# CG Practice 3

COLLEGE OF COMPUTING

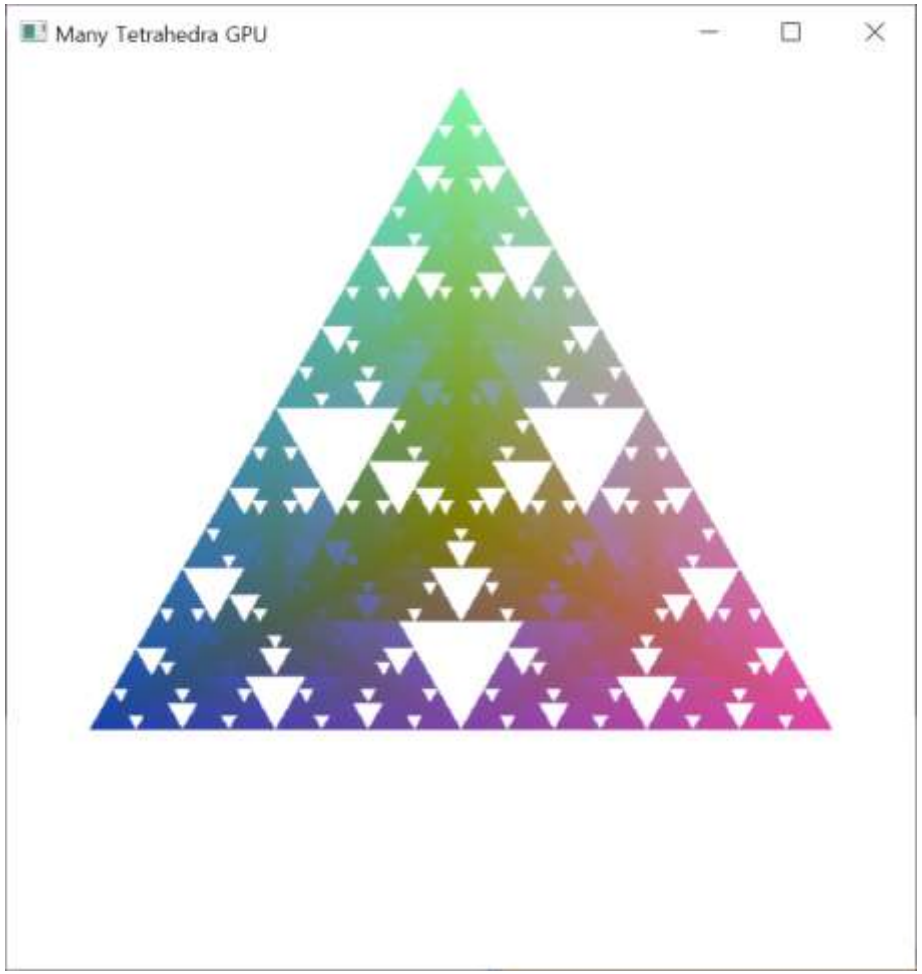
HANYANG ERICA CAMPUS

Q YOUN HONG (홍규연)

# Previously…



Shader들을 이용하여 간단한 그래픽스 프로그램 작성



# GLSL 문법(Syntax)

# GLSL Syntax Overview



- GLSL – Shader를 작성하는데 쓰는 언어로 C와 유사한 문법 (syntax)를 사용함
- GLSL에서 사용할 수 없는 것
  - 포인터 (Pointers)
  - 재귀함수 (Recursion)
  - 동적 할당 (Dynamic memory allocation)
- GLSL에 추가된 것
  - 새로운 타입들 – Built-in vector, matrix, sampler types
  - 생성자 (constructors)
  - 다양한 수학 함수들
  - 입력/출력 qualifiers

# GLSL 문법



- GLSL – 전처리문 (preprocessor) 있음

```
#version 430

#ifdef FAST_EXACT_METHOD
  FastExact();
#else
  SlowApproximate();
#endif

// ... many others
```

- 모든 Shader는 main() 함수에서 시작

```
void main(void)
{
}
```

# GLSL 데이터 타입



- Scalar types: float, int, bool
- Vector types: vec2, vec3, vec4
  ivec2, ivec3, ivec4,
  bvec2, bvec3, bvec4
- Matrix types: mat2, mat3, mat4
- Texture sampling: sampler1D, sampler2D, sampler3D,
  samplerCube
- C++ style 생성자들: vec3 a = vec3(1.0, 2.0, 3.0);

# GLSL 연산자 (Operators)



- C/C++ 스타일 수학, 논리 연산자들 사용
- matrix, vector 연산자들도 오버로딩(overloading) 되어있음

```
mat4 m;
vec4 a, b, c;

b = a*m;
c = m*a;
```

# Vector 타입의 각 요소 접근



- Vector 타입 변수의 각 요소 (component)는 [], xyzw, rgba, stpq 로 접근 가능

```
vec3 v;
```

`v[1], v.y, v.g, v.t` → 같은 요소 접근

- 요소들을 뒤섞는 것도 가능:

```
vec3 a, b;
a.xy = b.yx;
```

# GLSL Syntax: Vectors

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

- Vector 생성자

```
vec3 xyz = vec3(1.0, 2.0, 3.0);

vec3 xyz = vec3(1.0); // [1.0, 1.0, 1.0]

vec3 xyz = vec3(vec2(1.0, 2.0), 3.0);


vec3 xyz (1.0, 2.0, 3.0);   // ← Error!
```

