단국대학교 멀티미디어공학 그래픽스 프로그래밍 기말고사 (2014년 봄학기) 2014년 6월 10일

학과 _____ 학번 _____ 이름 _____

기말고사

담당교수: 단국대학교 멀티미디어공학전공 박경신

- **답은 반드시 답안지에 기술할 것. 공간이 부족할 경우 반드시 답안지 몇 쪽의 뒤에 있다고 명기한 후 기술할 것. 그 외의 경우의 답안지 뒤쪽이나 연습지에 기술한 내용은 답안으로 인정 안 함. 답에는 반드시 네모를 쳐서 확실히 표시할 것.**
- **답안지에 학과, 학번, 이름 외에 본인의 암호를 기입하면 성적공고시 학번 대신 암호를 사용할 것임.**

1. Vertex shader에서 3차원 물체의 정점 좌표(vertex position)에 Model, View, Projection 변환을 한다. Model, View, Projection이 무엇인지 설명하고, 각각 glm 함수 예를 들어라. (10점)

model 변환은 model space에서 구성한 3차원 물체의 정점 좌표를 world space로 변환해주는 것이다. glm::translate, glm::rotate, glm::scale 등

view 변환은 world space에 배치된 3차원 물체의 정점 좌표를 카메라의 관점인 view space으로 바꿔준다. glm::lookAt

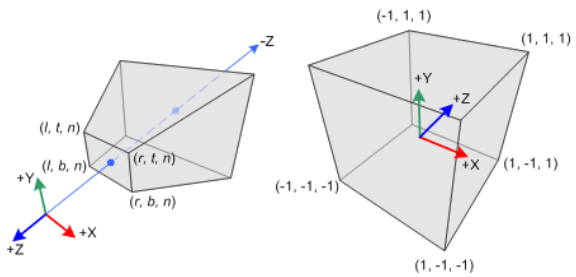
projection 변환은 view space에 있는 3차원 물체의 정점 좌표를 정규 장치 좌표계 (normalized device coordinate)로 바꿔준다. 정규 장치 좌표계에서 투영면에 투영시킨다. glm::frustum, glm::perspective 등

2. 2차원 그래픽스에서 자주 사용하는 isometric projection과 3차원 그래픽스에서 자주 사용하는 perspective projection이 무엇인지 그 특징을 설명하라. (5점)

등축투영 (Isometric projection)은 투영면이 객체의 모서리에서 만나는 세 개의 주 면에 대해서 대칭으로 놓여져서 3차원의 효과를 주는 직교투영(Orthographic projection)/축측투영 (Axonometric projection)이다.

투시 투영 (Perspective projection)은 소실점(vanishing point)을 한 개 이상 가지고 있으며 객체가 관측자로부터 멀리 떨어질 수록 크기가 축소되어 원근감을 생성해 주는 투영이다.

3. 원근 정규화(perspective normalization)를 설명하라. glm::frustum(left, right, bottom, top, near, far) 함수가 생성하는 투영 행렬에서 원근 정규화는 A, B, C 중 어느 부분인지? (10점)



$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

A B C

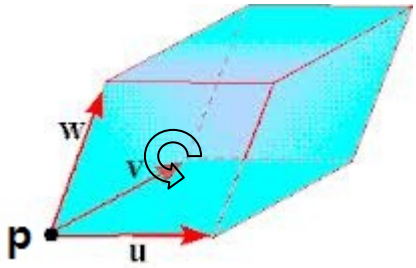
C는 원근 정규화 (perspective normalization).
원근 정규화는 원근 투영을 직교투영으로 바꾸는 작업이다.
즉, x=±z, y=±z, z=near/far =>x=±1, y=±1, z=±1로 정규화 (normalization).
이 과정에서 원근감이 생성된다.

