LEC13: Transformation-Part1

Ku-Jin Kim

School of Computer Science & Engineering

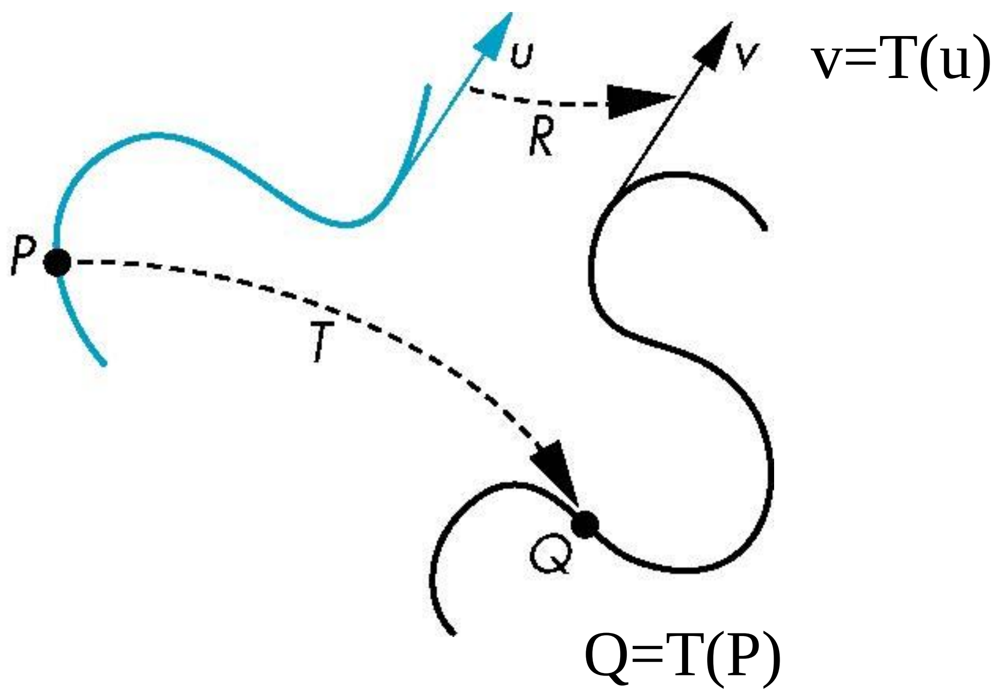
Kyungpook National University

Contents

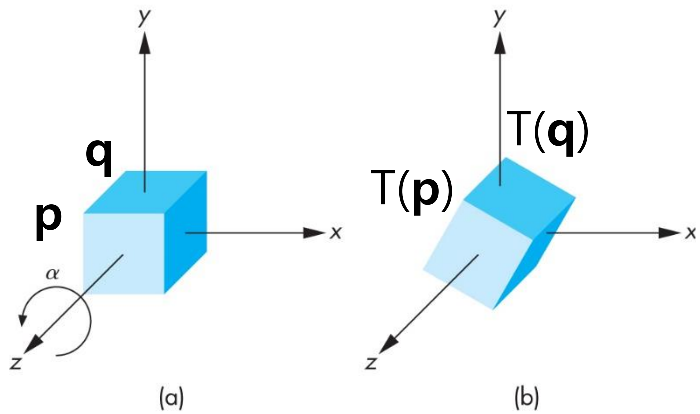
- Transformations – Part 1
  - Affine Transformations
  - Translation by a vector
  - Translation by a matrix
- Program Examples