# GLSL Syntax: Vectors

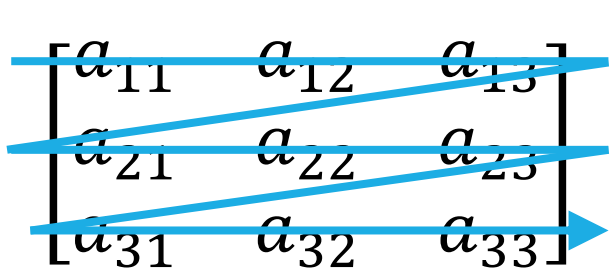
한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

```
vec4 c = vec4(0.5, 1.0, 0.8, 1.0);

vec3 rgb = c.rgb;  // [0.5, 1.0, 0.8]

vec3 bgr = c.bgr;  // [0.8, 1.0, 0.5]

vec3 rrr = c.rrr;  // [0.5, 0.5, 0.5]

c.a = 0.5;         // [0.5, 1.0, 0.8, 0.5]
c.rb = 0.0;        // [0.0, 1.0, 0.0, 0.5]

float g = rgb[1];  // 0.5, indexing, not swizzling
```

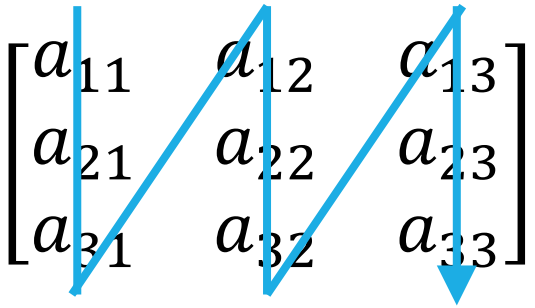
# GLSL Syntax: Matrices

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

- Matrices는 built-in type임:
  - Square: `mat2`, `mat3`, `mat4`
  - Rectangular: `mat`*mxn* ( `m` columns, `n` rows)
    `mat2x3`, `mat3x4`, `mat4x4`
- Stored **column major**




Row major order




Column major order

# GLSL Syntax: Matrices

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

- 생성자들

```
mat3 i = mat3(1.0); // 3x3 identity matrix

mat2 m = mat2(1.0, 2.0,  // [1.0 3.0]
              3.0, 4.0); // [2.0 4.0]
```

Column major!

- Matrix 요소 접근하기

```
float f = m[column][row];

float x = m[0].x; // x component of first column

vec2 yz = m[1].yz; // yz components of second column
```

Treat matrix as array of column vectors

# GLSL Syntax: Vectors and Matrices

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

- Matrix와 vector 연산은 쉽고 빠름:

```
vec3 xyz = // ...

vec3 v0 = 2.0 * xyz; // scale
vec3 v1 = v0 + xyz;  // component-wise
vec3 v2 = v0 * xyz;  // component-wise

mat3 m = // ...
mat3 t = // ...

mat3 mt = t * m;       // matrix * matrix
vec3 xyz2 = mt * xyz; // matrix * vector
vec3 xyz3 = xyz * mt; // vector * matrix  ( = transposed_matrix * vector )
```

- For more information:
  - http://en.wikibooks.org/wiki/GLSL_Programming/Vector_and_Matrix_Operations
  - http://en.wikibooks.org/wiki/GLSL_Programming/Applying_Matrix_Transformations

# GLSL 약속어 (Qualifiers)



- in, out: vertex attributes와 다른 변수들을 사용자 프로그램에서 쉐이더로 입/출력함

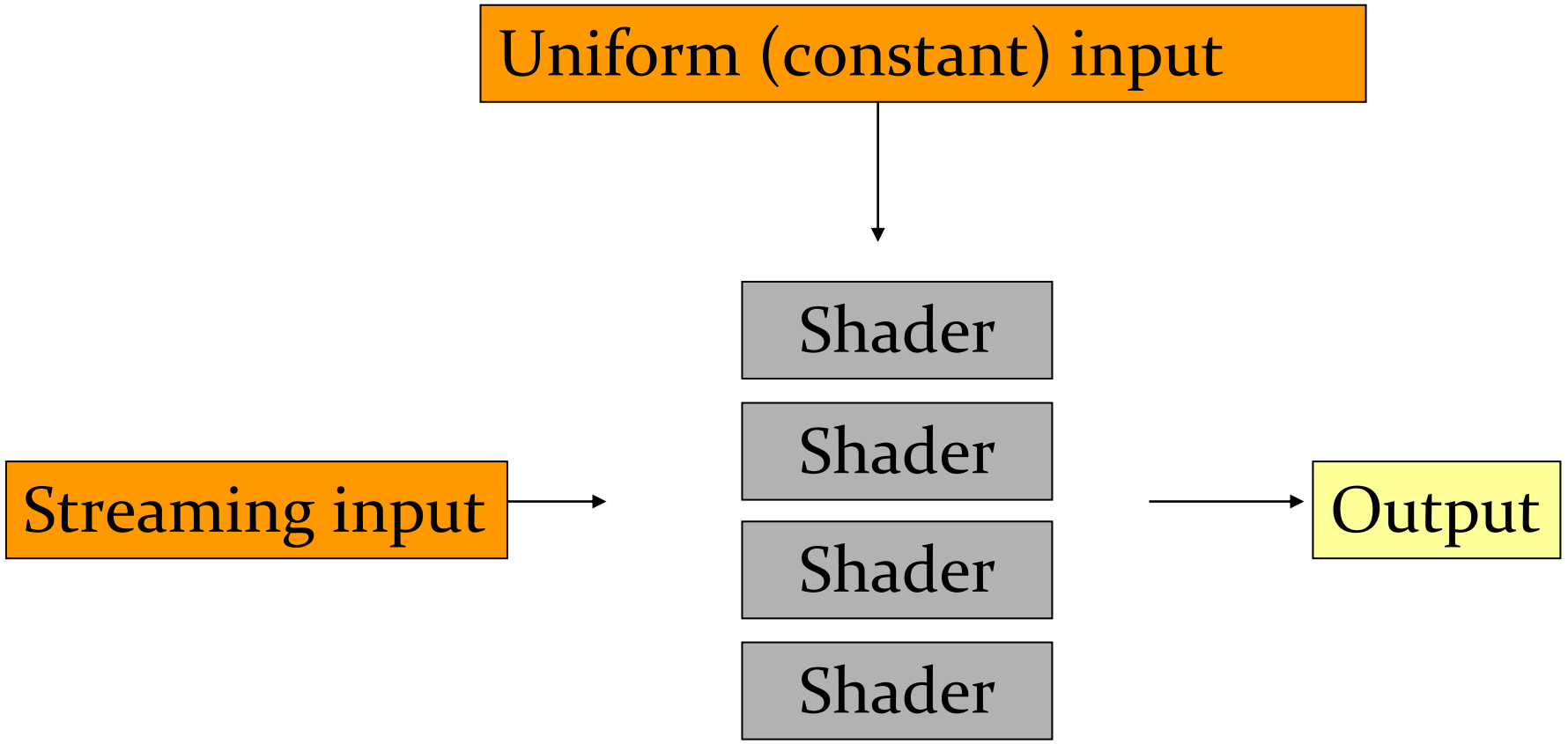
```
in vec2 tex_coord;
out vec4 color;
```

