418382
Lecture 12

Pramook Khungurn

ชนิดตัวแปรในภาษา GLSL

- เหมือนภาษา C
  - float, int

- เพิ่มจากภาษา C
  - vec2, vec3, vec4 ใช้แทนเวกเตอร์
  - mat2, mat3, mat4 ใช้แทนเมตริกซ์
  - sampler1D, sampler2D, sampler3D, samplerCube, ฯลฯ ใช้แทน texture

- ไม่มีเหมือนภาษา C
  - string
  - pointer

vec2

- เวกเตอร์สองมิติ มี 2 component แต่ละ component เป็น float

```
vec2 a;
a.x = 0.0;
a.y = 1.0; // a = (0,1)

vec2 b;
b.s = 10.0;
b.t = 12.5; // b = (10,12.5)

vec2 c;
c[0] = 9.0;
c[1] = 8.0; // c= (9,8)

float p = a.t;        // p = 1
float q = b[1] + c.x // q = 21.5

vec2 d = vec2(3,c.y * 2); // d = (3,18)
```

vec2

```
vec2 d = vec2(3,c.y * 2); // d = (3,18)

vec3 d = a + b; // d = (10,13.5)

vec3 e = b - c; // e = (1,4.5)

vec3 f = b * c; // f = (90,100)

vec3 g = 3 * a; // g = (0,3)

float h = length(c); // h = 12.042
```

vec3

- เวกเตอร์สามมิติ แต่ละ component เป็น float

```
vec3 a;
a.x = 10.0; a.y = 20.0; a.z = 30.0;  // a = (10,  20,  30)
a.r = 0.1;  a.g = 0.2;  a.b = 0.3;   // a = (0.1, 0.2, 0.3)
a.s = 1.0,  a.t = 2.0;  a.p = 3.0;   // a = (1, 2, 3)
```

```
vec3 b = vec3(4.0, 5.0, 6.0);

vec3 c = a + b; // c = (5, 7, 9)
vec3 d = a - b; // d = (-3, -3, -3)
vec3 e = a * b; // e = (4, 10, 18)
vec3 f = a * 3; // e = (3, 6, 9)
float g = dot(a,b); // g = 32
vec3 h = cross(a,b); // h = (-5,6,-3)
float i = length(a); // i = 3.742`
```

vec4

- เวกเตอร์ 4 มิติ แต่ละ component เป็น float

```
vec4 a;
a.x = 10.0; a.y = 20.0; a.z = 30.0; a.w = 40.0; // a = (10, 20, 30, 40)
a.r = 0.1;  a.g = 0.2;  a.b = 0.3;  a.a = 0.4;  // a = (0.1, 0.2, 0.3, 0.4)
a.s = 1.0;  a.t = 2.0;  a.p = 3.0;  a.q = 4.0;  // a = (1,2,3,4)
```

```
vec4 b = vec4(5,6,7,8);

vec4 c = a + b; // c = (6, 8, 10, 12)
vec4 d = a - b; // d = (-4, -4, -4, -4)
vec4 e = a * b; // e = (5, 12, 21, 32)
vec4 f = a * 3; // f = (3, 6, 9, 12)
float g = length(a); // g = 5.477
```

mat2

- เมตริกซ์ขนาด 2x2

```
mat2 A = mat2(1.0, 2.0, 3.0, 4.0); // in column-major order

vec2 x = vec2(1.0, 0.0);
vec2 y = vec2(0.0, 1.0);
```

```
vec2 a = A * x; // a = (1,2)
vec2 b = A * y; // b = (3,4)
```

mat3

- เมตริกซ์ขนาด 3x3

```
mat3 A = mat3(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0);
         // in column-major order

vec3 x = vec3(1.0, 0.0, 0.0);
vec3 y = vec3(0.0, 1.0, 0.0);
vec3 z = vec3(0.0, 0.0, 1.0);

vec3 a = A * x; // a = (1,2,3)
vec3 b = A * y; // b = (4,5,6)
vec3 c = A * z; // c = (6,7,8)
```

mat4

- เมตริกซ์ขนาด 4x4 (ตรงขนาดกับที่ OpenGL ใช้เก็บ transformation ต่างๆ)

```
mat4 A = mat2(1.0,  2.0,  3.0,  4.0,
              5.0,  6.0,  7.0,  8.0,
              9.0,  10.0, 11.0, 12.0
              13.0, 14.0, 15.0, 16.0); // in column-major order
```

```
vec4 x = vec4(1.0, 0.0, 0.0, 0.0);
vec4 y = vec4(0.0, 1.0, 0.0, 0.0);
vec4 z = vec4(0.0, 0.0, 1.0, 0.0);
vec4 w = vec4(0.0, 0.0, 0.0, 1.0);

vec4 a = A * x; // a = (1,2,3,4)
vec4 b = A * y; // b = (5,6,7,8)
vec4 c = A * z; // c = (9,10,11,12)
vec4 d = A * w; // d = (13,14,15,16)
```

Array

- เราสามารถประกาศอะเรย์ที่มีจำนวนสมาชิกคงตัวได้

```
float A[4];
A[0] = 5; A[3] = 10;

vec4 B[10];
B[3] = vec4(1,2,3,4); B[8].y = 10.0;
```

Twizzle

- ใช้สร้างเวกเตอร์จากเวกเตอร์ด้วยการอ้างถึงสมาชิกพร้อมกันทีละหลายๆ ตัว

```
vec4 a = vec4(1,2,3,4);

vec3 b = a.xyz;  // b = (1,2,3)
vec2 c = a.qp;   // c = (4,3)
vec4 d = a.xxyy; // d = (1,1,2,2)
```

