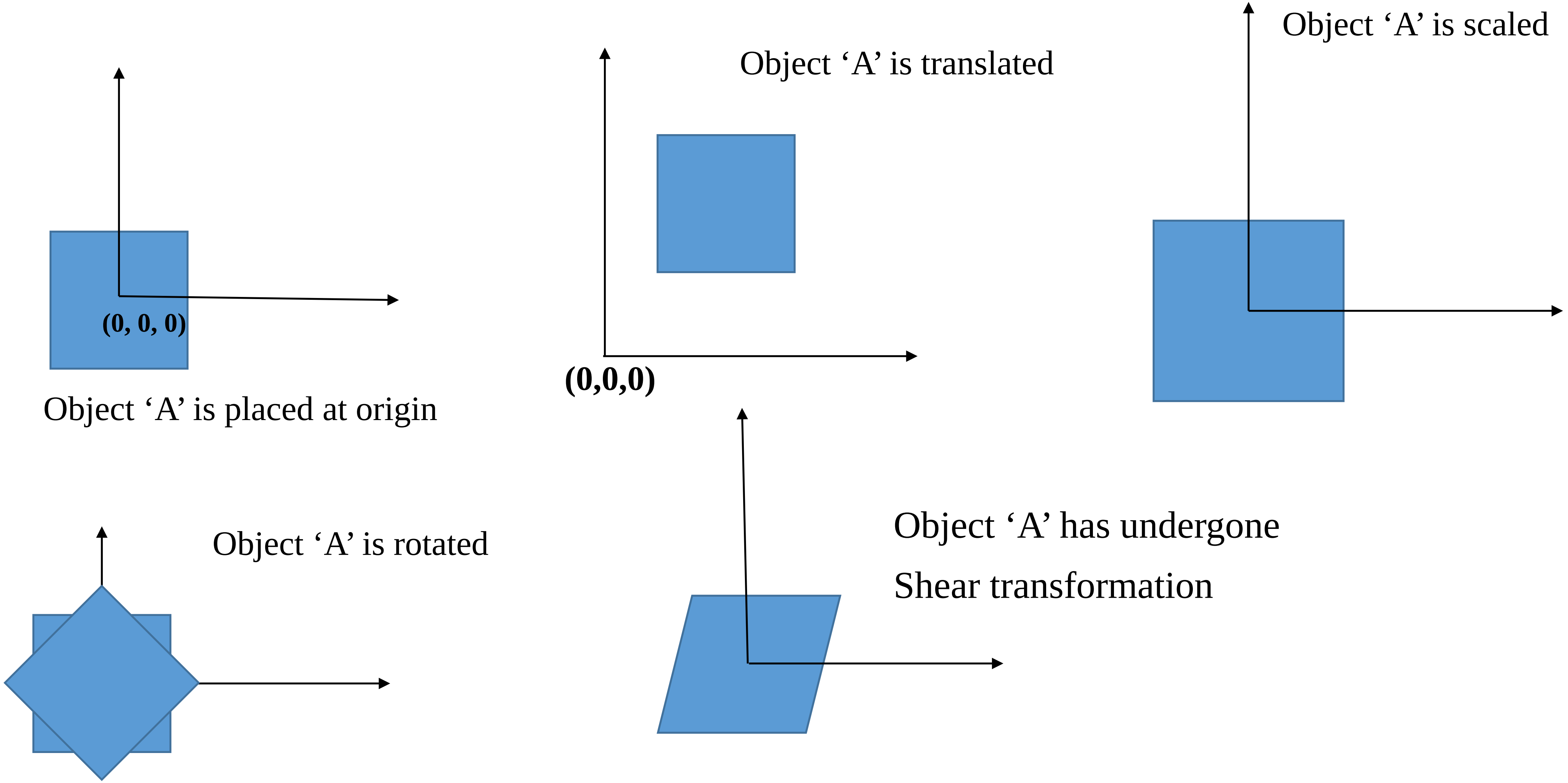
Transformation

- An object is defined as a set of points or vertices;
- Any affine transformation to the object is applied on each vertices;
- Transformation (affine) can be as follows:
  - Translation;
  - Scale;
  - Rotation;
  - Shear;
  - Reflection etc.

Transformation

- We’ll focus on affine transformation. Some characteristics of affine transformation are as follows:
  - parallel lines remain parallel after transformation.
  - angles between lines or distances between points may not be preserved;
  - ratios of distances between points on a straight line is preserved.