- Uniform: 프로그램으로부터 받는 변수. vertex/fragment에 따라서 변하지 않음

```
uniform float time;
uniform vec4 rotation;
```

# GLSL Syntax: in/out/uniform





# GLSL Syntax: in/out/uniform



```
#version 430

uniform mat4 u_ModelView;
in vec3 Position;
in vec3 Color;
out vec3 fs_Color;

void main(void)
{
  fs_Color = Color;
  gl_Position = u_ModelView * vec4(Position, 1.0);
}
```

`uniform`: shader input constant across `glDraw`

`in`: shader input varies per vertex attribute

`out`: shader output

# Flow Control



- if
- if else
- expression ? true-expression : false-expression
- while, do while
- for

# Functions



- Built-in functions
  - 수학 함수들: sqrt, power, abs
  - 삼각 함수들: sin, asin
  - 기하 함수들: length, reflect
- User-defined functions

# Built-in 변수들



- gl_Position: output position from vertex shader
- gl_FragColor: output color from fragment shader
  - Only for ES, WebGL, and older versions of GLSL
  - Present version use an out variable

# Vertex Shader 예시



```
#version 330

in vec4 vPosition;
in vec4 vColor;
out vec4 color;

void main() {
    color = vColor;
    gl_Position = vPosition;
}
```

# Fragment Shader 예시



```
#version 330

in vec4 color;
out vec4 FragColor;

void main() {
    FragColor = color;
}
```

# 사용자 프로그램에서 입력 받기

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

Shader에 입력으로 넘길 shader 변수 이름을 알고 shader에서의 주소를 찾음:

```
GLint  idx =
 glGetAttribLocation(program, "name");

GLint  idx =
 glGetUniformLocation(program, "name");
```

# 사용자 프로그램에서 입력 받기



```
// set up vertex arrays (after shaders are loaded)
GLuint vPosition = glGetAttribLocation(program, "vPosition”);
glEnableVertexAttribArray(vPosition);
glVertexAttribPointer(vPosition, 4, GL_FLOAT, GL_FALSE, 0,
 BUFFER_OFFSET(0));



GLuint vColor = glGetAttribLocation(program, "vColor”);
glEnableVertexAttribArray(vColor);
glVertexAttribPointer(vColor, 4, GL_FLOAT, GL_FALSE, 0,
 BUFFER_OFFSET(sizeof(points)));
```

# Uniform 변수 초기화하기

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

Uniform 변수들:

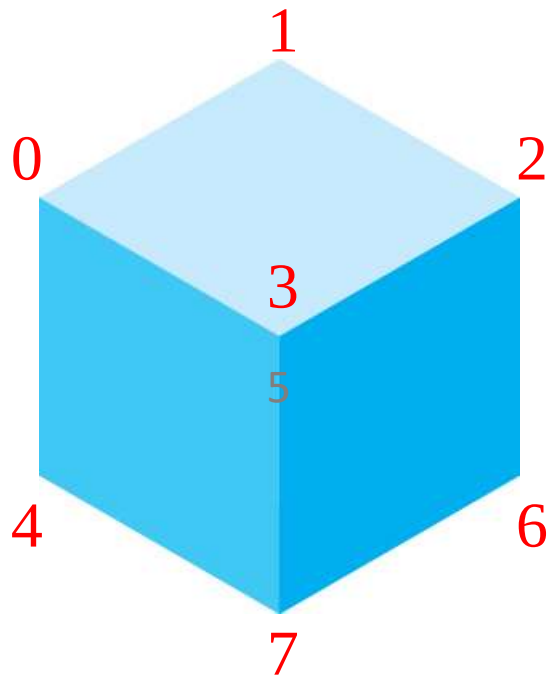
```
glUniform1f(index, value);

glUniform4f(index, x, y, z, w);

Glboolean transpose = GL_TRUE;
    // Since we're C programmers
Glfloat mat[3][4][4] = { … };

glUniformMatrix4fv(index, 3, transpose, mat);
```