การแปลงข้อมูลระหว่างชนิดข้อมูลต่างๆ

- ใช้คำสั่ง: ตัวแปร = ชนิดข้อมูล(ค่า);
- นิพจน์ทางด้านซ้ายเรียกว่า “constructor expression”
- ตัวอย่าง:

```
float a = 1.0;
int b = int(a);
```

การแปลงข้อมูลระหว่างชนิดข้อมูลต่างๆ

- สามารถสร้างเวกเตอร์ขนาดใหญ่กว่าจากเวกเตอร์ขนาดเล็กกว่าได้

```
vec2 a = vec2(1,2);
vec2 b = vec2(3,4);

vec4 c = vec4(a,b);     // c = (1,2,3,4)
```

```
vec3 d = vec3(0,0,1);

vec4 e = vec4(d,0);    // d = (0,0,1,0)

vec4 f = vec2(0,a,3); // f = (0,1,2,3)
```

Uniform Variables

Uniform Variable

- ตัวแปรที่ผู้ใช้สามารถกำหนดค่าจากโปรแกรมภาษา C แบบหนึ่ง
- ค่าเปลี่ยนแปลงไม่บ่อย
- เหมาะกับการใช้กำหนด
  - สัมประสิทธ์ต่างๆ ของ Phong lighting model
  - เมตริกซ์ที่ใช้แปลง vertex ต่างๆ
  - ความเข้มแสง ตำแหน่งของแหล่งกำเนิดแสง
  - texture
- ห้ามกำหนดค่าระหว่าง glBegin(...) กับ glEnd( )
  - คล้ายกับสี material

วิธีประกาศ Uniform Variable

- ประกาศไว้เป็น global variable (อยู่นอกฟังก์ชันต่างๆ ทั้งหมด)
- เติมคำว่า “uniform” ข้างหน้าชนิดตัวแปร
- ตัวอย่าง

```
uniform float shininess;
uniform vec3 color;
uniform mat4 model_transform;
uniform sampler2D texture;

void main()
{
    // Code here...
}
```

ข้อควรระวัง

- ถ้าในโปรแกรมหนึ่งมีการประกาศ uniform ชื่อเดียวกัน ไว้ในทั้ง vertex shader และ fragment shader แล้ว ตัวแปรทั้งสองตัวนั้นคือตัวเดียวกัน

- ดังนั้นใน vertex shader และ fragment shader จะมี uniform variable ที่มีชื่อเดียวกัน แต่ชนิดข้อมูลคนละชนิดกันไม่ได้

การเตรียมการและตั้งค่า Uniform Variable จาก C

- มี 2 ขั้นตอน

    1. หาเลข location ของตัวแปร uniform หลังจาก link โปรแกรมเสร็จแล้ว

    2. กำหนดค่าให้ตัวแปร uniform ด้วยคำสั่ง glUniform หลังจากใช้โปรแกรมแล้ว

1. หาเลข Location ของตัวแปร uniform

- ใช้คำสั่ง
  GLint **glGetUniformLocation**(GLuint *program*, const GLchar **name*);

  - program คือหมายเลขโปรแกรม
  - name คือชื่อของตัวแปร uniform

1. หาเลข Location ของตัวแปร uniform

- สมมติว่าในโค้ดของ shader เป็นดังนี้

```
uniform vec3 color;

void main()
{
    // Some code here...
}
```

- สมมติว่าเลขของโปรแกรมเก็บไว้ในตัวแปรชื่อ prog_id

- เราสามารถหาเลข location ของตัวแปร color ได้ดังนี้

```
GLint color_location = glGetUniformLocation(prog_id, "color");
```

2. กำหนดค่าให้ตัวแปร Uniform

- ใช้คำสั่ง glUniform[1234][fi](GLint location, ...)
  - location คือหมายเลข location ที่หาได้จาก glGetUniformLocation
  - ส่วนที่เหลือเป็นข้อมูล
- ลักษณะคล้ายกับฟังก์ชัน glVertex, glColor, ฯลฯ ที่เลือกจำนวน argument และชนิดของ argument ได้

2. กำหนดค่าให้ตัวแปร Uniform

- ใช้คำสั่งให้ถูกต้องตามชนิดตัวแปร
  - กำหนดค่า float ให้ใช้ glUniform1f
  - กำหนดค่า vec2 ให้ใช้ glUniform2f
  - กำหนดค่า vec3 ให้ใช้ glUniform3f
  - กำหนดค่า vec4 ให้ใช้ glUniform4f
  - กำหนดค่า int ให้ใช้ glUniform1i

2. กำหนดค่าให้ตัวแปร Uniform

- ตัวอย่าง: สมมติว่าเราจะกำหนดค่าให้ตัวแปร color จากสามสไลด์ที่แล้ว สามารถทำได้ดังนี้

```
glUniform3f(color_location, 1.0f, 2.0f, 3.0f);
```

2. กำหนดค่าให้ตัวแปร Uniform

- สำหรับการกำหนดค่าให้ตัวแปรที่เป็นเมตริกซ์ ให้ใช้คำสั่ง
  void **glUniformMatrix[234]fv**(GLint *location*, GLsizei *count*,
  GLboolean *transpose*, const GLfloat **value*);

  - location คือเลข location ของตัวแปร uniform

  - count คือจำนวนเมตริกซ์ที่จะกำหนดค่า
    ส่วนมากจะเป็น 1 เนื่องจากเรากำหนดให้เมตริกซ์แค่ตัวเดียว
    แต่ถ้ากำหนดค่าให้อะเรย์ของเมตริกซ์ค่าจะมากกว่าหนึ่งก็ได้