Transformation



Transformation

- Homogeneous coordinates are used for the representation of points;

- Suppose a point A (x, y, z, 1) has gone through some transformation (i.e., translation, rotation, scale, shear etc.) to A'(x', y' z', 1).

- This transformation can be written in terms of matrix multiplication M where, A' = MA and M is a 4X 4 matrix such as

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} M11 & M21 & M31 & M41 \\ M12 & M22 & M32 & M42 \\ M13 & M23 & M33 & M43 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous Coordinate System

- In OpenGL, 2D and 3D vertices, all internally treated as homogeneous vertices comprising four coordinates. Every column vector $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$

which is written as $(x, y, z, w)^T$ represents a homogeneous vertex if at least one of its elements is non zero.

Homogeneous Coordinates

- If the real number ‘a’ is nonzero, $(x, y, z, w)^T$ and $(ax, ay, az, aw)^T$ represent the same homogeneous vertex.
- In 3D, homogeneous coordinates are represented as $(x, y, z, 1.0)^T$ and 2D representation is $(x, y, 0.0, 1.0)^T$
- If ‘w’ becomes zero, if refers to some idealized “point at infinity”.
- Consider the sequence of points, (1,2, 0, 1) , (1, 2, 0, 0.1), (1, 2, 0, 0.01), (1,2 , 0, 0.001), (1, 2, 0, 0.0001),…….. (1,2, 0, 0) … forms an equation of a line moving toward infinity… represents homogeneous coordinates of line $2x = y$ in the projective plane

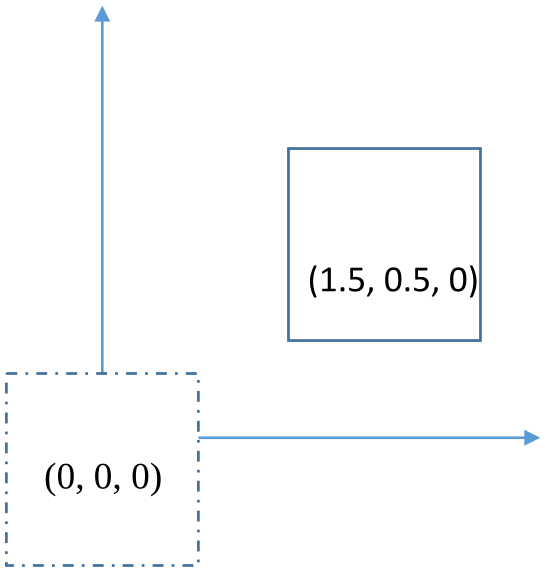
Correlation between Homogeneous and Cartesian Coordinates

- If $(x, y, z, w)^T$, $w \neq 0$, are the homogeneous coordinates of a point in the three-dimensional projective plane, the corresponding 3D Cartesian coordinates is $(x/w, y/w, z/w)^T$.
- If $(x,y,z)^T$ is a point in Cartesian coordinate, its homogeneous counterpart is $(x, y, z, 1)^T$

Translation

- Suppose, an object A has translated from origin (0, 0, 0) to A’
- As we know this transformation can be written in terms of matrix multiplication T where, A’ = TA and T is a 4X 4 matrix such as follows:

$$
\begin{array}{ccccc}
A' & & T & & A \\
\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} & = & \begin{bmatrix} 1 & 0 & 0 & 1.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & & \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}
\end{array}
$$



Translation

- Inverse of a translation can be found:
  - By inversing the matrix, i.e. $T^{-1}(t_x, t_y, t_z)$;
    - http://mathworld.wolfram.com/MatrixInverse.html
    - http://www.cg.info.hiroshima-cu.ac.jp/~miyazaki/knowledge/teche23.html
  - Translating back by –tx, -ty, -tz, i.e. T(-tx, -ty, -tz);
  - Both will result in same matrix; i.e.
    - $T^{-1}(t_x, t_y, t_z) = T(-tx, -ty, -tz) =$

$$\begin{bmatrix} 1 & 0 & 0 & -tx \\ 0 & 1 & 0 & -ty \\ 0 & 0 & 1 & -tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Identity Matrix

- Suppose a matrix is defined as mat4 M;
- An identity matrix is created with its diagonal elements all equal to ‘1’; Hence

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Multiplication of a matrix with its inverse results in identity matrix, i.e., $[M][M^{-1}] = I$, where ‘I’ is denoted as identity matrix.
- Multiplication of matrix ‘M’ by an Identity matrix ‘I’ results in the same matrix ‘M’.