# Program: Rotating ColorCube

# Cube Data



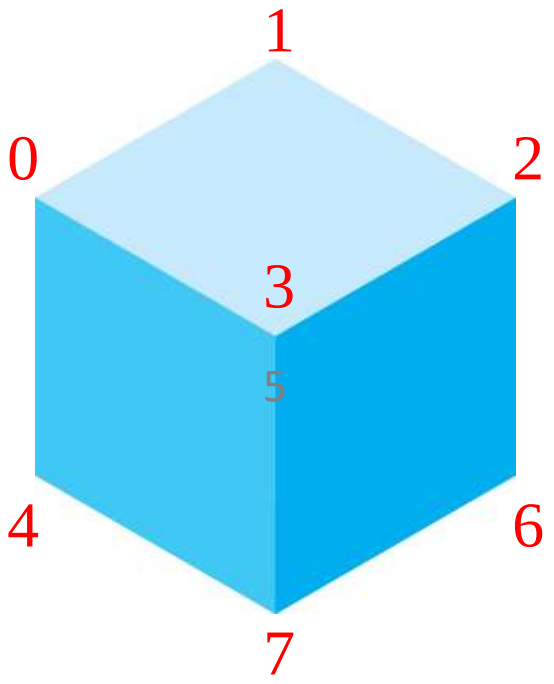
```
// Vertices of a unit cube centered at origin, sides aligned with axes

vec4 vertex_pos [8] = {
        vec4 ( -0.5, -0.5,  0.5, 1.0 ),
        vec4 ( -0.5,  0.5,  0.5, 1.0 ),
        vec4 (  0.5,  0.5,  0.5, 1.0 ),
        vec4 (  0.5, -0.5,  0.5, 1.0 ),
        vec4 ( -0.5, -0.5, -0.5, 1.0 ),
        vec4 ( -0.5,  0.5, -0.5, 1.0 ),
        vec4 (  0.5,  0.5, -0.5, 1.0 ),
        vec4 (  0.5, -0.5, -0.5, 1.0 )
};
```



# Cube Data



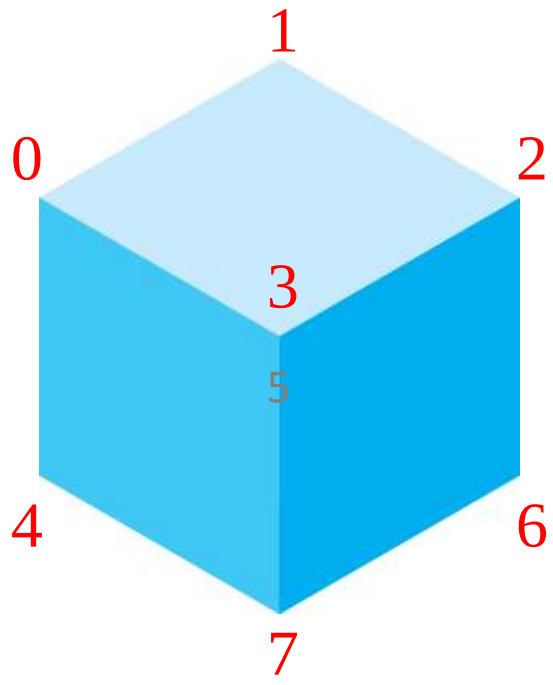
```
// RGBA colors
vec4 vertex_colors[8] = {
        vec4 ( 0.0, 0.0, 0.0, 1.0 ),  // black
        vec4 ( 1.0, 0.0, 0.0, 1.0 ),  // red
        vec4 ( 1.0, 1.0, 0.0, 1.0 ),  // yellow
        vec4 ( 0.0, 1.0, 0.0, 1.0 ),  // green
        vec4 ( 0.0, 0.0, 1.0, 1.0 ),  // blue
        vec4 ( 1.0, 0.0, 1.0, 1.0 ),  // magenta
        vec4 ( 1.0, 1.0, 1.0, 1.0 ),  // white
        vec4 ( 0.0, 1.0, 1.0, 1.0 )   // cyan
};
```