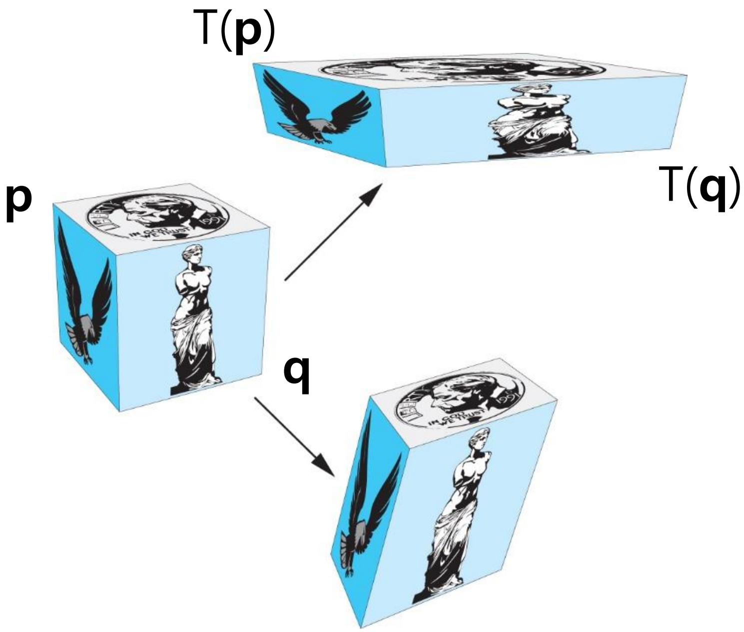
General Transformations

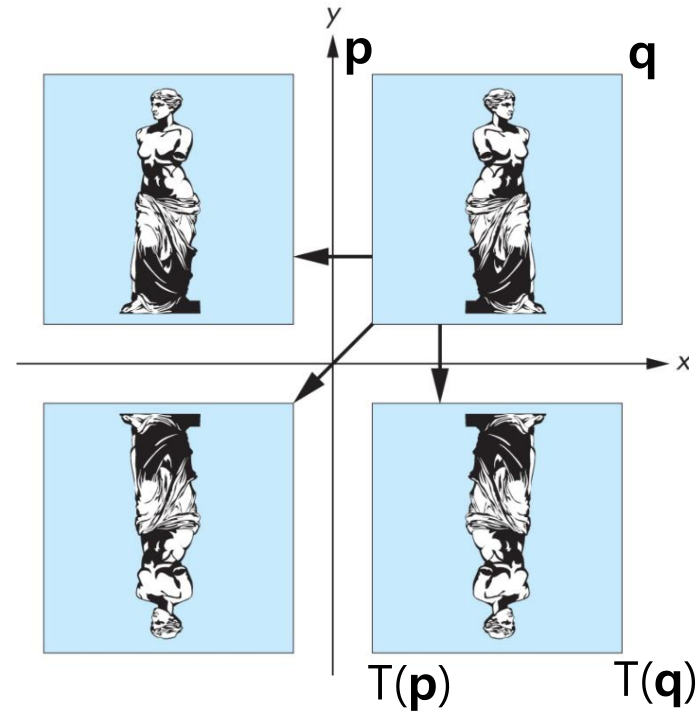
- Transformation
  - is a function that takes a point (or vector) and maps it into another point (or vector)

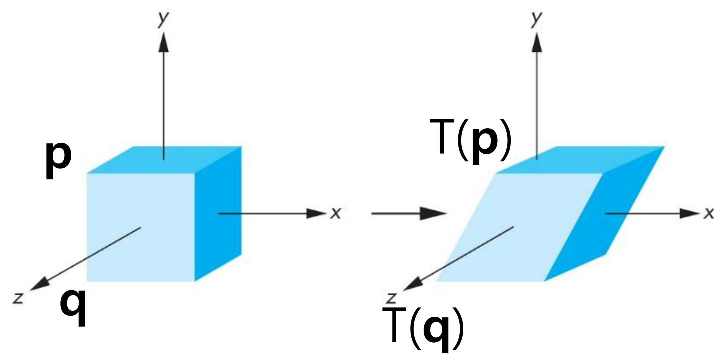


Transformation Examples









Affine Transformations (1)