Rotation

- 3D rotation requires an angle of rotation and an axis of rotation;

- Any of the coordinate axes (x, y, z) can be chosen as axis of rotation, R_x (around x –axis), R_y (around y axis) and R_z(around z axis);

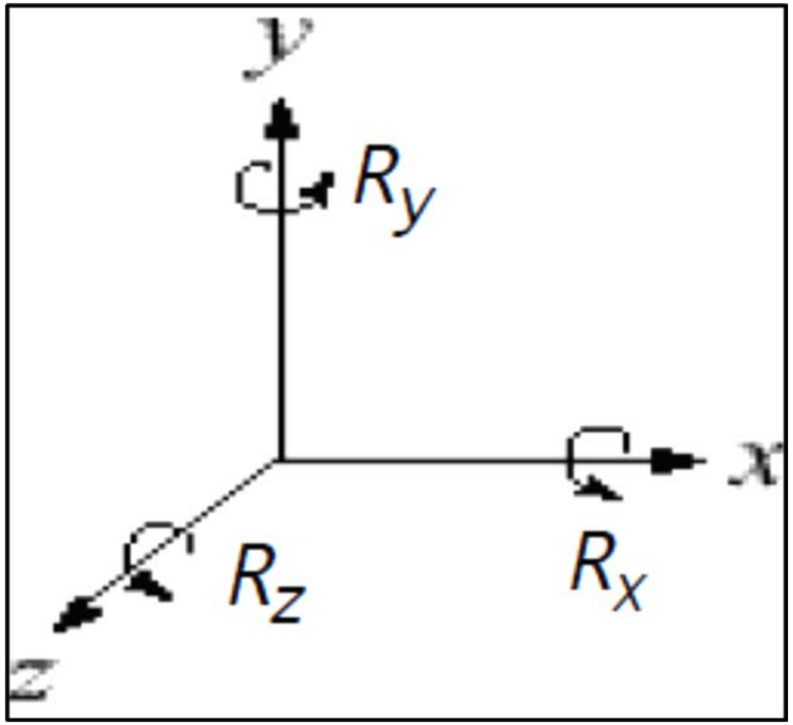
- Rotation axis can also be any arbitrary axis;

- Any desired rotation matrix can be constructed as a product of a number of individual rotation matrices, i.e. $R = R_z R_y R_x$

- Matrix multiplication is associative, but not commutative, i.e.,

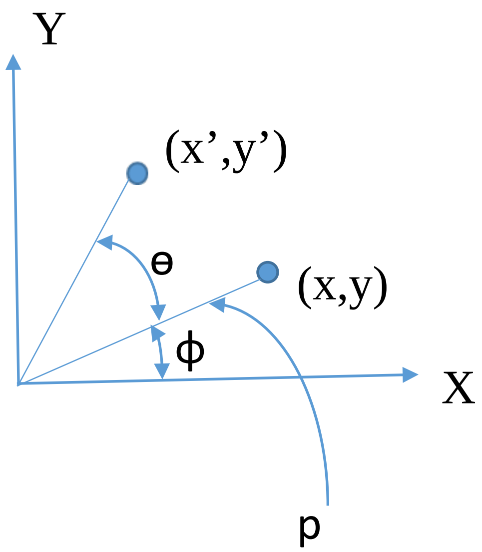
  $(R_z R_y)R_x = R_z(R_y R_x)$; but $R_z R_y \neq R_y R_z$

Rotation



Rotation

Rotation around z-axis: similar to rotation in 2D



$x = p\cos\phi$

$y = p\sin\phi$

$x' = p\cos(\theta+\phi) = p\cos\theta\cos\phi - p\sin\theta\sin\phi = x\cos\theta - y\sin\theta$