# Generating a Cube Face from Vertices

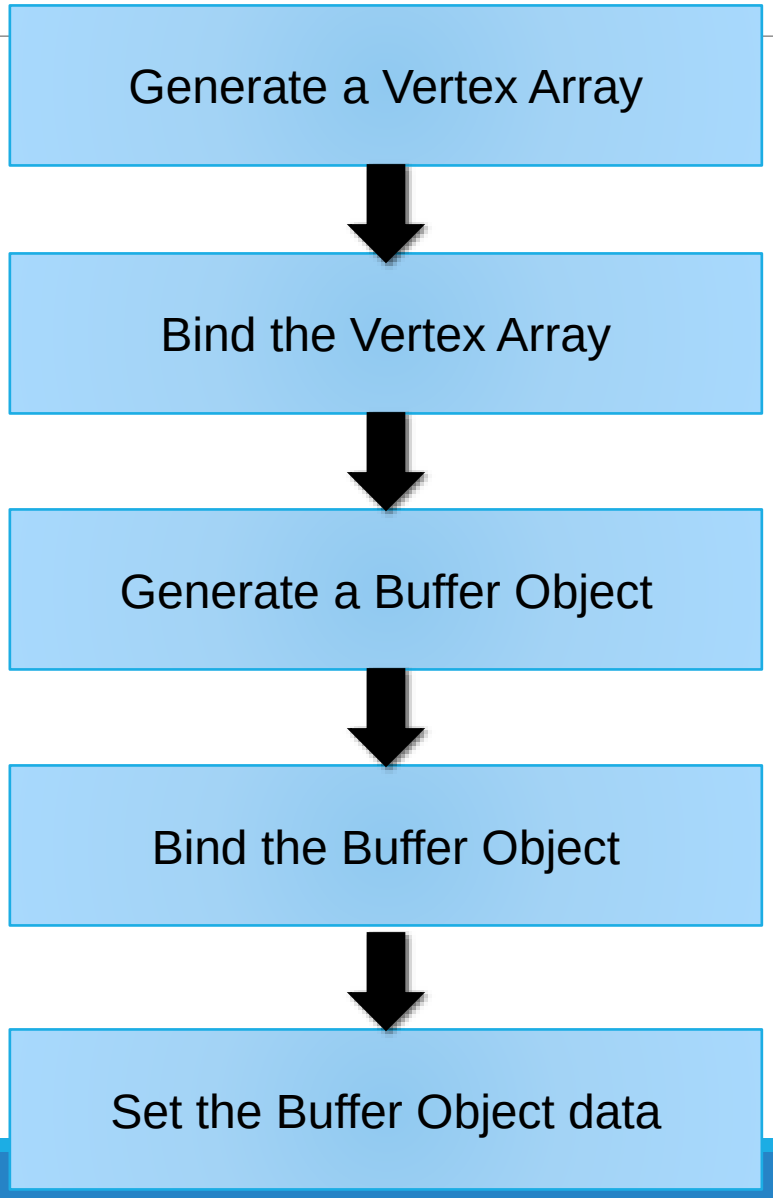


```
// generate 12 triangles: 36 vertices and 36 colors
void
colorcube() {
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```



# How to Send Data

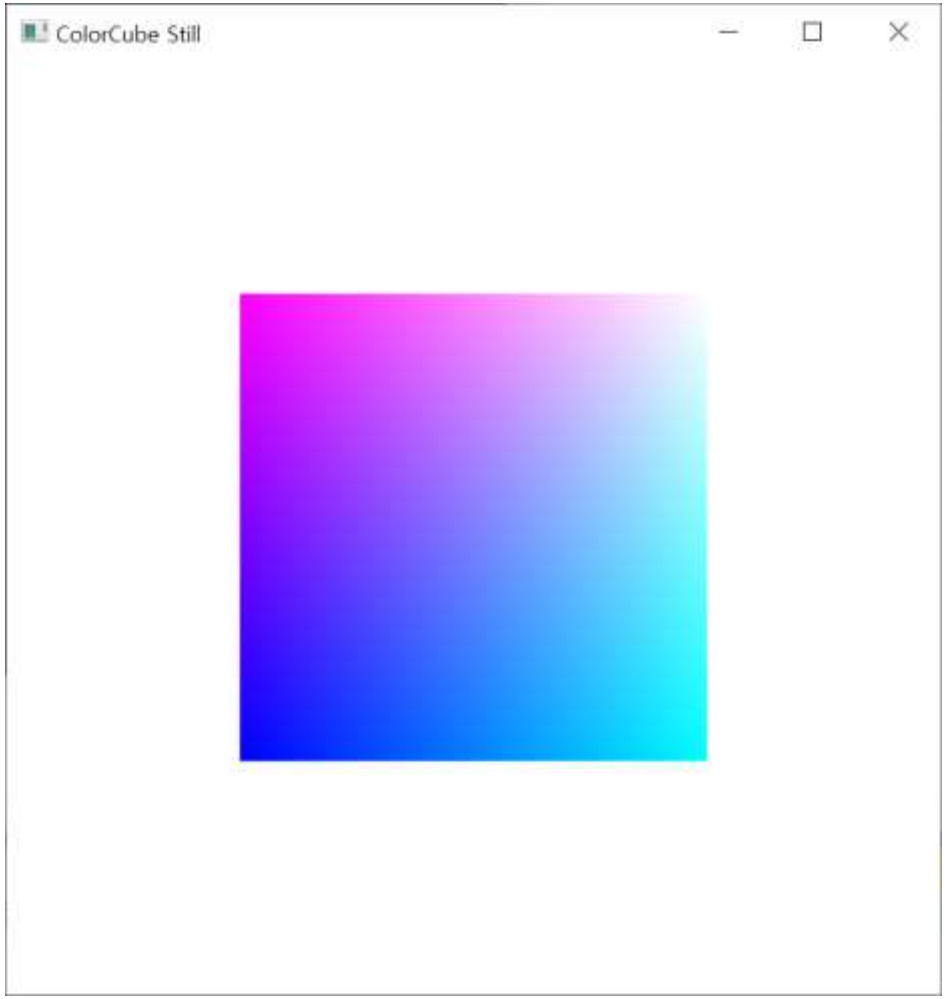





