LEC15: Concatenation of Affine Transformations

Ku-Jin Kim

School of Computer Science & Engineering

Kyungpook National University

Contents

Concatenation

- We can form arbitrary affine transformation matrices by multiplying together rotation, translation, and scaling matrices
- Because the same transformation is applied to many vertices, the cost of forming a matrix $\mathrm{M}=\mathrm{ABCD}$ is not significant compared to the cost of computing $\mathrm{M}\mathbf{p}$ for many vertices $\mathbf{p}$
- The difficult part is how to form a desired transformation from the specifications in the application

Order of Transformations

- Note that matrix on the right is the first applied
- Mathematically, the following are equivalent

$$\mathbf{p}' = \mathbf{ABCp} = \mathbf{A}(\mathbf{B}(\mathbf{Cp})) = (\mathbf{ABC})\mathbf{p}$$

- Note many references use row vectors to represent points. In terms of row vectors

$$\mathbf{p}'^{\mathrm{T}} = \mathbf{p}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$$

LEC15.1_concatenation.c (1)

```
static char* vsSource = "#version 130 ₩n₩
in vec4 aPosition; ₩n₩
in vec4 aColor; ₩n₩
out vec4 vColor; ₩n₩
uniform mat4 u_rotate;  ₩n₩
uniform float u_scale_factor; ₩n₩
uniform vec2 u_trans_vec; ₩n₩
void main(void) { ₩n₩
  mat4 scalemat = mat4(u_scale_factor); ₩n₩
  scalemat[3][3] = 1.0; ₩n₩
  mat4 transmat = mat4(1.0); ₩n₩
  transmat[3][0] = u_trans_vec[0]; ₩n₩
  transmat[3][1] = u_trans_vec[1]; ₩n₩
  gl_Position = transmat*u_rotate*aPosition; ₩n₩
//  gl_Position = u_rotate*transmat*aPosition; ₩n₩
//  gl_Position = scalemat*transmat*u_rotate*aPosition; ₩n₩
  vColor = aColor; ₩n₩
}";
```

LEC15.1_concatenation.c (2)

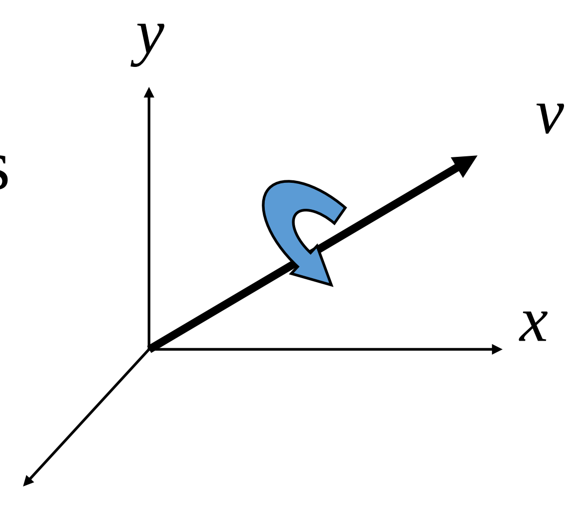
```
// rotation about x-axis
m[0] = 1.0; m[4] = 0.0;     m[8] = 0.0;      m[12] = 0.0;
m[1] = 0.0; m[5] = cos(t);  m[9] = -sin(t);  m[13] = 0.0;
m[2] = 0.0; m[6] = sin(t);  m[10] = cos(t);  m[14] = 0.0;
m[3] = 0.0; m[7] = 0.0;     m[11] = 0.0;     m[15] = 1.0;

loc = glGetUniformLocation(prog, "u_rotate");
glUniformMatrix4fv(loc, 1, GL_FALSE, m);

float scale_factor = 1.5;
loc = glGetUniformLocation(prog, "u_scale_factor");
glUniform1f(loc, scale_factor);

float trans_vec[] = { 0.5, 0.5 };
loc = glGetUniformLocation(prog, "u_trans_vec");
glUniform2fv(loc, 1, trans_vec);
```

General Rotation About the Origin

- A rotation by θ about an arbitrary axis can be decomposed into the concatenation of rotations about the *x*, *y*, and *z* axes
  - Note that rotations do not commute
  - We can use rotations in another order but with different angles

$$\mathbf{R}(\theta) = \mathbf{R}_z(\theta_z)\, \mathbf{R}_y(\theta_y)\, \mathbf{R}_x(\theta_x)$$