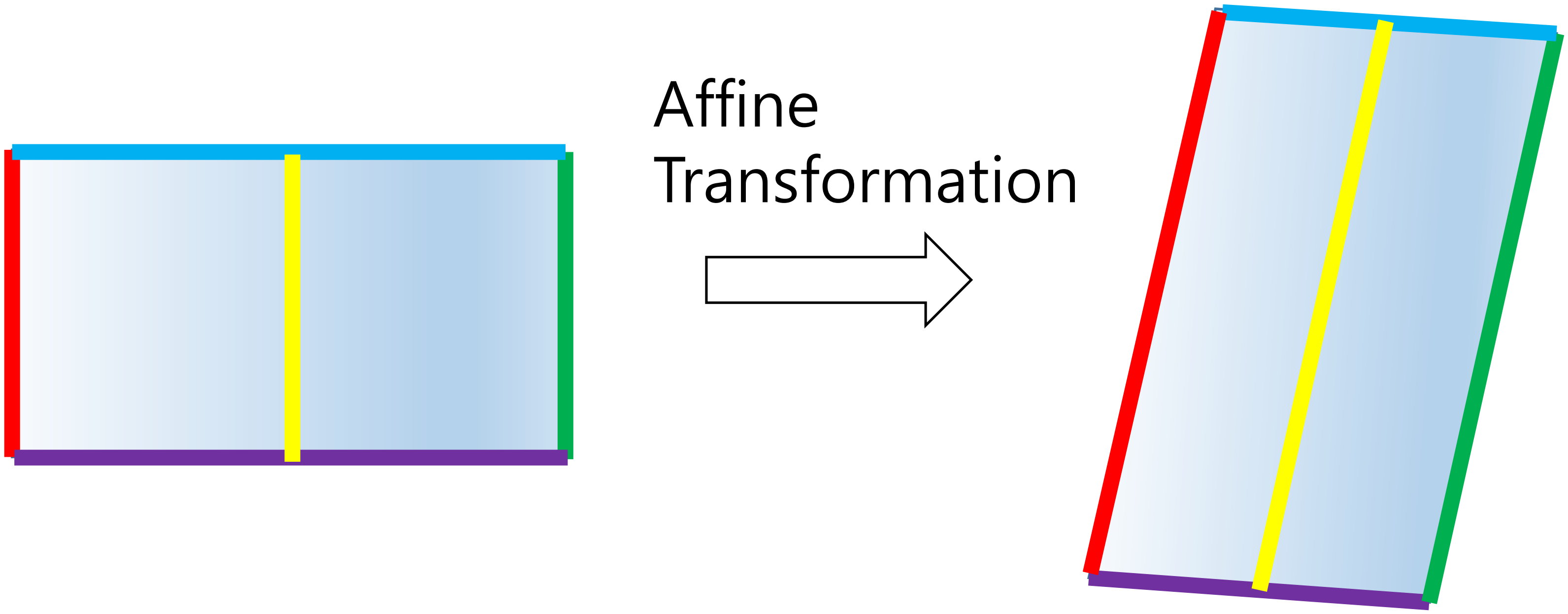
- Definition: Any transformation that preserves colinearity and ratios of distances
  - Ex) all points lying on a line initially still lie on a line after transformation
  - Ex) the midpoint of a line segment remains the midpoint after transformation
  - Ex) sets of parallel lines remain parallel after transformation
- The matrix M for frame changes is 4 x 4 and specifies an affine transformation in homogeneous coordinates

Affine Transformations (2)

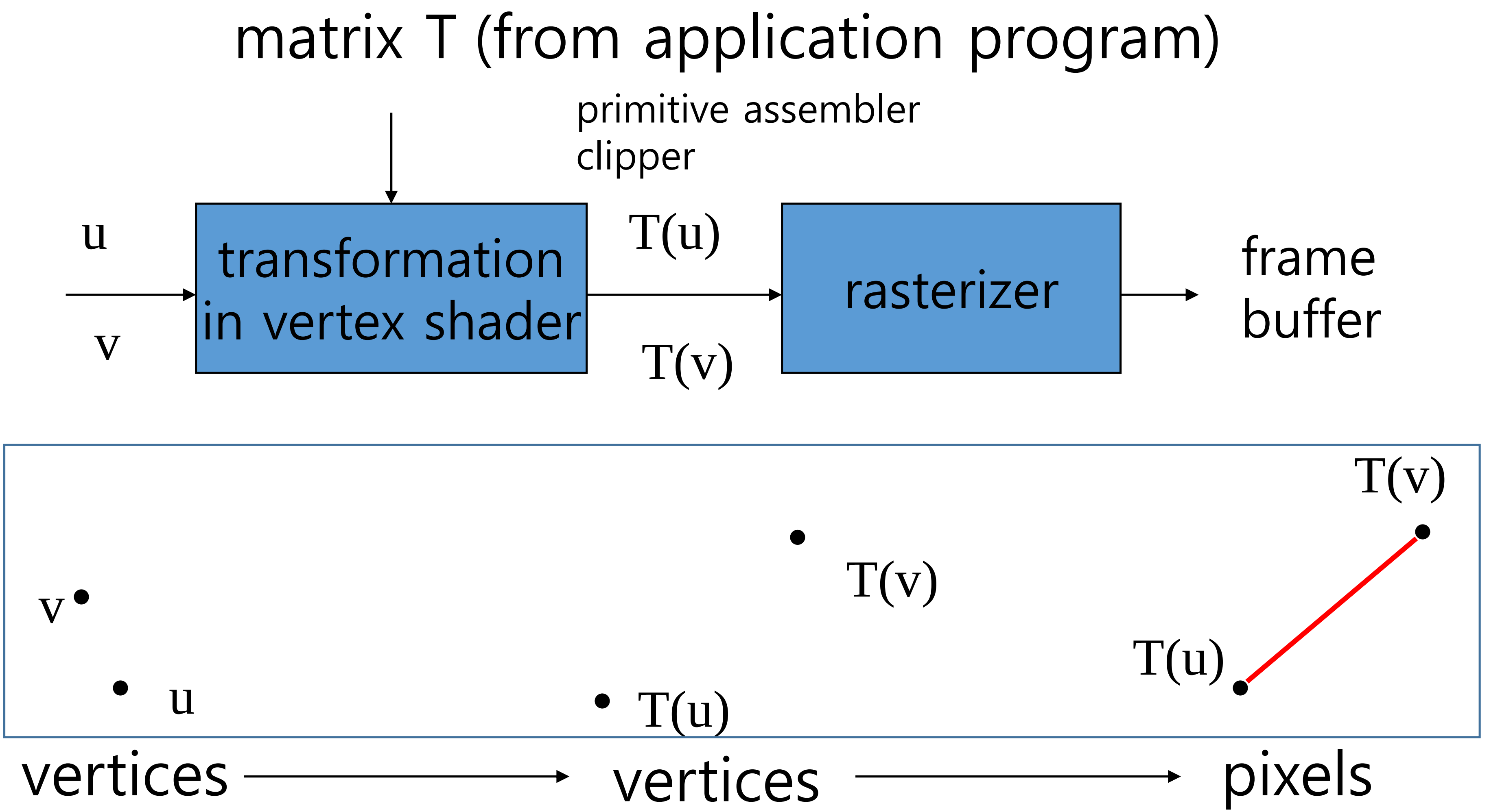
- Characteristic of many physically important transformations
  - Rigid body transformations: rotation, translation
  - Scaling, shear
- We need only transform endpoints of line segments and let implementation draw line segment between the transformed endpoints by line preserving property