학과 _____ 학번 _____ 이름 _____

4. 현재 카메라의 위치(position)가 (1, 0, 5), 카메라가 바라보는 곳(at)이 (0, 0, 5)이고, 카메라의 위쪽 방향(up)이 (0, 1, 0)일 때, view matrix를 계산하라. (10점)

$$View = \begin{pmatrix} 0 & 0 & -1 & 5 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. 다음은 Parallelpiped 클래스 코드 일부이다. 다음 빈 칸에 코드를 채우시오. (15점)




```
void Parallelpiped::init()
{
        glm::vec3 pv = p + v;
        glm::vec3 pw = p + w;
        glm::vec3 pvw = ____p + v + w;_____

        glm::vec3 rightNormal = _____glm::cross(v, w);_____
        rightNormal = glm::normalize(rightNormal);
        glm::vec3 leftNormal = _____-rightNormal;_____
// 중간 생략…
        // Left face (왼쪽 면)
        vbo.addData(&pv[0], sizeof(glm::vec3)); // vertex position
        vbo.addData(&glm::vec2(0.0f, 0.0f), sizeof(glm::vec2)); // vertex texture coordnate
        vbo.addData(&leftNormal[0], sizeof(glm::vec3)); // vertex normal

        vbo.addData(___&p[0] ___, sizeof(glm::vec3)); // vertex position
        vbo.addData(___&glm::vec2(1.0f, 0.0f) ___, sizeof(glm::vec2)); // vertex texture coordnate
        vbo.addData(___&leftNormal[0] ___, sizeof(glm::vec3)); // vertex normal
```

```
        vbo.addData(___&pw[0] ___, sizeof(glm::vec3)); // vertex position
        vbo.addData(___&glm::vec2(1.0f, 1.0f) ___, sizeof(glm::vec2)); // vertex texture coordnate
        vbo.addData(___&leftNormal[0] ___, sizeof(glm::vec3)); // vertex normal

        vbo.addData(&pv[0], sizeof(glm::vec3)); // vertex position
        vbo.addData(&glm::vec2(0.0f, 0.0f), sizeof(glm::vec2)); // vertex texture coordnate
        vbo.addData(&leftNormal[0], sizeof(glm::vec3)); // vertex normal

        vbo.addData(___&pw[0] ___, sizeof(glm::vec3)); // vertex position
        vbo.addData(___&glm::vec2(1.0f, 1.0f) ___, sizeof(glm::vec2)); // vertex texture coordnate
        vbo.addData(___&leftNormal[0] ___, sizeof(glm::vec3)); // vertex normal

        vbo.addData(___&pvw[0] ___, sizeof(glm::vec3)); // vertex position
        vbo.addData(___&glm::vec2(0.0f, 1.0f) ___, sizeof(glm::vec2)); // vertex texture coordnate
        vbo.addData(___&leftNormal[0] ___, sizeof(glm::vec3)); // vertex normal
    // 중간 생략…
    }
```

6. 다음 OpenGL 텍스쳐 매핑과 블렌딩 질문에 답하라. (20점)

```
GLuint initTexture(const char * flename, GLint sWrap=GL_REPEAT, GLint tWrap=GL_REPEAT)
{
   unsigned char *imageBuffer;
   int imgWidth = 0, imgHeight = 0, num = 0;
   imageBuffer = simage_read_image(filename, &imgWidth, &imgHeight, &num);
   GLuint textureID;
   if (imageBuffer != NULL) {
      glGenTextures(1, &textureID);
      glBindTexture(GL_TEXTURE_2D, textureID);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, sWrap);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, tWrap);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      glTexImage2D(GL_TEXTURE_2D, 0, num == 3 ? GL_RGB : GL_RGBA, imgWidth, imgHeight,
                   0, num == 3 ? GL_RGB : GL_RGBA, GL_UNSIGNED_BYTE, imageBuffer);
      glBindTexture(GL_TEXTURE_2D, 0);
   }
   return textureID;
}

void init()
{  // 중간 생략…
   texture[0] = initTexture("opengl.jpg", __GL_CLAMP_TO_EDGE__, __GL_REPEAT__);
   texture[1] = initTexture("opengl.jpg", __GL_REPEAT__, __GL_REPEAT__);
   texture[2] = initTexture("opengl.jpg", __GL_MIRRORED_REPEAT__, __GL_MIRRORED_REPEAT__);
   squareVAO = setSquareData();
```

```
}

GLuint setSquareData()
{
   squareVertices.push_back(glm::vec3(-1.0f, -1.0f, 0.0f));
   squareVertices.push_back(glm::vec3( 1.0f, -1.0f, 0.0f));
   squareVertices.push_back(glm::vec3( 1.0f, 1.0f, 0.0f));
   squareVertices.push_back(glm::vec3(-1.0f, 1.0f, 0.0f));
   squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
   squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
   squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
   squareNormals.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
   squareTextureCoords.push_back(glm::vec2(___0.0f, -1.0f___));
   squareTextureCoords.push_back(glm::vec2( ___3.0f, -1.0f___));
   squareTextureCoords.push_back(glm::vec2( __3.0f, 2.0f___));
   squareTextureCoords.push_back(glm::vec2(__0.0f, 2.0f___));
   squareIndices.push_back(0); squareIndices.push_back(1); squareIndices.push_back(2);
   squareIndices.push_back(0); squareIndices.push_back(2); squareIndices.push_back(3);
   // 중간 생략…
   return vao;
}

void drawTextureQuad(int textureID)
{
   glActiveTexture(GL_TEXTURE0);
   glBindTexture(GL_TEXTURE_2D, textureID);
   glBindVertexArray(squareVAO);
   glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
   glBindVertexArray(0);
   glBindTexture(GL_TEXTURE_2D, 0);
}

void draw()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   // 중간 생략…
   drawTextureQuad(texture[0]); // 중간 생략…
   drawTextureQuad(texture[1]); // 중간 생략…
   drawTextureQuad(texture[2]); // 중간 생략…
}
```