  - value คืออะเรย์ของสมาชิกของเมตริกซ์

  - ถ้า transpose เป็น GL_FALSE หมายความว่าค่าใน value อยู่ใน column-major order แต่ถ้าเป็น GL_TRUE หมายความว่าอยู่ใน row-major order

2. กำหนดค่าให้ตัวแปร Uniform

- ใช้คำสั่งให้ถูกต้องตามชนิดตัวแปร
    - กำหนดค่า mat2 ให้ใช้ glUniformMatrix2fv
    - กำหนดค่า mat3 ให้ใช้ glUniformMatrix3fv
    - กำหนดค่า mat4 ให้ใช้ glUniformMatrix4fv

2. กำหนดค่าให้ตัวแปร Uniform

- ตัวอย่าง

```
float *mat2_data = {1.0f, 2.0f, 3.0f, 4.0f};
glUniformMatrix2f(mat2_location, 1, GL_FALSE, mat2_data);

float *mat4_data = {1.0f, 0.0f, 0.0f, 3.0f,
                    0.0f, 1.0f, 0.0f, 2.0f,
                    0.0f, 0.0f, 1.0f, 5.0f,
                    0.0f, 0.0f, 0.0f, 1.0f};
glUniformMatrix4f(mat4_location, 1, GL_TRUE, mat4_data);
```

ตัวอย่าง: 2D Twisting

ตัวอย่าง: 2D Twisting

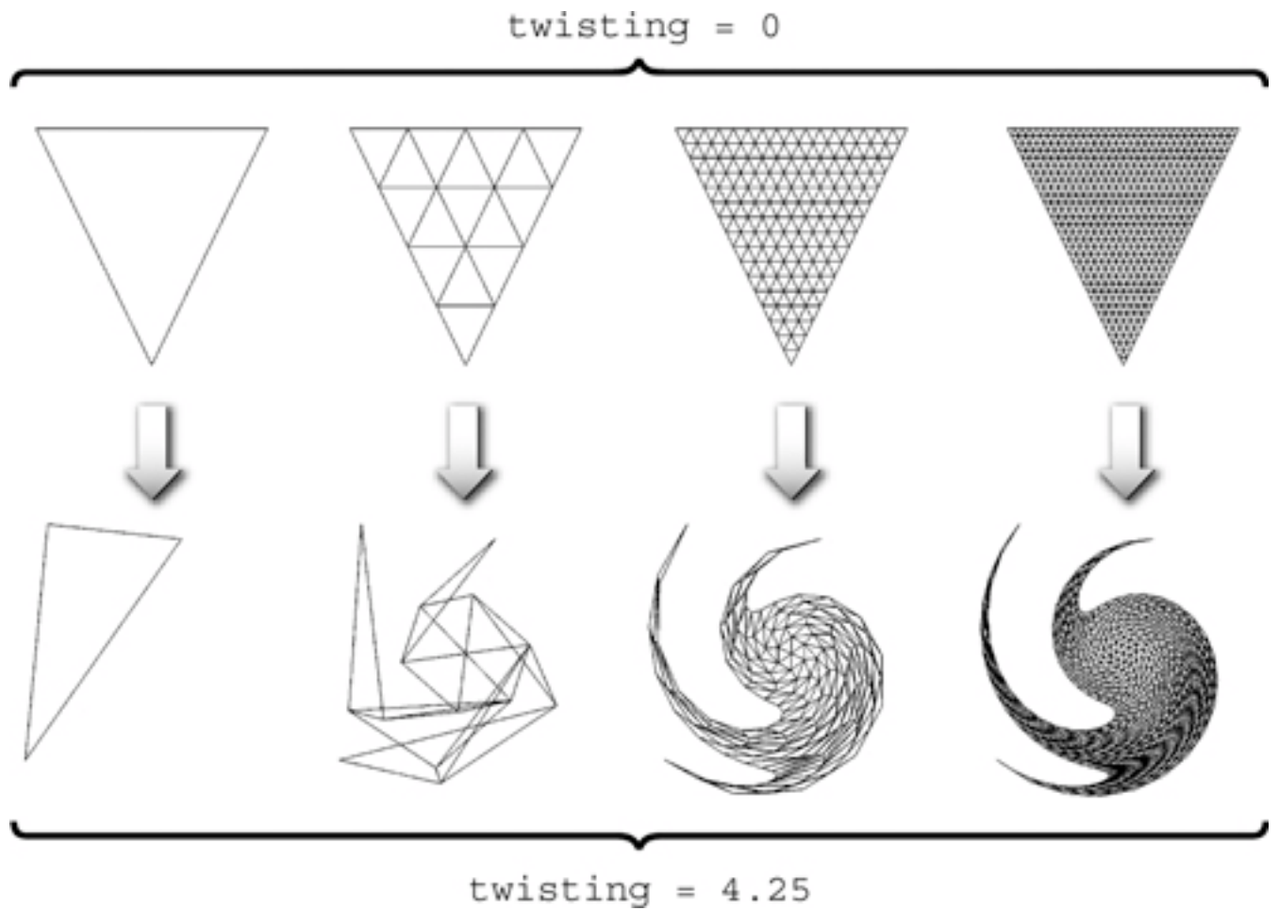
- หลักการ
  - ทำการแปลง vertex เอง ตามสูตรต่อไปนี้

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- (x,y) คือตำแหน่งสองมิติของ vertex ใน object space
- (x’,y’) คือตำแหน่งสองมิติของ vertex ใน clip space
- θ มีค่าเท่ากับ twisting (ค่าที่ผู้ใช้กำหนด) คูณกับความยาวของเวกเตอร์ (x,y)