Affine Transformations (3)

- Preserving lines
- Preserving parallel lines
- Preserving distance ratios



Pipeline Implementation

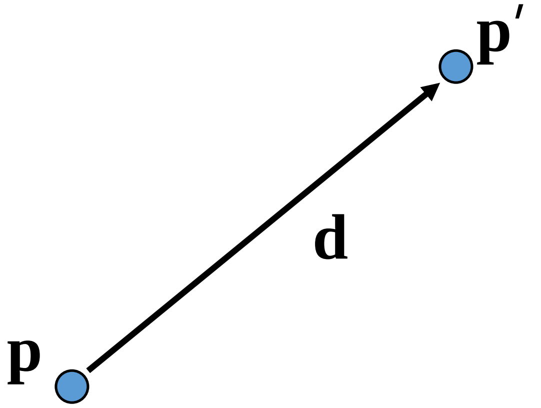


Notation

- We will be working with both coordinate-free representations of transformations and representations within a particular frame
- P, Q, R : points in an affine space
- u, v, w : vectors in an affine space
- α, β, γ : scalars
- **p**, **q**, **r** : representations of points
  - array of 4 scalars in homogeneous coordinates
- **u**, **v**, **w**: representations of vectors
  - array of 4 scalars in homogeneous coordinates

Translation

- Move (translate, displace) a point to a new location



- Displacement determined by a vector d
  - Three degrees of freedom
  - $\mathbf{p}' = \mathbf{p} + \mathbf{d}$