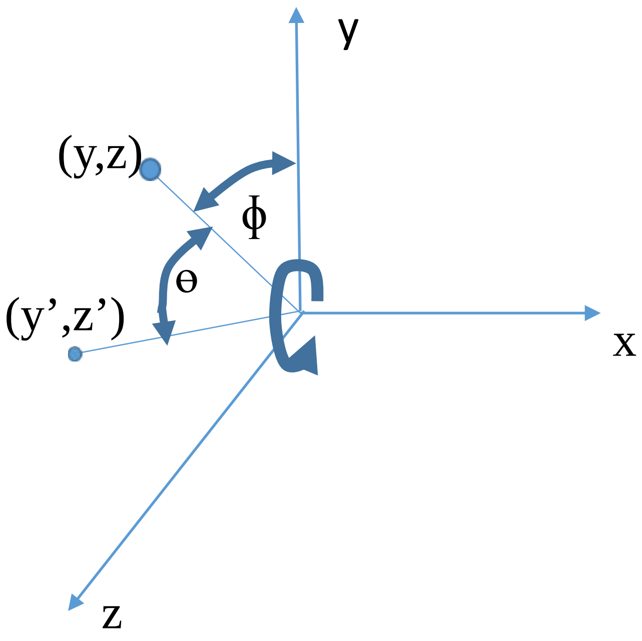
$y' = p\sin(\theta+\phi) = p\sin\theta\cos\phi + p\cos\theta\sin\phi = x\sin\theta + y\cos\theta$

$z' = z$

Now, the Rotation $R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotation

- Rotation about x-axis

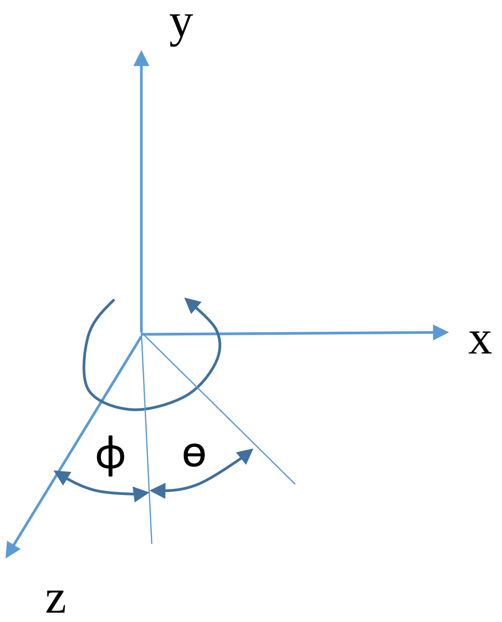


$y = p\cos\phi$
$z = p\sin\phi$
$x' = x$
$y' = p\cos(\theta+\phi) = p\cos\theta\cos\phi - p\sin\theta\sin\phi = y\cos\theta - z\sin\theta$
$z' = p\sin(\theta+\phi) = p\sin\theta\cos\phi + p\cos\theta\sin\phi = y\sin\theta + z\cos\theta$

Now, the Rotation $R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotation

Rotation about y axis



$z = p\cos\phi$
$x = p\sin\phi$
$z' = p\cos(\theta+\phi) = p\cos\theta\cos\phi - p\sin\theta\sin\phi = z\cos\theta - x\sin\theta$
$x' = p\sin(\theta+\phi) = p\sin\theta\cos\phi + p\cos\theta\sin\phi = z\sin\theta + x\cos\theta$

Now, the Rotation $R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotation

- A rotation by any angle (θ) can be undone by a subsequent rotation by ($-\theta$), hence, $R^{-1}(\theta) = R(-\theta)$;

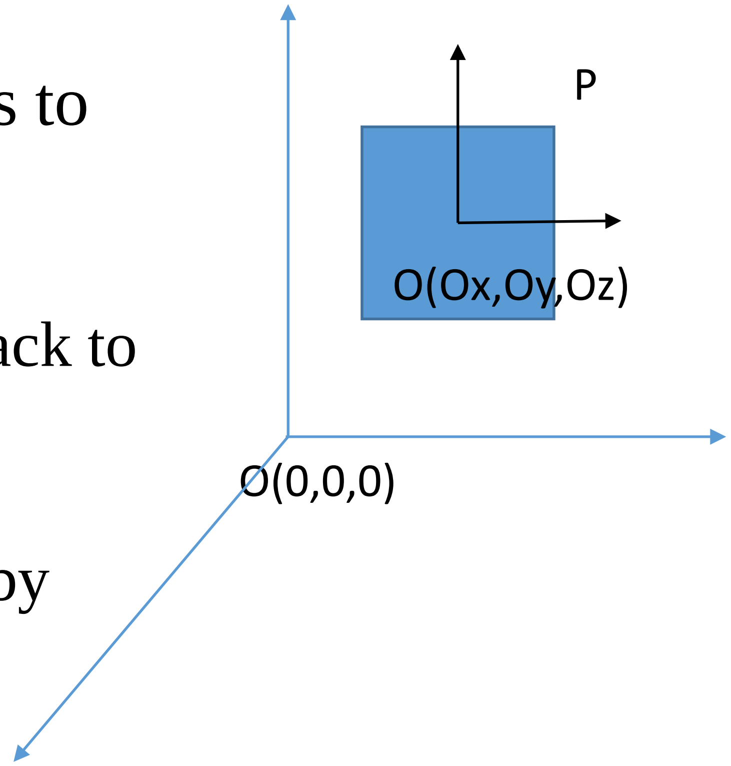
- Inverse of a rotation is equal to the transpose of the rotation, i.e.
  $R^{-1}(\theta) = R^T(\theta)$, as $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$;