1) 출력 결과가 아래의 그림과 같을 때, 위의 코드에 빈칸 (즉, 텍스쳐 좌표와 wrap 필터 GL_REPEAT | GL_CLAMP_TO_EDGE | GL_MIRRORED_REPEAT)을 채우시오. (10점)



학과 _____ 학번 _____ 이름 _____

2) *glBlendFunc(...);*는 블렌딩 필터를 지정하는 함수이다. 아래의 표를 채워라. (10점)

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);	*As*Cs + (1-As)*Cd*	*알파 블렌딩*
glBlendFunc(GL_ONE, GL_ONE);	*1*Cs + 1*Cd = Cs + Cd*	*덧셈 블렌딩*
glBlendFunc(GL_ZERO, GL_SRC_COLOR)	*0*Cs + Cs*Cd = Cs*Cd*	*곱셈 블렌딩*
glBlendFunc(GL_ONE, GL_ZERO);	*1*Cs + 0*Cd = Cs*	*No blending*
glBlendFunc(GL_ZERO, GL_ONE);	*0*Cs + 1*Cd = Cd*	*Draw background only*
glBlendFunc(GL_SRC_ALPHA, GL_ONE);	*As*Cs + Cd*	*Brighten the scene*
glBlendFunc(GL_ZERO, GL_SRC_ALPHA);	*0*Cs + As*Cd = As*Cd*	*Darken the scene*

7. 다음은 조명 (lighting) Fragment shader 코드의 일부이다. 아래의 질문에 답하시오. (20점)

```
void main()
{
    vec3 MaterialDiffuseColor = texture2D(gTextureSampler, TexCoordPass).rgb;
    vec3 MaterialAmbientColor = vec3(0.1,0.1,0.1) * MaterialDiffuseColor;
    vec3 MaterialSpecularColor = vec3(0.8,0.8,0.8);
    float distance = length(gLightPosition - PositionWorldPass);
    vec3 N = normalize(NormalViewPass);
    vec3 L = normalize(LightDirectionViewPass);
    float cosTheta = clamp(dot(N, L), 0, 1);
    vec3 V = normalize(EyeDirectionViewPass);
    vec3 R = reflect(-L, N);
    float cosPi = clamp(dot(V,R), 0, 1);
    float shininess = 5.0;
    Color = MaterialAmbientColor +
        MaterialDiffuseColor * gLightColor * cosTheta / (distance*distance) +
        MaterialSpecularColor * gLightColor * pow(cosPi,shininess) / (distance*distance);
}
```

1) 램버트 법칙 (Lambertian Law)을 자세히 설명하라. 위의 코드에서 찾아라. (5점)

램버트 코사인 법칙은 난반사 (diffuse reflection)에서 면의 밝기가 입사각 (즉, 광원 벡터와 법선 벡터의 사이각)의 코사인에 정비례 함을 말하는 것이다. 즉, 면이 서 있는 방향에 따라 차등적 밝기를 제공하여 입체감을 부여할 수 있다.
Fragment shader 코드에서 cosTheta는 법선벡터 (N)과 광원벡터 (L)간의 내적은 두 벡터간의 입사각의 코사인과 비례하며, diffuse reflection과의 곱에서 사용된다.