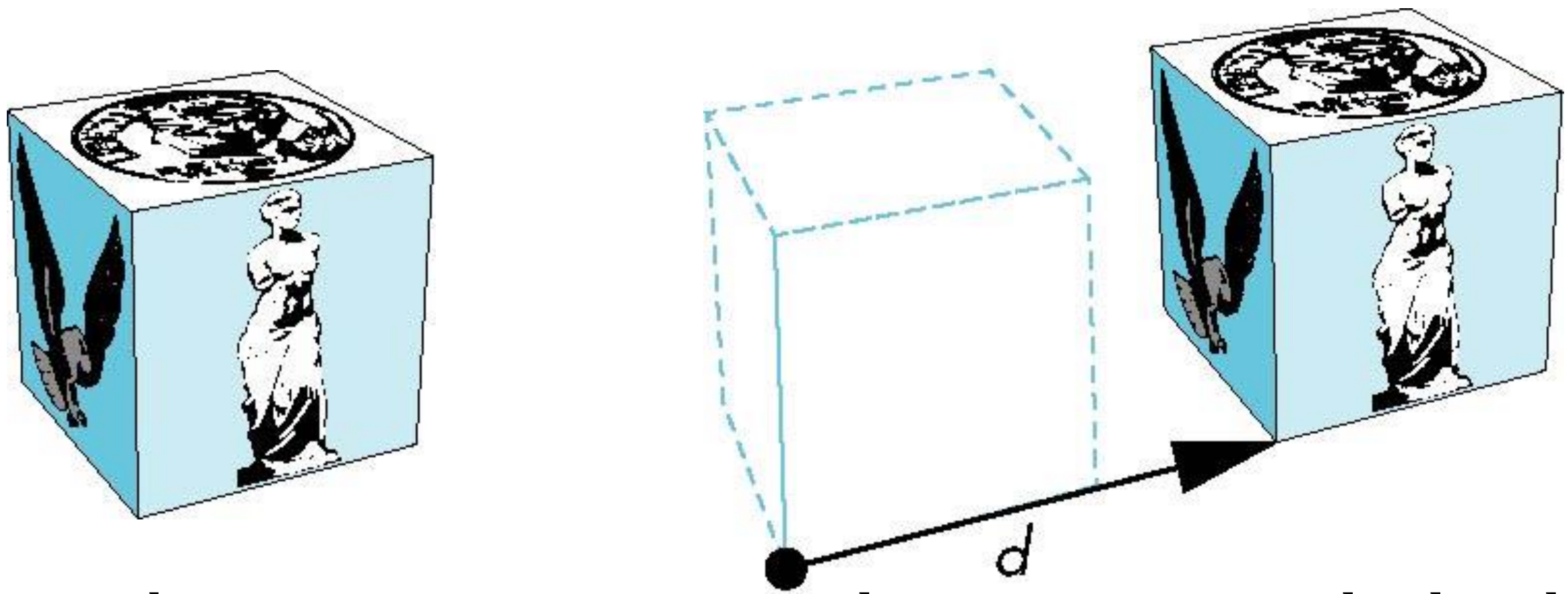
Translation Using Representations

- Using the homogeneous coordinate representation in some frame
  - $\mathbf{p} = [x\ y\ z\ 1]^T$
  - $\mathbf{p}' = [x'\ y'\ z'\ 1]^T$
  - $\mathbf{d} = [d_x\ d_y\ d_z\ 0]^T$
- Hence $\mathbf{p}' = \mathbf{p} + \mathbf{d}$ or
  - $x' = x + d_x$
  - $y' = y + d_y$
  - $z' = z + d_z$

note that this expression is in four dimensions and expresses point = vector + point

How many ways?

- Although we can move a point to a new location in infinite ways, when we move many points there is usually only one way



object

translation: every point displaced by same vector

LEC13.1_translate_DrawElements

```
GLfloat vertices[] = {
        -0.2, -0.2, -0.2, 1.0, // 0
        -0.2, -0.2, +0.2, 1.0, // 1
        -0.2, +0.2, -0.2, 1.0, // 2
        -0.2, +0.2, +0.2, 1.0, // 3
        +0.2, -0.2, -0.2, 1.0, // 4
        +0.2, -0.2, +0.2, 1.0, // 5
        +0.2, +0.2, -0.2, 1.0, // 6
        +0.2, +0.2, +0.2, 1.0, // 7
};
```

```
GLushort indices[] = {
        0, 4, 6,
        6, 2, 0,
        4, 5, 7,
        7, 6, 4,
        1, 3, 7,
        7, 5, 1,
        0, 2, 3,
        3, 1, 0,
        2, 6, 7,
        7, 3, 2,
        0, 1, 5,
        5, 4, 0,
};
```

```
glDrawElements(GL_TRIANGLES, 12 * 3, GL_UNSIGNED_SHORT, indices);
```

glDrawElements (1)

void **glDrawElements**(GLenum *mode*, GLsizei *count*, GLenum *type*, const void * *indices*);

mode

Specifies what kind of primitives to render. Symbolic constants `GL_POINTS`, `GL_LINE_STRIP`, `GL_LINE_LOOP`, `GL_LINES`, `GL_LINE_STRIP_ADJACENCY`, `GL_LINES_ADJACENCY`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`, `GL_TRIANGLES`, `GL_TRIANGLE_STRIP_ADJACENCY`, `GL_TRIANGLES_ADJACENCY` and `GL_PATCHES` are accepted.

count

Specifies the number of elements to be rendered.

type

Specifies the type of the values in *indices*. Must be one of `GL_UNSIGNED_BYTE`, `GL_UNSIGNED_SHORT`, or `GL_UNSIGNED_INT`.

indices

Specifies a pointer to the location where the indices are stored.

glDrawElements (2)

- When glDrawElements is called, it uses count sequential elements from an enabled array, starting at indices to construct a sequence of geometric primitives.
- mode specifies what kind of primitives are constructed and how the array elements construct these primitives. If more than one array is enabled, each is used.