ตัวอย่าง: 2D Twisting

- ผลของการแปลง



โค้ดของ vertex shader

```
#version 110

uniform float twisting;

void main()
{
    float angle = twisting * length(gl_Vertex.xy);
    float s = sin(angle);
    float c = cos(angle);
    gl_Position.x = c * gl_Vertex.x - s * gl_Vertex.y;
    gl_Position.y = s * gl_Vertex.x + c * gl_Vertex.y;
    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
}
```

โค้ดของ fragment shader

```
#version 110

uniform vec3 color;

void main()
{
    gl_FragColor = vec4(color, 1);
}
```

โค้ดการเตรียมโปรแกรม GLSL ในภาษา C

```
vert_id = glCreateShader(GL_VERTEX_SHADER);
frag_id = glCreateShader(GL_FRAGMENT_SHADER);

glShaderSource(vert_id, 1, &twisting_twisting_program_vert_code, NULL);
glShaderSource(frag_id, 1, &twisting_twisting_program_frag_code, NULL);

glCompileShader(vert_id);
glCompileShader(frag_id);

print_shader_info_log(vert_id);
print_shader_info_log(frag_id);

prog_id = glCreateProgram();
glAttachShader(prog_id, vert_id);
glAttachShader(prog_id, frag_id);

glLinkProgram(prog_id);
print_program_info_log(prog_id);

twisting_location = glGetUniformLocation(prog_id, "twisting");
color_location = glGetUniformLocation(prog_id, "color");
```

โค้ดตอนวาดรูป

```
glUseProgram(prog_id);
glUniform1f(twisting_location, 4.25f);
glUniform3f(color_location, 0.5f, 9.0f, 0.5f);

int count = 100;
float size = 1.0f / count;

glBegin(GL_QUADS);

FOR(i, count)
FOR(j, count)
{
    float x = -0.5f + i * size;
    float y = -0.5f + j * size;

    glVertex2f(x,      y);
    glVertex2f(x+size, y);
    glVertex2f(x+size, y+size);
    glVertex2f(x,      y+size);
}

glEnd();
```

การใช้ Texture ใน GLSL

sampler2D

- สำหรับเก็บ texture
- มักประกาศเป็นตัวแปร uniform
- อ่านค่าใน GLSL โดยใช้ฟังก์ชัน
  vec4 **texture2D**(sampler2D *texture*, vec2 *tex_coord*)

ตัวแปรสำหรับ Texture Coordinate

- ใน vertex program
  - ตัวแปรชื่อ gl_MultiTexCoord0 ถึง glMultiTexCoord7 (รวม 8 ตัว) จะเก็บค่า texture coordinate ไว้
  - ตัวแปรเหล่านี้แต่ละตัวมีชนิด vec4
  - เวลาจะเอาไปใช้อ่าน texture จะใช้แค่ component แรก เช่น gl_MultiTexCoord0.st
  - ค่าที่กำหนดให้ด้วย glTexCoord ในภาษา C จะมาอยู่ใน gl_MultiTexCoord0

ตัวแปรสำหรับ Texture Coordinate

- ถ้าอยากให้ fragment shader สามารถอ่าน texture coordinate ได้ด้วย ใน vertex program จะต้องกำหนดตัวแปรชื่อ gl_TexCoord

- ตัวแปรนี้เป็นอะเรย์ซึ่งมีสมาชิกอยู่ gl_MaxTextureCoords (ตัวแปรพิเศษอีกตัวหนึ่งที่เสาสามารถอ่านได้) แต่ละตัวเป็น vec4