Concatenation of Transformation

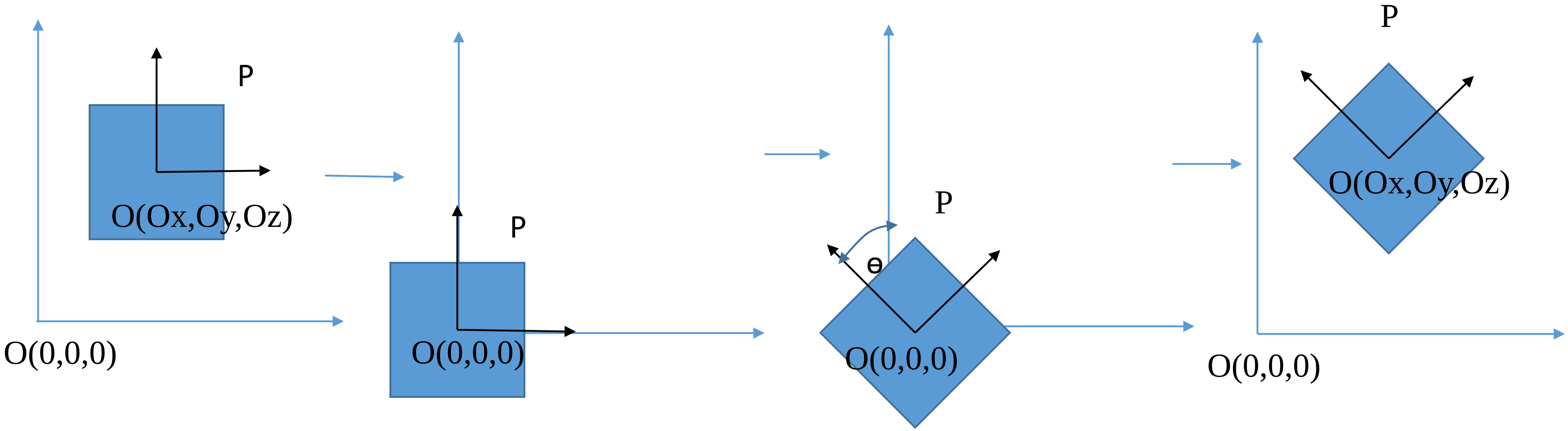
- Suppose, object P will rotate by ‘ɵ’ around ‘z’ axis.

- As it is centered away from the origin, it needs to undergo a number of transformation;

  - Firstly, translation by (-Ox, -Oy, -Oz) to get back to origin, i.e., T(-Ox, -Oy, -Oz)
  - Next, rotation around ‘z’ axis, i.e., $R_z(ɵ)$;
  - Lastly, translate back to the original position, by (Ox, Oy, Oz), i.e., T(Ox, Oy, Oz)



Concatenation of Transformation



Concatenation of Transformation

- Now the composite transformation matrix CTM can be written follows:

$$M= \begin{bmatrix} 1 & 0 & 0 & Ox \\ 0 & 1 & 0 & Oy \\ 0 & 0 & 1 & Oz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -Ox \\ 0 & 1 & 0 & -Oy \\ 0 & 0 & 1 & -Oz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & O_x - O_x \cos\theta + O_y \sin\theta \\ \sin\theta & \cos\theta & 0 & O_y - O_x \sin\theta - O_y \cos\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