LEC13.1_translate_DrawElements.c

- Replace glDrawArrays with glDrawElements

LEC13.2_translate_vec.c (1)

```
static char* vsSource = "#version 120 ₩n₩
attribute vec4 aPosition; ₩n₩
attribute vec4 aColor; ₩n₩
varying vec4 vColor; ₩n₩
uniform vec4 udvec; ₩n₩
void main(void) { ₩n₩
  gl_Position = aPosition + udvec; ₩n₩
  vColor = aColor; ₩n₩
}";
```

LEC13.2_translate_vec.c (2)

```
GLfloat d[4] = {2.0, 1.0, 0.0, 0.0};

loc = glGetUniformLocation(prog, "udvec");
glUniform4fv(loc, 1, d);

glDrawElements(GL_TRIANGLES, 12 * 3, GL_UNSIGNED_SHORT,
indices);
```

LEC13.3_translate_vec_animate.c (3)

```
GLfloat d[4] = {0.2, 0.5, 0.0, 0.0};
GLfloat dvec[4];

dvec[0] = t*d[0];
dvec[1] = t*d[1];
dvec[2] = t*d[2];
dvec[3] = t*d[3];

loc = glGetUniformLocation(prog, "udvec");
glUniform4fv(loc, 1, dvec);

glDrawElements(GL_TRIANGLES, 12 * 3, GL_UNSIGNED_SHORT,
indices);
```

Translation Matrix

- We can also express translation using a 4 x 4 matrix T in homogeneous coordinates **p**′=T**p**, where

- $T = T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together

LEC13.4_translate_matrix.c (1)

```
static char* vsSource = "#version 120 ₩n₩
attribute vec4 aPosition; ₩n₩
attribute vec4 aColor; ₩n₩
varying vec4 vColor; ₩n₩
uniform mat4 utranslate; ₩n₩
void main(void) { ₩n₩
  gl_Position = utranslate*aPosition; ₩n₩
  vColor = aColor; ₩n₩
}";
```

LEC13.4_translate_matrix.c (2)

```
GLfloat d[4] = { 0.2, 0.3, 0.0, 0.0};
GLfloat m[16];

void mydisplay(void) {
        GLuint loc;
        glClearColor(0.7f, 0.7f, 0.7f, 1.0f); // gray
        glClear(GL_COLOR_BUFFER_BIT);
        m[0] = 1.0; m[4] = 0.0; m[8] = 0.0; m[12] = d[0];
        m[1] = 0.0; m[5] = 1.0; m[9] = 0.0; m[13] = d[1];
        m[2] = 0.0; m[6] = 0.0; m[10] = 1.0; m[14] = d[2];
        m[3] = 0.0; m[7] = 0.0; m[11] = 0.0; m[15] = 1.0;

        loc = glGetUniformLocation(prog, "utranslate");
        glUniformMatrix4fv(loc, 1, GL_FALSE, m);
        glDrawElements(GL_TRIANGLES, 12 * 3, GL_UNSIGNED_SHORT, indices);
        glFlush();
        glutSwapBuffers();
}
```

HW #13 Translation using a trigonometric function

- Due date: This Friday 6:00pm
- Construct a pyramid shape model M:
  - M must be a polyhedron with 5 vertices and 6 faces (triangles).
  - Each vertex has different color.
  - The longest distance between any two vertices must be smaller than 0.5. (Don't make an object too big to see their movement)
- Use glDrawElements for drawing.
- Implement a program for the model M to translate along (t, sin(5.0*t)/2.0, 0, 0) by using a matrix. t must be used as follows, where the initial value is -1.0f;

```
t += 0.0001f;
if (t > 1)
      t = -1.0f;
```

- For using sin function, you need '#include <math.h>'.
- Submit .c file through LMS. Please make .c file name as your student number and name: ex) 2000232_홍길동.c
- HW running example will be explained. Please watch until the end of the video.