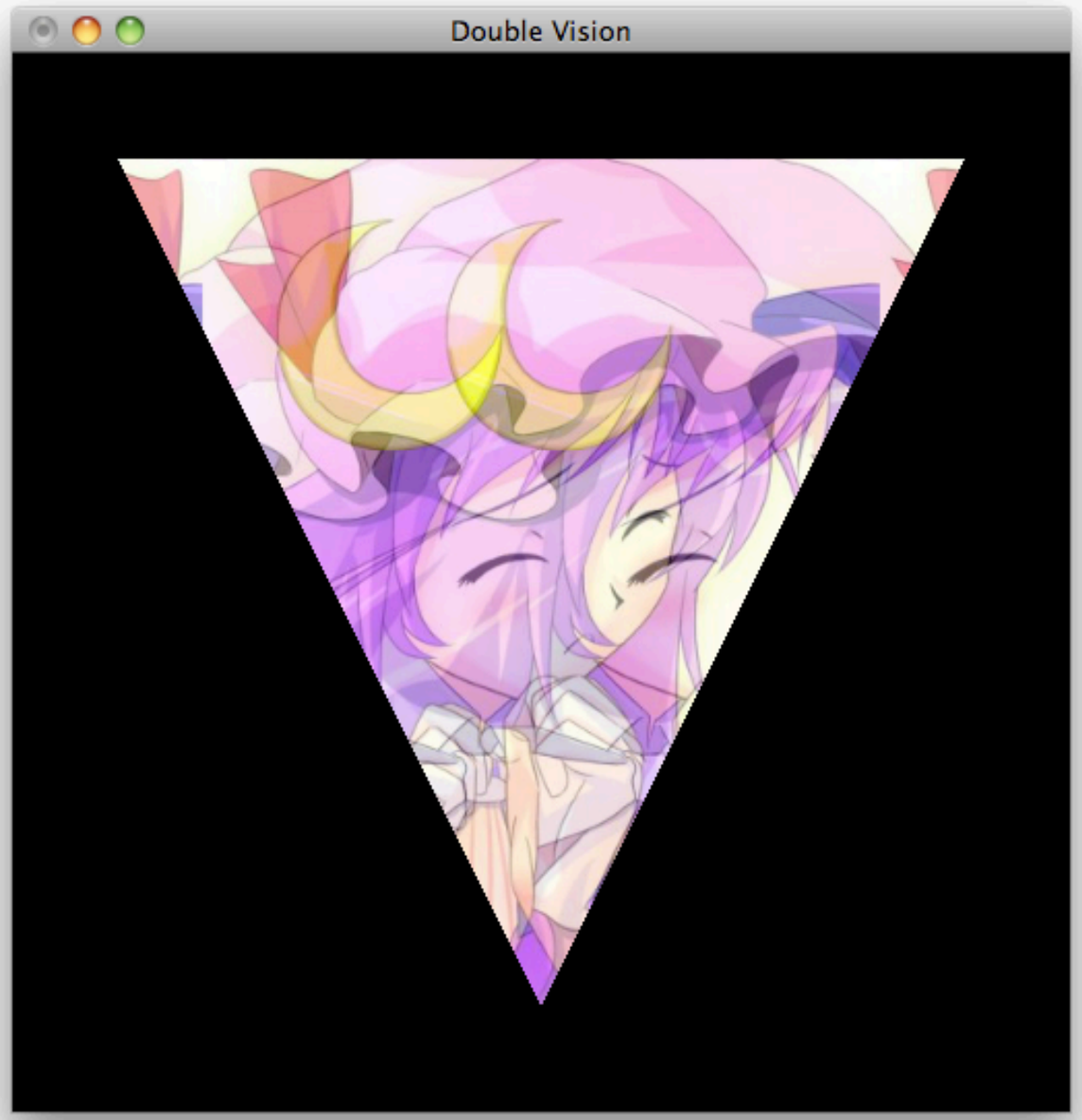
- ตัวอย่าง: คำสั่งต่อไปนี้ทำการ copy ค่า texture coordinate ที่กำหนดโดย glTexCoord ไปยังตัวแปร gl_TexCoord[0]

```
gl_TexCoord[0] = gl_MultiTexCoord0;
```

ตัวแปรสำหรับ Texture Coordinate

- ใน fragment shader
    - สามารถอ่านค่า texture coordinate ได้จาก gl_TexCoord[k] เมื่อ k มีค่าตั้งแต่ 0 ถึง gl_MaxTextureCoords
    - ค่าของ gl_TexCoord[k] จะเป็นค่าที่ได้จากการ interpolate ค่า gl_TexCoord[k] ที่เรากำหนดใน vertex program

ตัวอย่าง: Double Vision



ตัวอย่าง: Double Vision

- หลักการ
  - เมื่อได้รับ texture coordinate มา ให้แปลงมันเป็น texture coordinate ใหม่สองค่า
  - ในกรณีนี้เราแปลงโดยให้ texture coordinate อันหนึ่งเยื้องไปทางขวา อีกอันเยื้องไปทางซ้าย
  - เอา texture coordinate ใหม่ไปอ่านสีจาก texture มาสองสี
  - แล้วเอาสีที่ได้มาเฉลี่ยกันเป็นสีของ fragment

โค้ดของ Vertex Shader

```
#version 110

void main()
{
    gl_Position = ftransform();
    gl_TexCoord[0] = gl_MultiTexCoord0;
}
```

โค้ดของ Fragment Shader

```
#version 110

uniform sampler2D texture;
uniform vec2 left_separation;
uniform vec2 right_separation;

void main()
{
    vec2 left_tex_coord = gl_TexCoord[0].st + left_separation;
    vec2 right_tex_coord = gl_TexCoord[0].st + right_separation;
    vec4 left_color = texture2D(texture, left_tex_coord);
    vec4 right_color = texture2D(texture, right_tex_coord);
    gl_FragColor = (left_color + right_color) * 0.5;
}
```

การกำหนดค่าให้ตัวแปรแบบ sampler2D

- มีอยู่ 4 ขั้นตอน

1. สั่ง glEnable(GL_TEXTURE_2D)

2. เลือก active texture unit

3. Bind texture ที่ต้องการส่งให้ sampler2D

4. สั่ง glUniform1i แล้วบอกเลข location ของตัวแปร และ texture unit ที่ bind texture ในข้อ 3 ไป

2. **เลือก** Active Texture Unit

- ใน OpenGL เวอร์ชันในปัจจุบัน
  เราสามารถ bind texture ได้มากกว่าหนึ่ง texture ในคราวเดียว

- OpenGL มีโครงสร้างข้อมูลที่เรียกว่า texture unit
  ซึ่งทำหน้าที่เก็บสถานะของ texture ต่างๆ ที่เราใช้

- Texture unit หนึ่งตัวสามารถรับผิดชอบ texture ได้เพียงผืนเดียว

- ใช้หลาย texture พร้อมกับต้องใช้หลาย texture unit พร้อมๆ กัน

2. เลือก Active Texture Unit

- Texture unit ใน OpenGL จะถูกระบุด้วยค่าคงตัว GL_TEXTUREk
  โดยที่ k มีค่าตั้งแต่ 0 ถึง 31

- รวมแล้ว OpenGL มี texture unit ให้ใช้สูงสุด 32 ตัว
  แต่การ์ดจอมีให้ใช้น้อยกว่านั้น