```
glGenVertexArray(..)
glBindVertexArray(..)
glGenBuffers(..)
glBindBuffer(..)
glBufferData(..)
```

# 실행 결과



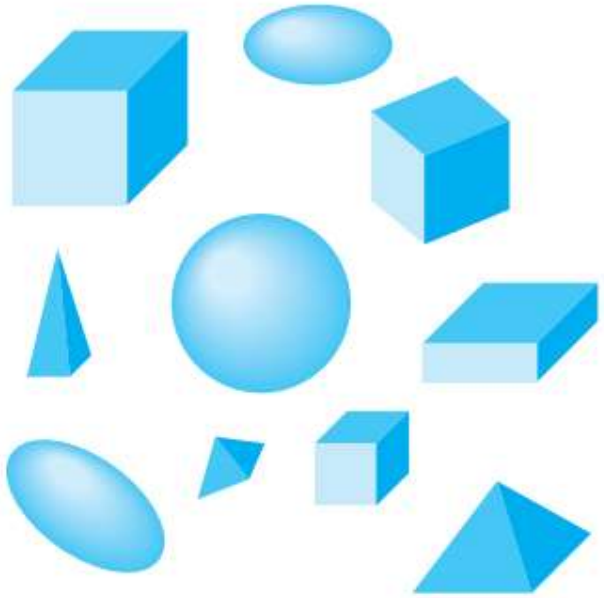


# Instance Transformation

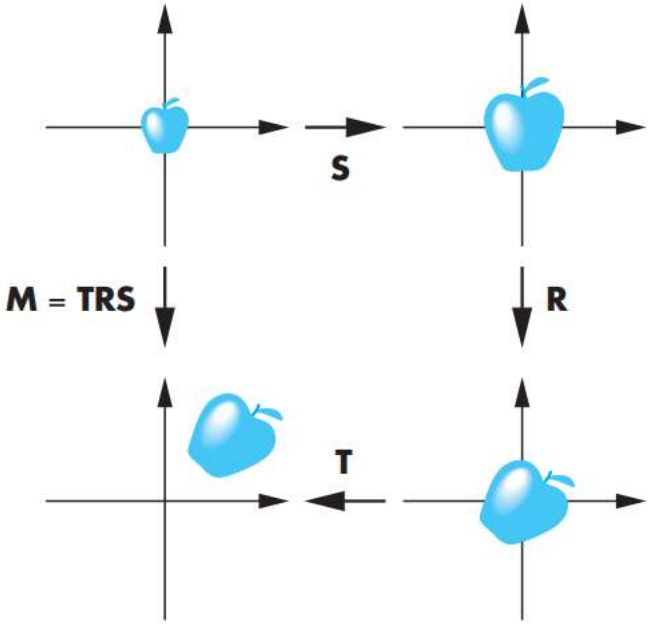


모델링에서 일반적으로 transformation은 다음의 순서로 적용됨

$$\mathbf{M} = \mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S}$$



Scene of geometric objects

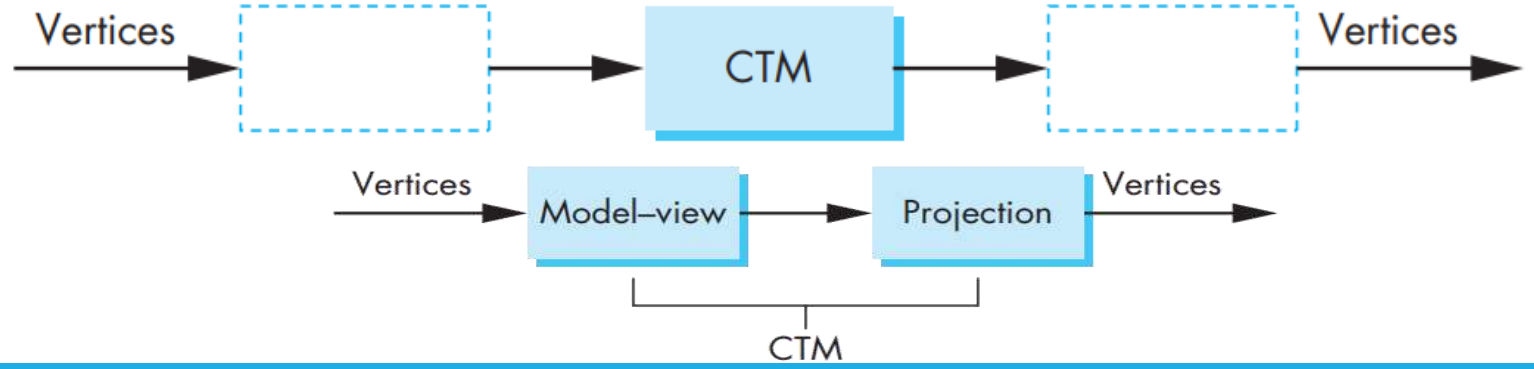



Instance Transformation

# OpenGL Transformation Matrices



- Transformation matrix은 OpenGL에서 상태 변수(state variable)
- Transformation matrix in OpenGL
  - model-view matrix: model frame (geometric object representation)에서 eye frame으로
  - projection matrix: 투영변환 + clip coordinate
- Current Transformation Matrix (CTM)
  - Pipeline은 CTM을 이용하여 어플리케이션에서 입력받은 점을 변환



# Current Transform Matrix



- CTM은 수행하고자 하는 순서의 반대로 업데이트 한다
  - ‘Stack’ like operation

(예시) (4,5,6)를 지나고 방향이 (1,2,3)인 회전축을 중심으로 45도만큼 회전하기 위한 CTM

$$\mathbf{C} \leftarrow \mathbf{I},$$

$$\mathbf{C} \leftarrow \mathbf{CT}(4.0, 5.0, 6.0),$$

$$\mathbf{C} \leftarrow \mathbf{CR}(45.0, 1.0, 2.0, 3.0),$$

$$\mathbf{C} \leftarrow \mathbf{CT}(-4.0, -5.0, -6.0).$$

# Spinning the Cube



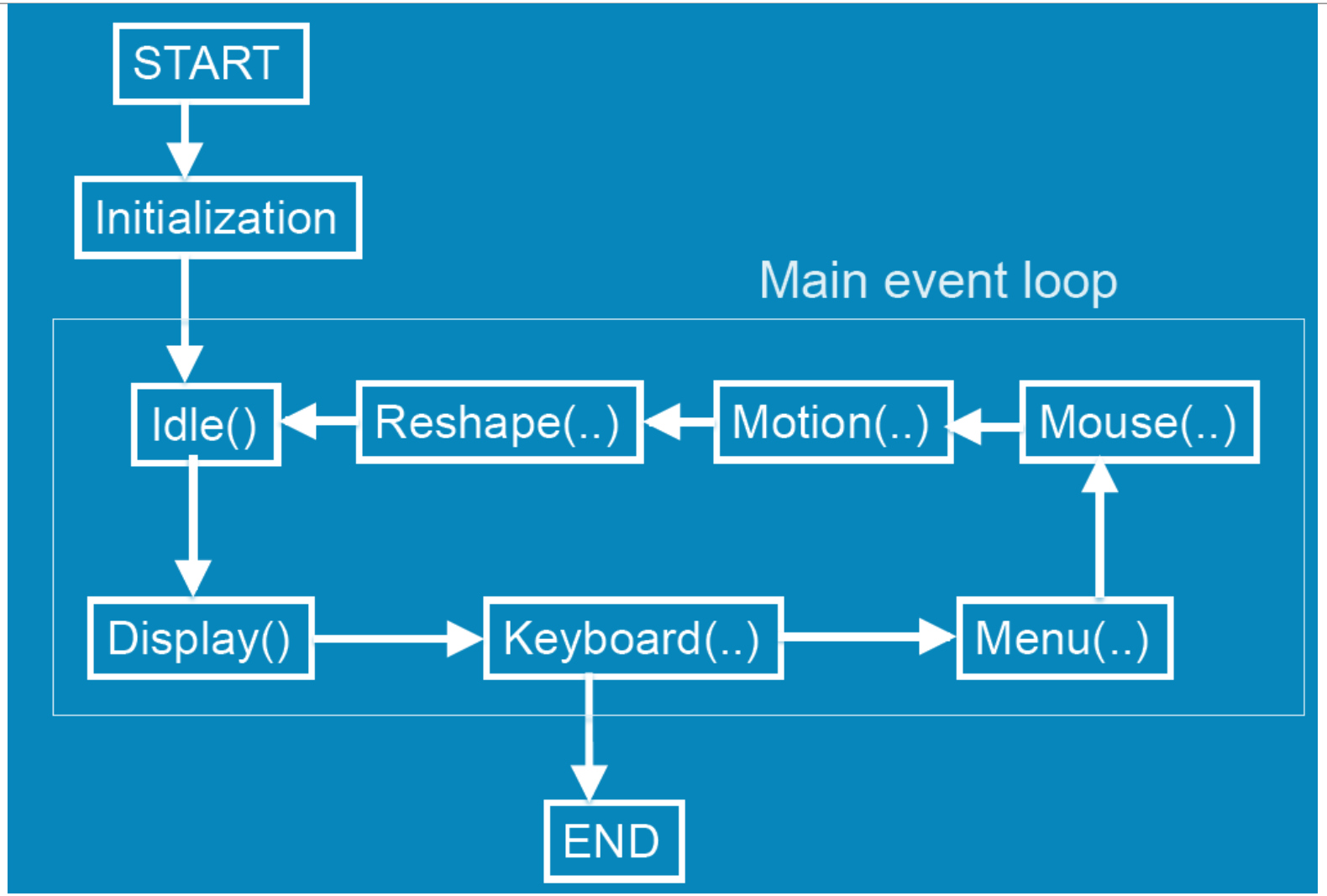
Cube를 회전시키기 위해서는 필요한 callback들