θ_x θ_y θ_z are called the Euler angles

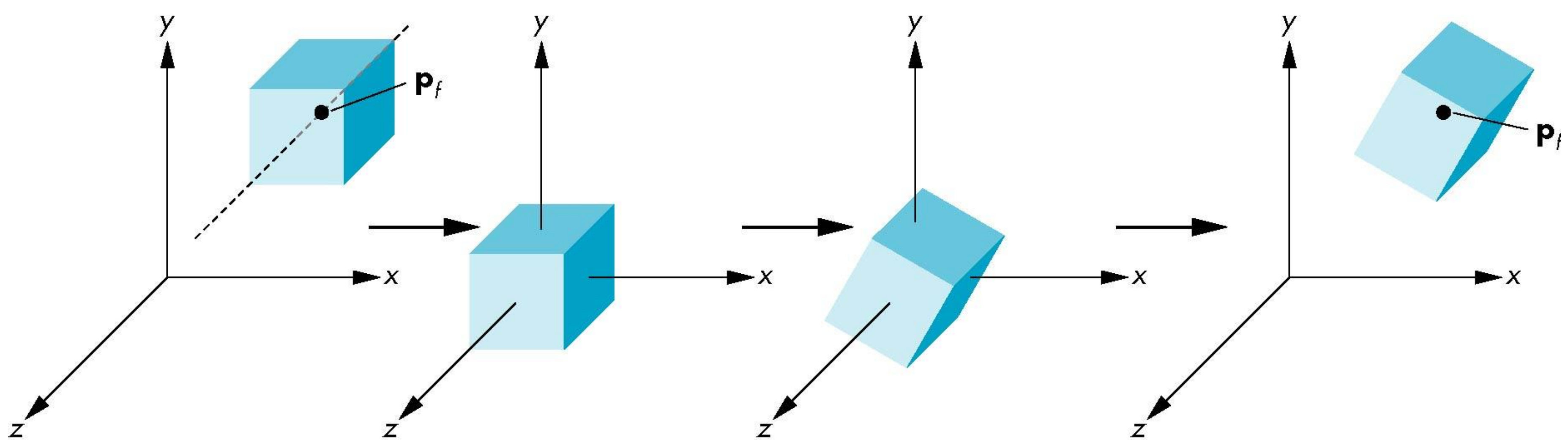


LEC15.2_EulerAngle.c

```
static char* vsSource = "#version 130 ₩n₩
in vec4 aPosition; ₩n₩
in vec4 aColor; ₩n₩
out vec4 vColor; ₩n₩
uniform mat4 umx;  ₩n₩
uniform mat4 umy;  ₩n₩
uniform mat4 umz;  ₩n₩
void main(void) { ₩n₩
  gl_Position = umz*umy*umx*aPosition; ₩n₩
  vColor = aColor; ₩n₩
}";
```

Rotation About a Fixed Point other than the Origin

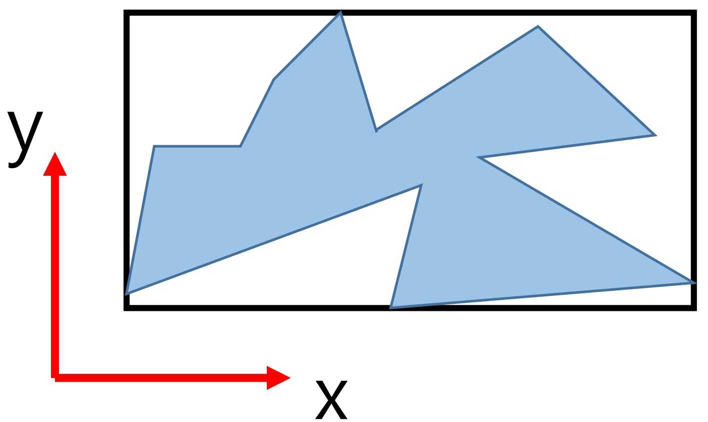
- Steps:
  1) Move fixed point $\mathbf{p}_f$ to origin
  2) Rotate
  3) Move fixed point back
- $\mathbf{M} = \mathbf{T}(p_f)\,\mathbf{R}(\theta)\,\mathbf{T}(-p_f)$

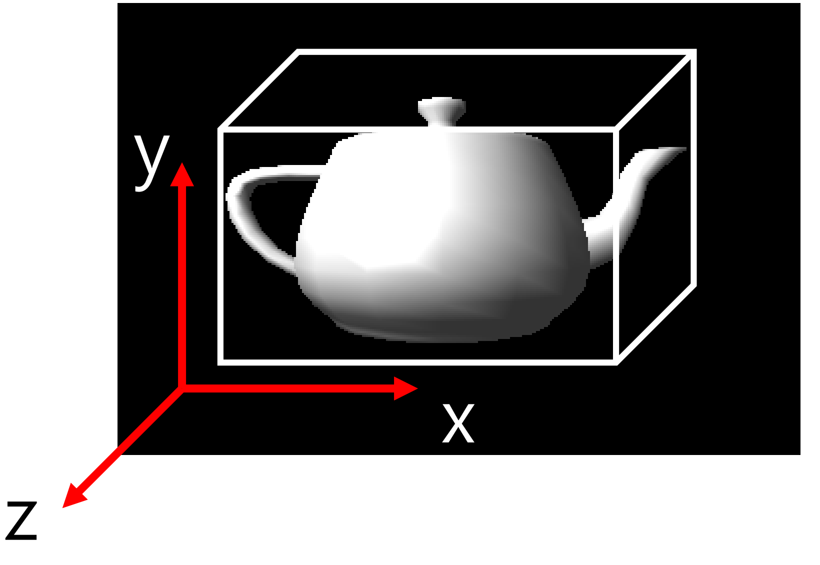


How Decide the Center of Object?