- หากต้องการทราบว่ามี texture unit ให้ใช้กี่ตัว คุณสามารถใช้โค้ดต่อไปนี้ได้

```
int texture_unit_count;
glGetIntegerv(GL_MAX_TEXTURE_UNITS, &texture_unit_count);
printf("Number of texture units = %d\n", texture_unit_count);
```

2. **เลือก** Active Texture Unit

- เวลาจะเปลี่ยนแปลงข้อมูลของ texture unit (เช่น เปลี่ยน texture ที่ bind)
  เราจะต้องบอกก่อนว่าต่อไปเราจะใช้ texture unit ตัวไหน

- ใช้ฟังก์ชัน
  void **glActiveTexture**(GLenum *texture_unit*);

  - unit เป็นค่าคงที่สำหรับแทน texture unit แต่ละตัว
    มีค่าตั้งแต่ GL_TEXTURE0 ถึง GL_TEXTUREk
    เมื่อ k เป็นจำนวน texture unit ที่มากที่สุดที่มีให้ใช้

- ตัวอย่าง: ถ้าอยากใช้ texture unit ตัวที่ 3 ก็สั่ง

```
glActiveTexture(GL_TEXTURE3);
```

4. กำหนดโดยใช้ glUniform1i

- หลังจาก bind texture เสร็จแล้วให้สั่ง glUniform1i
  โดยให้หมายเลขของ texture unit ที่ใช้ควบคุม texture ที่เพิ่งจะ bind ไป

- ถ้าใช้ GL_TEXTUREk แล้วให้ป้อนค่า k เข้า glUniform1i

- ตัวอย่าง: โค้ดในกรณีที่ใช้ texture unit หมายเลข 3

```
glEnable(GL_TEXTURE_2D);
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D, tex_id);
glUniform1f(texture_location, 3);
```

โค้ดตอนวาดรูปของโปรแกรม Double Vision

```
glUseProgram(prog_id);
glUniform2f(left_separation_location, 0.1f, 0);
glUniform2f(right_separation_location, -0.1f, 0);

glEnable(GL_TEXTURE_2D);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex_id);
glUniform1f(texture_location, 0);

glBegin(GL_TRIANGLES);
    glTexCoord2f(0, 0);
    glVertex2f(-0.8, 0.8);

    glTexCoord2f(1, 0);
    glVertex2f(0.8, 0.8);

    glTexCoord2f(0.5, 1);
    glVertex2f(0.0, -0.8);
glEnd();
```

Varying Variables

Varying Variable

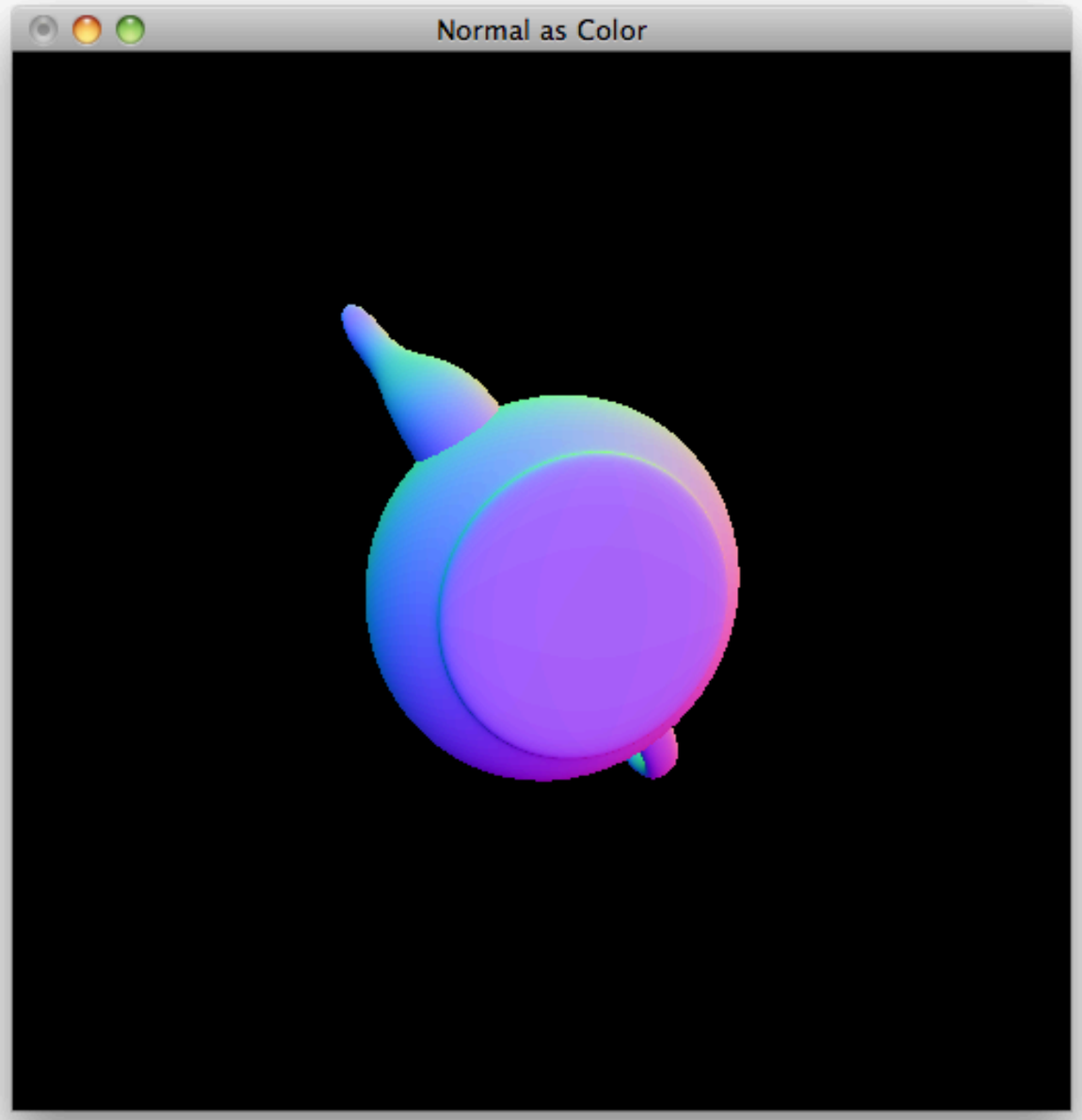
- ตัวแปรแบบ varying คือตัวแปรที่
  - ผู้ใช้สามารถกำหนดค่าให้ใน vertex shader
  - เมื่อเรียกใช้ตัวแปรเดียวกันใน fragment shader จะได้ค่าที่เป็นผลลัพธ์จากการ interpolate ค่าที่กำหนดให้ใน vertex shader
- จริงๆ แล้ว gl_TexCoord[k] ต่างๆ เป็นตัวแปรแบบ varying