학과 _____ 학번 _____ 이름 _____

2) 정반사 (Specular reflection)가 무엇인지 자세히 설명하라. 정반사에 영향을 미치는 벡터 V와 R과 광택계수 (shininess coefficient)가 무엇인지 설명하라. (5점)

정반사(specular reflection)란 반질반질한 표면에서 반사되는 빛으로, 광택계수(shininess coefficient)를 통하여 시점 (V)이 정반사(R)에서 벗어남에 따라 반사되는 빛의 세기가 약해지는 속도를 조절한다.
V는 정점의 위치에서 카메라의 위치로 향하는 시점 벡터 (view vector)
R은 정점의 위치에서 광원벡터 (L)가 정점의 법선벡터 (N)에 정반사된 벡터 (reflection vector)

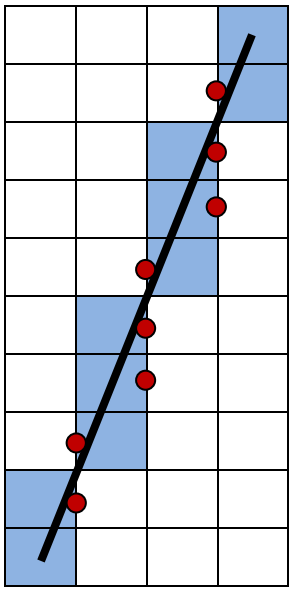
3) 위의 코드는 Per-pixel lighting(일명, Phong shading)이다. Per-vertex lighting과의 차이점은 무엇인가? Per-pixel lighting과 Per-vertex lighting을 자세히 설명하라. (10점)

http://dis.dankook.ac.kr/lectures/cg13/entry/ShadedQuad

Per-pixel lighting (일명, Phone shading)은 각 pixel/fragment마다 정점의 법선 벡터를 보간하여 lighting이 계산되는 방식으로, 곡면의 기울기가 복원되며 정확히 경면광 (specular lighting)을 부여할 수 있음.

Per-vertex lighting (일명, Gouraud shading)은 퐁 조명 모델 공식을 이용하여 계산된 정점의 색으로부터 내부 면의 색을 선형보간함. 면에 전체적으로 부드러운 음영을 제공함. 그러나 경면광을 감안하지 않아(실제적인 정점의 법선벡터와 근사적으로 계산된 법선벡터가 완전히 일치하지 않기 때문) 뭉개지는 현상이 나타남.

8. 현재 래스터기의 표준 알고리즘이 된 Bresenham line-drawing algorithm (일명, Midpoint algorithm)을 사용하여 (1, 1)과 (4, 10)을 연결하는 선분을 그릴 때에, 왼쪽 그림에서 중점(M)의 위치와 선택된 픽셀을 색칠하라. 그리고 오른쪽 표에서는 선택된 픽셀 값과 결정변수(D)가 음수/양수인지 표시하라. (10점)



(1, 1)	D<0
(1, 2)	D>0
(2, 3)	D<0
(2, 4)	D<0
(2, 5)	D>0
(3, 6)	D<0
(3, 7)	D<0
(3, 8)	D>0
(4, 9)	D<0
(4, 10)	

9. 다음은 계층적 변환 (Hierarchical transformation) 구조를 가진 SimpleObject 클래스이다.

학과 _____ 학번 _____ 이름 _____

이 클래스가 동작하는 그 출력 결과를 그림으로 그려라. (회전 방향을 명확히 표시해 줄 것) (extra 10점).

```
void SimpleObject::init()
{
part1 = Cube(glm::vec3(0, 0, 0), 2.0f);
part2 = Torus(glm::vec3(0, 0, 0), 1.0f, 0.25f, 16, 16);
}

void SimpleObject::draw(Program* p, glm::mat4 projection, glm::mat4 view, glm::mat4 model)
{
p->useProgram();
p->setUniform("gProjection", projection);
p->setUniform("gView", view);

glm::mat4 m1 = model * partTransform1;
p->setUniform("gModel", m1);
part1.draw();

glm::mat4 m2 = m1 * partTransform2;
p->setUniform("gModel", m2);
part2.draw();

glm::mat4 m3 = m1 * partTransform3;
p->setUniform("gModel", m3);
part2.draw();
}

bool SimpleObject::update(float deltaTime)
{
angle = angle - 180.0f * (float) (deltaTime) * 0.0001f;

partTransform1 = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0, 1, 0));

partTransform2 = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 2.0f, 0.0f)) *
glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0, 0, 1));

partTransform3 = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -2.0f, 0.0f)) *
glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0, 0, 1));

return true;
}
```