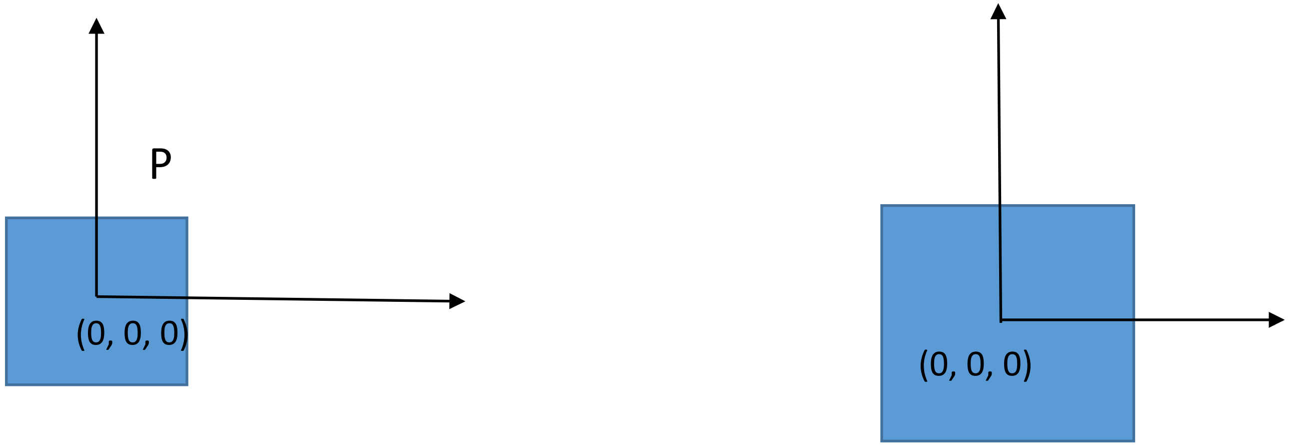
Transformation

- Translation and rotation are known as affine '***Rigid-body transformation***';

    - Rotation and/or translation can not alter the shape/ volume of an object;
    - Rotation and/or translation can alter object's position or orientation;

Scaling

- Scaling is an affine non-rigid transformation;
- Scaling is the process of increasing or decreasing the size of an object;



- A scaling matrix ‘$S(s_x, s_y, s_z)$’ with a fixed point scales along the coordinate axes, $P' = SP$, where $S =$

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- If $S_x = S_y = S_z$, then the scaling transformation is called ‘homogeneous'

Scaling

- A general form of a scaling of a matrix with respect to a fixed point $P(P_x, P_y, P_z)$ is as follows;

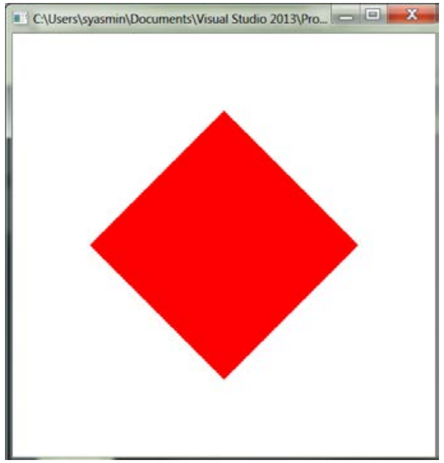
$$S(S_x, S_y, S_z, P) = \begin{bmatrix} 1 & 0 & 0 & Px \\ 0 & 1 & 0 & Py \\ 0 & 0 & 1 & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -Px \\ 0 & 1 & 0 & -Py \\ 0 & 0 & 1 & -Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & -PxSx + Px \\ 0 & Sy & 0 & -PySy + Py \\ 0 & 0 & Sz & -PzSz + Pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Inverse of a scaling matrix can be obtained from the reciprocals of the scaling factors, i.e.,

$$S^{-1}(S_x, S_y, S_z) = S(1/S_x, 1/S_y, 1/S_z)$$

Transforming the Rectangle in RectangleShader

- Let's scale (half of its size) and rotate (around z-axis by 45 degree) the rectangle to make it look like this:



- This can be performed in application or shader;
- Let's take a look how transformation can work in application first;
- There can be several ways to do this;
- Next, we'll take a look how transformation done in shader.

Transforming in Application

```
GLfloat rect_vertices[4][4] = { -0.9f, -0.9f, 0.0, 1.0,
                                 0.9f, -0.9f, 0.0, 1.0,
                                 0.9f, 0.9f, 0.0, 1.0,
                                -0.9f, 0.9f, 0.0, 1.0};
```

angle 45 degree in radian

Rotation around z axis

```
for (int i = 0; i < 4; i++){
        GLfloat vert_x = cos(angle)*rect_vertices[i][0] -sin(angle)*rect_vertices[i][1];
        GLfloat vert_y = sin(angle)*rect_vertices[i][0] + cos(angle)*rect_vertices[i][1];
        rect_vertices[i][0] = vert_x*0.5;
        rect_vertices[i][1] = vert_y*0.5;
}
```

applies scaling

```
glBufferData(GL_ARRAY_BUFFER, sizeof(rect_vertices), rect_vertices, GL_STATIC_DRAW);
```