Varying Variable

- ต้องกำหนดไว้เป็นตัวแปรแบบ global (กำหนดอยู่นอกฟังก์ชันต่างๆ)
- ไวยากรณ์
  - varying <ชนิดตัวแปร> <ชื่อตัวแปร>;
- ตัวอย่าง

```
varying vec3 color;

void main()
{
    // Some code here...
}
```

ตัวอย่าง: ใช้สีแทน normal



ตัวอย่าง: ใช้สีแทน normal

- หาค่าสีจาก normal ใน world space
- คำนวณจากสูตร

$$\text{สี} = (\text{normal} + (1,1,1)) / 2$$

โค้ดของ Vertex Shader

```
#version 110

varying vec3 normal;

void main()
{
    gl_Position = ftransform();
    normal = (gl_NormalMatrix * gl_Normal).xyz;
}
```

โค้ดของ Fragment Shader

```
#version 110

varying vec3 normal;

void main()
{
    vec3 n = normalize(normal);
    vec3 color = (n + vec3(1.0, 1.0, 1.0)) * 0.5;
    gl_FragColor = vec4(color, 1);
}
```

ตัวแปรที่เกี่ยวข้องกับ normal

- ใน vertex shader ตัวแปร gl_Normal
  มีค่าเท่ากับค่าที่เรากำหนดได้ด้วยคำสั่ง glNormal ในภาษา C

- ใน vertex shader ตัวแปร gl_NormalMatrix
  มีค่าเท่ากับ inverse transpose ของ modelview matrix
  (ซึ่งคือ เมตริกซ์ที่คุณกำหนดค่าให้ด้วยคำสั่งต่างๆ หลังจากสั่ง
  glMatrixMode(GL_MODELVIEW))

- ฟังก์ชัน normalize ไม่เกี่ยวกับ normal
  แต่เป็นฟังก์ชันที่คืนเวกเตอร์หนึ่งหน่วยที่มีทิศทางเดียวกับ argument

  - เราต้อง normalize เวกเตอร์หลังมันถูก interpolate เพราะมันอาจจะไม่ใช่เวกเตอร์หนึ่งหน่วยอีกต่อไป