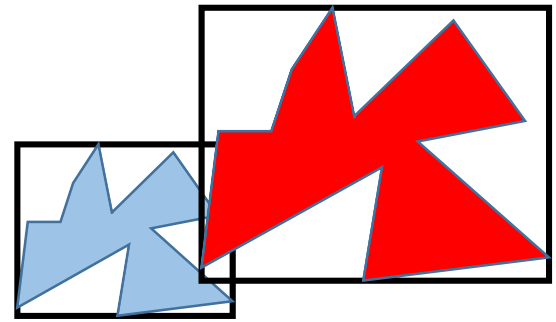
- Centroid computing
  - $C = \frac{1}{n}\sum_{i=0}^{n-1} v_i$

- Center of bounding box





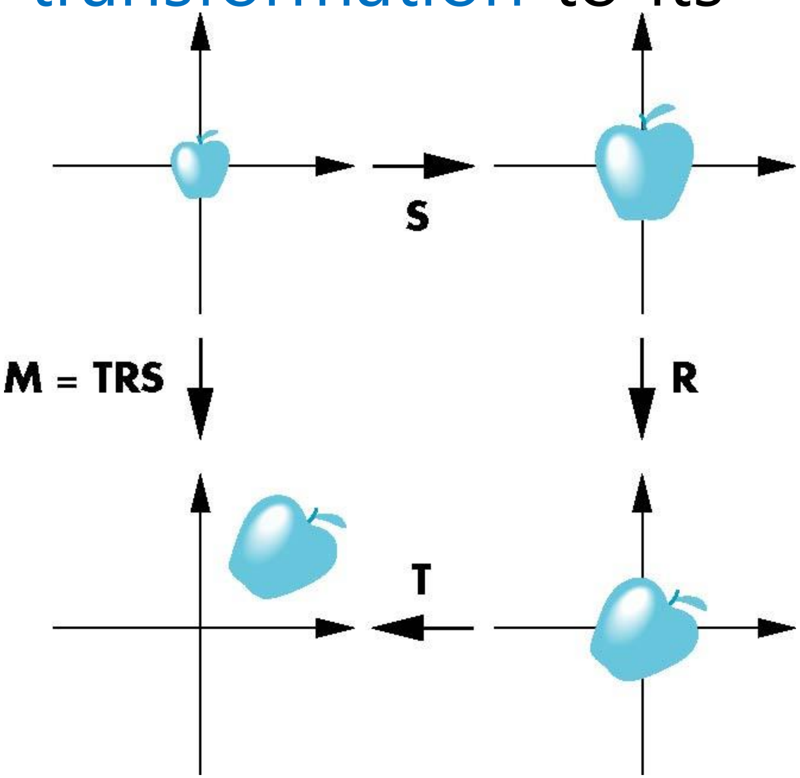
Axis-Aligned Bounding box



- Useful for collision detection
- 2D bounding box
  - the smallest rectangle, aligned with the x and y-axes, that contains the polygon.
  - computed by finding the minimum and maximum of both the x and y values for every vertex of the polygon.
- 3D bounding box
  - the smallest cuboid, aligned with the x, y, and z-axes, that contains the 3D polygon mesh.

Instancing

- In modeling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size
- We apply an instance transformation to its vertices to decide
  - Size
    - by scaling
  - Orientation
    - by rotation
  - Location
    - by translation



HW#15 Concatenation of Transformations (20 points) Due date: This Friday 6pm

```
GLfloat vertices[] = {
  0.5, 0.8, 0.0, 1.0, // 0
  0.3, 0.3, +0.2, 1.0, // 1
  0.7, 0.3, +0.2, 1.0, // 2
  0.7, 0.3, -0.2, 1.0, // 3
  0.3, 0.3, -0.2, 1.0, // 4
};
```

```
GLfloat colors[] = {
  1.0, 0.0, 0.0, 1.0,  //0
  0.0, 1.0, 0.0, 1.0,  //1
  0.0, 0.0, 1.0, 1.0,  //2
  1.0, 0.0, 1.0, 1.0,  //3
  1.0, 1.0, 0.0, 1.0   //4
};
```

```
GLushort indices[] = {
      0, 1, 2,
      2, 3, 0,
      4, 0, 3,
      1, 0, 4,
      2, 3, 1,
      3, 4, 1
};
```

Requirement (Score will be deducted if the requirement is not satisfied)

- Step 1. Construct the object by the above arrays.
- Step 2. Compute the centroid of the object.
- Step 3. Translate the object for the centroid to be at the origin.
- Step 4. Rotate the object by Euler angles: $\theta_x = 30^o$, $\theta_y = 30^o$, $\theta_z = 30^o$, by using the rotation matrix in this Lecture. The rotation angles must be used as radians in cos, sin functions in C-language.
- Step 5. Draw the object after translating the rotated object back to it's original position by using the centroid

Execution result of HW#15 must be