Transformation in Shader

- Vertex shader looks like follows with two uniform variables ‘theta’ and ‘scale’

```
#version 430 core
in vec4 vPosition;
uniform float theta;
uniform float scale;
float angle = radians(theta);
mat4 r = mat4( cos(angle), -sin(angle), 0.0, 0.0,
               sin(angle), cos(angle),   0.0, 0.0,
                0.0,             0.0,        1.0, 0.0,
                0.0,             0.0,        0.0, 1.0);
mat4 ss = mat4( scale, 0.0, 0.0, 0.0,
               0.0, scale, 0.0, 0.0,
               0.0,  0.0,   1.0, 0.0,
               0.0,  0.0,   0.0, 1.0 );
void main () {
     gl_Position = r*ss*vPosition;
}
```

Transformation in Shader

- In application ‘Theta’ and ‘Scale’ are defined as follows:

```
GLfloat theta;
GLfloat scale;
GLfloat Theta = 45.0;
GLfloat Scale = 0.5;
```

- Location of uniform variables queried:

```
theta = glGetUniformLocation(program, "theta");
scale = glGetUniformLocation(program, "scale");
```

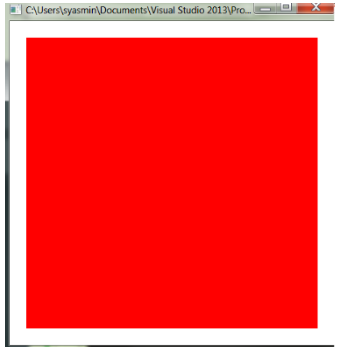
- Assign the values of the uniform variables before drawing the rectangle;