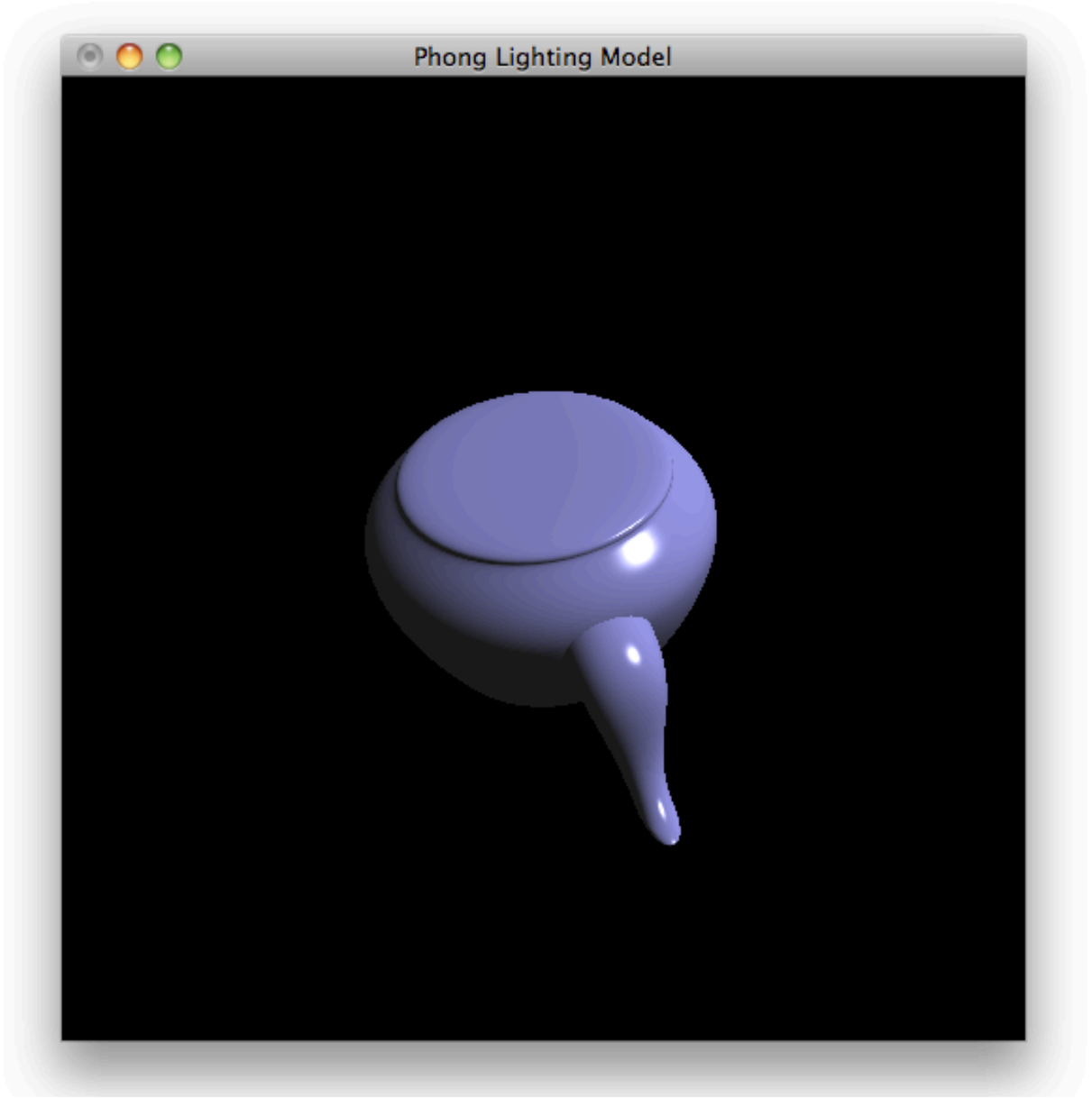
โค้ดตอนวาดรูป

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60, 1, 0.1, 100);
gluLookAt(0,0,5,0,0,0,0,1,0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glUseProgram(prog_id);

glPushMatrix();
glRotated(rotate_z, 0, 0, 1);
glRotated(rotate_y, 0, 1, 0);
glRotated(rotate_x, 1, 0, 0);
glutSolidTeapot(1.0);
glPopMatrix();
```

โค้ดตอนวาดรูป

- สังเกต
  - เราต้องการ normal ใน world space
  - OpenGL ไม่มีเมตริกซ์ที่เก็บ model transform โดยตรง
    แต่มี modelview transform ซึ่งเปลี่ยน object space เป็น eye space
  - ฉะนั้นโค้ดตอนวาดรูป จึงย้าย gluLookAt (view transform)
    ไปคูณต่อท้าย gluPerspective (projection transform)
    เพื่อทำให้ modelview transform กลายเป็น model transform เฉยๆ
  - สามารถใช้ trick นี้ได้ทุกครั้งที่ต้องการข้อมูลใน worldspace
    แต่ไม่ต้องการข้อมูลใน eye space

ตัวอย่าง: การจำลอง Phong Lighting Model



ตัวอย่าง: การจำลอง Phong Lighting Model

- ผู้ใช้สามารถกำหนด
  - สี ambient, diffuse, specular, และ emission
  - shininess ของพื้นผิว
  - สี และทิศทางของแหล่งกำหนดแสงแบบทิศทาง
  - ตำแหน่งของตา (สำหรับใช้คำนวณ specular highlight)
- ตัวแปรทั้งหมดข้างบนนี้เป็น uniform variables

โค้ดของ Vertex Shader

```
varying vec3 normal;
varying vec3 position;

void main()
{
    gl_Position = ftransform();
    normal = gl_NormalMatrix * gl_Normal;
    position = (gl_ModelViewMatrix * gl_Position).xyz;
}
```

โค้ดของ Fragment Shader

```
uniform vec3 material_ambient;
uniform vec3 material_diffuse;
uniform vec3 material_specular;
uniform vec3 material_emission;
uniform float material_shininess;

uniform vec3 light_ambient;
uniform vec3 light_intensity;
uniform vec3 light_direction;

uniform vec3 eye;

varying vec3 normal;
varying vec3 position;
```

โค้ดของ Fragment Shader (ต่อ)

```
void main()
{
    vec3 l = normalize(light_direction);

    vec3 color = material_emission;
    color += light_ambient * material_ambient;

    vec3 n = normalize(normal);
    float cos_theta = max(dot(n, l), 0.0);
    color += light_intensity * material_diffuse * cos_theta;

    vec3 r = n * 2.0 * dot(n, l) - l;
    vec3 v = normalize(eye - position);
    float cos_alpha = dot(r, v);
    if (cos_theta > 0.0 && cos_alpha > 0.0)
        color +=  light_intensity * material_specular * pow(cos_alpha,
material_shininess);

    gl_FragColor = vec4(color, 1);
}
```

โค้ดตอนวาดรูป

```
glUniform3f(eye, 0, 0, 5);

glUniform3f(material_ambient, 0.1f, 0.1f, 0.1f);
glUniform3f(material_diffuse, 0.5f, 0.5f, 0.8f);
glUniform3f(material_specular, 1.0f, 1.0f, 1.0f);
glUniform3f(material_emission, 0, 0, 0);
glUniform1f(material_shininess, 100.0f);

glUniform3f(light_ambient, 1, 1, 1);
glUniform3f(light_intensity, 1, 1, 1);
glUniform3f(light_direction, 1, 1, 1);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0f, 1, 0.1, 100);
gluLookAt(0,0,5,0,0,0,0,1,0);
```

โค้ดตอนวาดรูป (ต่อ)

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0f, 1, 0.1, 100);
gluLookAt(0,0,5,0,0,0,0,1,0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotated(rx, 1, 0, 0);
```