```
glutDisplayFunc(display);
glutMouseFunc(mouse);
glutIdleFunc(idle);
```

# Interaction: Callback Functions

# GLUT Program with Callbacks





# 이벤트 종류들



- Window: resize, expose, iconify
- Mouse: 하나 (또는 하나 이상)의 버튼을 클릭함
- Motion: 마우스 이동
- Keyboard: key를 누르거나 뗌
- Idle: non-event

  (다른 이벤트들이 발생하지 않았을 때 불리는 함수)

# Callbacks



- 이벤트가 발생했을 때 호출되는 함수
- 유저는 앞서 정의된 특정 이벤트가 발생했을 때 호출되는 함수를 형식에 맞게 정의함
- 유저가 정의한 함수를 callback 함수로 등록함 (in main())

예시)

```
glutMouseFunc(mymouse)
```

mouse callback function

# GLUT callbacks



모든 윈도우 기반 OS 시스템에서 공통적으로 사용할 수 있는 GLUT callback들

- **`glutDisplayFunc`**
- **`glutMouseFunc`**
- **`glutReshapeFunc`**
- **`glutKeyboardFunc`**
- **`glutIdleFunc`**
- **`glutMotionFunc, glutPassiveMotionFunc`**

# Callback의 종류들



- Display(): 윈도우가 그려져야만 할 때 호출
- Idle(): 다른 이벤트가 발생하지 않으면 호출
- Keyboard(unsigned char key, int x, in y): key가 눌렸을 때 호출, 어떤 key를 눌렀는지 key의 ASCII 코드가 전달됨
- Menu(…): menu에서 선택한 후 호출
- Mouse(int button, int state, int x, in y): mouse 버튼 callback. 어떤 버튼인지(왼쪽/중간/오른쪽), 어떤 상태인지(press/release), 이벤트가 발생한 (화면) 위치와 함께 전달됨
- Reshape(int w, int h): 윈도우의 크기가 바뀔 때 호출
- 따로 지정해 주지 않으면 callback은 NULL임

# Event가 발생한 후 처리



- 많은 이벤트는 display callback 함수를 호출함
  - 한 event loop 안에서 불필요하게 많은 display 함수가 수행됨
- **`glutMainLoop()`:** flag을 설정하여 display 함수의 호출을 조절할 수 있음
- GLUT는 event loop의 끝에서 이 flag를 check하고 이 flag가 설정되어 있으면 display 함수를 호출함

# Rotating Cube Again



How to display the rotated cube?

(방법 1) 어플리케이션에서 큐브를 회전시키고 업데이트된 데이터를 다시 buffer로 보냄

```
Mat4

ctm = RotateX(theta[0])*RotateY(theta[1])*RotateZ(theta[2]);

…

New_points[i] = ctm *points[i];

…

glBufferSubData(GL_ARRAY_BUFER, 0, sizeof(new_points), new_points);

glDrawArray(GL_TRIANGLES, 0, NumVertices)
```

# Rotating Cube Again



How to display the rotated cube?

(방법 2) 어플리케이션에서 CTM을 계산, 매트릭스를 shader로 넘김

```
Mat4

ctm = RotateX(theta[0])*RotateY(theta[1])*RotateZ(theta[2]);

…

Gluint matrix_loc = getUniformLocation(program, “rotation”);

glUniformMatrix4fv(matrix_loc, 1, GL_TRUE, ctm);

…
```

# Rotating Cube Again



How to display the rotated cube?

(방법 2) 어플리케이션에서 CTM을 계산, 매트릭스를 shader로 넘김

**(vshader.glsl)**