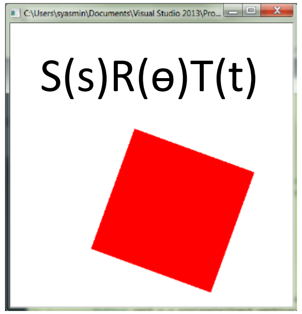
```
glUniform1f(theta, Theta);
glUniform1f(scale, Scale);
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

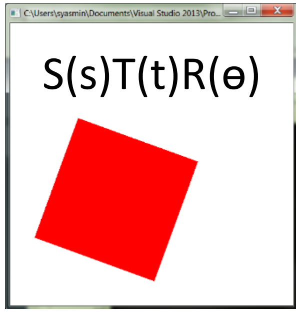
Transformation

- Order matters!
  - Translation, followed by rotation is not equivalent to rotation followed by translation;
  - Looks smaller as scale has done!

  - $S(s)R(\theta)T(t) \neq S(s)(T(t)R(\theta))$

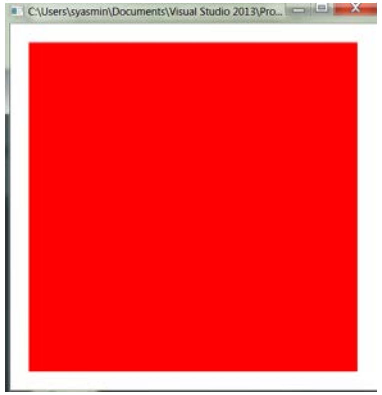


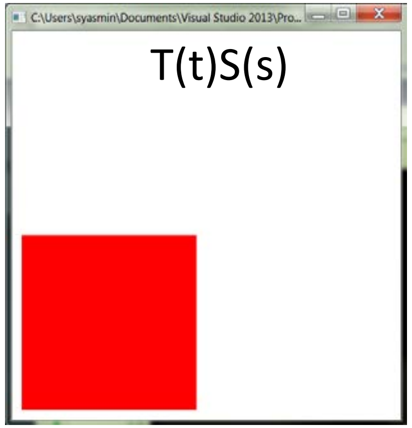


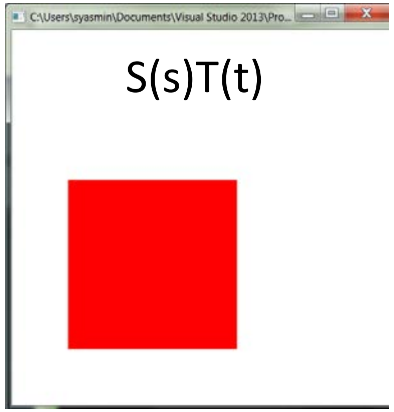


Transformation

- S(s)T(t) ≠ T(t)S(s)

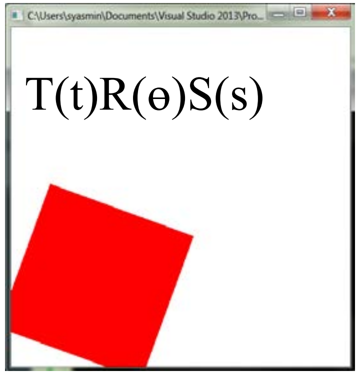


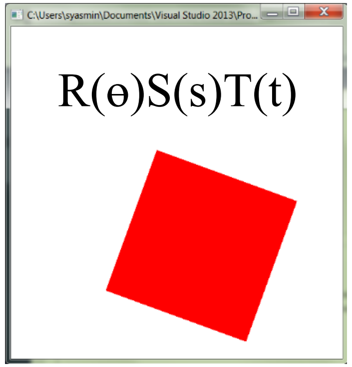




Transformation

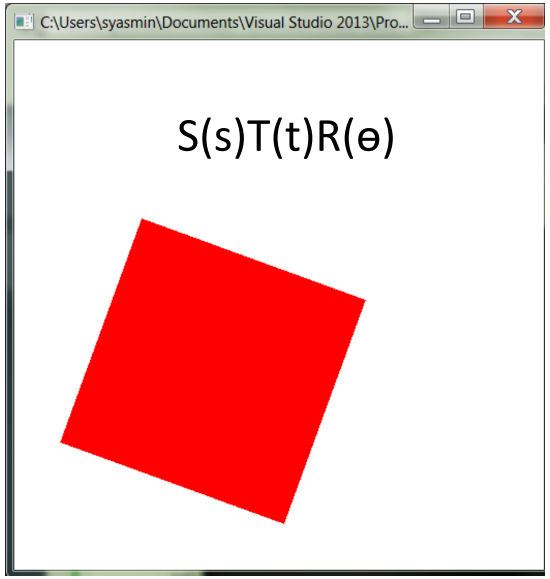
- Let’s apply more transformation;
  - $T(t)R(\theta)S(s) \neq R(\theta)S(s)T(t)$

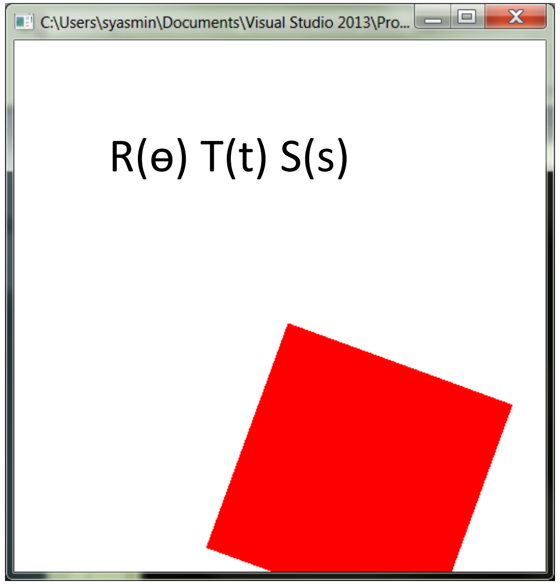




Transformation

- S(s)T(t)R(ɵ) ≠ R(ɵ)T(t)S(s)





Using GLM Library

- OpenGL Mathematics Library: A header only library
- Download from the following website:
  - http://glm.g-truc.net/0.9.7/index.html

- **For windows**
  - After extracting, you just need to place the GLM folder (that contains header files) to
  - …..\Microsoft Visual Studio 14.0\VC\include\glm ← Visual Studio 2015
  - …...\Microsoft Visual Studio\2017\Professional\VC\Tools\MSVC\14.10.25017\include\glm

- **For Linux**
  - Should already be installed if you use the OpenGL installation commands (Lecture 1 Week 1)

Using GLM Library

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define GLM_FORCE_RADIANS

#include <glm/mat4x4.hpp>
#include <glm/gtc/matrix_transform.hpp>
```

Using GLM Library

- glm::mat4 model_matrix = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f, 0.5, 1.0));

- model_matrix = glm::rotate(model_matrix, -45.0f, glm::vec3(0.0f, 0.0f, 1.0f));

- model_matrix = glm::translate(model_matrix, glm::vec3(0.5f, 0.5f, 0.0f));