```
#version 330

in vec4 vPosition;
in vec4 vColor;
out vec4 color;
Uniform mat4 rotation

void main() {
    color = vColor;
    gl_Position = rotation * vPosition;
}
```

# Rotating Cube Again



How to display the rotated cube?

(방법 3) 어플리케이션에서 theta만 shader로 넘기고, shader에서 CTM을 계산

```
Gluint theta_loc = getUniformLocation(program, “theta”);

glUniform3fv(theta_loc, 1, Theta);
```

# Rotating Cube Again



How to display the rotated cube?

(방법 3) 어플리케이션에서 theta만 shader로 넘기고, shader에서 CTM을 계산

**(vshader.glsl)**

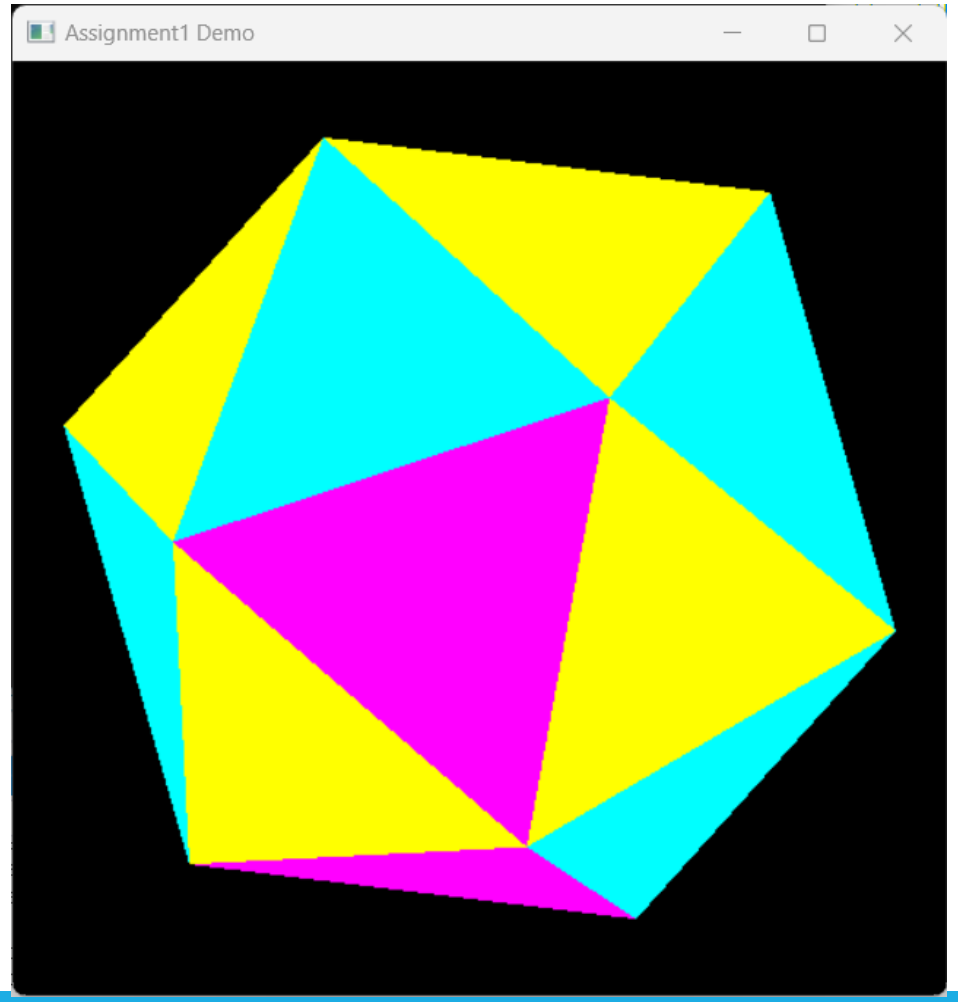
```
#version 330

in vec4 vPosition;
in vec4 vColor;
out vec4 color;
Uniform vec3 theta;

void main() {
    // Compute the sines and cosines of theta for each of
    //   the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin( angles );

    // Remeber: thse matrices are column-major
    mat4 rx = mat4( 1.0,  0.0,  0.0, 0.0,
                    0.0,  c.x,  s.x, 0.0,
                    0.0, -s.x,  c.x, 0.0,
                    0.0,  0.0,  0.0, 1.0 );

    mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                    0.0, 1.0,  0.0, 0.0,
                    s.y, 0.0,  c.y, 0.0,
                    0.0, 0.0,  0.0, 1.0 );

    // Workaround for bug in ATI driver
    ry[1][0] = 0.0;
    ry[1][1] = 1.0;

    mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
                    s.z,  c.z, 0.0, 0.0,
                    0.0,  0.0, 1.0, 0.0,
                    0.0,  0.0, 0.0, 1.0 );

    // Workaround for bug in ATI driver
    rz[2][2] = 1.0;

    color = vColor;
    gl_Position = rz * ry * rx * vPosition;
}
```

# Programming Assignment #1

# Assignment #1 정20면체 그리기

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

- 정 20면체(Regular Icosahedron) 그리기:
  - 정 20면체를 화면에 그리기
  - Edge를 공유하는 면은 다른 색으로 칠하기
  - Space bar가 누르면 회전을 시작,
  다시 누르면 회전 중지 (마우스로 회전축 선택)
  - aswd를 누르면 좌우 또는 상하로 평행이동
  - z,x를 누르면 20면체 축소, 또는 확대
  - Q (q)를 누르면 프로그램 종료



# Assignment #1

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

제출물: main.cpp + .h 파일들, vshader.glsl, fshader.glsl을 zip으로 압축, 제출

기한: 10월 5일 23시 59분까지