단국대학교 멀티미디어공학 그래픽스 프로그래밍 기말고사 (2013년 봄학기) 2013년 6월 12일

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. 다음은 OpenGL 프로그램의 일부이다. 아래의 질문에 답하시오. (70점)

```
GLuint initTexture(const char * flename, GLint wrap, GLint mag, GLint min)
{
   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, wrap);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, mag);
      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, min);
      glTexImage2D(GL_TEXTURE_2D, 0, num == 3 ? GL_RGB : GL_RGBA, imgWidth, imgHeight,
                        0, num == 3 ? GL_RGB : GL_RGBA, GL_UNSIGNED_BYTE, imageBuffer);
      if (minfilter == GL_NEAREST || minfilter == GL_LINEAR) { }
      else {
            glGenerateMipmap(GL_TEXTURE_2D);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_BASE_LEVEL, 0);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_LEVEL, 4);
      }
      glBindTexture(GL_TEXTURE_2D, 0);
   }
}

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(-1.0f, -1.0f));
   squareTextureCoords.push_back(glm::vec2( 2.0f, -1.0f));
   squareTextureCoords.push_back(glm::vec2( 2.0f, 3.0f));
   squareTextureCoords.push_back(glm::vec2(-1.0f, 3.0f));
   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;
}

GLuint setCylinderData(float radius=1.0f, float height=2.0f, int slices=16)
{
    glm::vec3 vertex;
    float xTexCoord = 0.0f;
    float theta = (float) (2*M_PI/slices);
    for (int i=0; i<=slices; i++) {
        vertex[0] = radius * cosf(theta * i);
        vertex[1] = -height/2.0f;
        vertex[2] = radius * sinf(theta * i);
        cylinderVertices.push_back(vertex);
        cylinderNormals.push_back(_____1_____);
        cylinderTextureCoords.push_back(glm::vec2(xTexCoord, 0.0f));
        numCylinderVertices++;
        vertex[0] = radius * cosf(theta * i);
        vertex[1] = height/2.0f;
        vertex[2] = radius * sinf(theta * i);
        cylinderVertices.push_back(vertex);
        cylinderNormals.push_back(_____2_____);
        cylinderTextureCoords.push_back(glm::vec2(xTexCoord, 1.0f));
        numCylinderVertices++;
        xTexCoord += 1.0f/slices;
    }
    // 중간 생략…
    return vao;
}

void init()
{  // 중간 생략…
    texture[0] = initTexture("opengl.jpg", GL_REPEAT, GL_NEAREST, GL_NEAREST);
    texture[1] = initTexture("opengl.jpg", GL_REPEAT, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR);
    texture[2] = initTexture("opengl.jpg", GL_REPEAT, GL_LINEAR, GL_LINEAR);
    texture[3] = initTexture("opengl.jpg", GL_MIRRORED_REPEAT, GL_LINEAR, GL_LINEAR);
    texture[4] = initTexture("opengl.jpg", GL_CLAMP_TO_EDGE, GL_LINEAR, GL_LINEAR);
    texture[5] = initTexture("rock.jpg", GL_REPEAT, GL_LINEAR, GL_LINEAR);
    texture[6] = initTexture("lightmap.png", GL_REPEAT, GL_LINEAR, GL_LINEAR);
    squareVAO = setSquareData();
}

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);
  projection = glm::perspective(45.0f, 1.0f, 1.0f, 100.0f);
  view = glm::lookAt(glm::vec3(4, 0, 0), glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));
```

```
    // 중간 생략…
    drawTextureQuad(texture[0]); // 중간 생략…
    drawTextureQuad(texture[1]); // 중간 생략…
    drawTextureQuad(texture[2]); // 중간 생략…
    drawTextureQuad(texture[3]); // 중간 생략…
    drawTextureQuad(texture[4]); // 중간 생략…
    // 중간 생략…
    drawTextureQuad(texture[5]);
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_BLEND);
    glBlendFunc(GL_ONE, GL_ONE);
    drawTextureQuad(texture[6]);
    glDepthFunc(GL_LESS);
    glDisable(GL_BLEND);
}
```

1) 위의 코드에서 view matrix와 projection matrix가 무엇인지 자세히 설명하라. (5점)

view matrix는 world space에 배치된 3차원 물체의 정점 좌표를 카메라의 관점인 view space(카메라 좌표계)로 바꿔주는 행렬이다.

projection matrix는 view space에 있는 3차원 물체의 정점 좌표를 2차원 투영면 (projection plane 즉, 카메라의 영상면)에 투영해주는 행렬이다.

2) 위의 코드에서 *glm::lookAt(glm::vec3(4, 0, 0), glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));* 함수가 생성한 view matrix를 유도하라. (10점)

peudo code: n = eye – at; n.normalize(); u = up x n; u.normalize(); v = n x u; v.normalize();

View[0][0] = u[0]; View[1][0] = u[1]; View[2][0] = u[2]; View[3][0] = - u ・eye;

View[0][1] = v[0]; View[1][1] = v[1]; View[2][1] = v[2]; View[3][1] = - v ・eye;

View[0][2] = n[0]; View[1][2] = n[1]; View[2][2] = n[2]; View[3][2] = - n ・eye;

View[0][3] = 0; View[1][3] = 0; View[2][3] = 0; View[3][3] = 1;

n = (1, 0, 0) u = (0, 0, -1) v = (0, 1, 0)

$$View = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

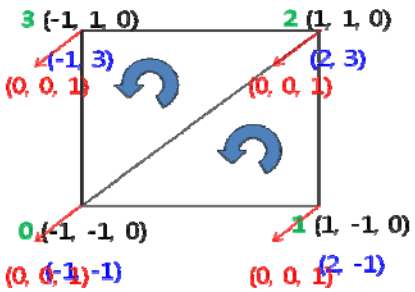
http://dis.dankook.ac.kr/lectures/cg13/entry/OpenGL-Camera-Movement

3) 위에 코드에서 setCylinderData(…) 함수 내부에 cylinderNormals.push_back(__1__)과 (__2__) 안에 들어갈 내용은 무엇인가? (5점)

(1)과 (2) 모두 같은 법선 벡터 값을 넣어준다.

```
vertex[0] = radius * cosf(theta*i); vertex[1] = 0.0f; vertex[2] = radius * sinf(theta*i);
vertex = glm::normalize(vertex);
cylinderNormals.push_back(vertex);
```

4) **위의 코드에서 glDrawElements(...) 함수가 어떻게 사각형을 그리는지 (정점 위치, 법선 벡터, 텍스쳐 좌표, 인덱스를 표시할 것)을 설명하라. (5점)**



vertex buffer를 생성해서 4개의 정점 위치(position), 법선벡터(normal), 텍스쳐좌표(texture coordinates) 정보를 버퍼에 넣어서 초기화
index buffer를 생성해서 6개의 인덱스 정보를 버퍼에 넣어서 초기화
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
4개의 정점 정보를 가지고 있는 정점리스트(즉 4개)와 3개씩 2개의 삼각형에 대한 인덱스 리스트(즉 6개)를 사용하여 그림을 그리는 방식

5) **텍스쳐 맵핑의 밉맵 필터 (Mipmap filter)를 자세히 설명하라. 그리고 위 코드에서 밉맵과 관련된 부분을 모두 찾아서 적어라. (5점)**

http://dis.dankook.ac.kr/lectures/cg13/entry/ModernOpenGL-Mipmap

밉맵필터는 일련의 축소된 크기의 밉맵 택스쳐를 생성하여, 작은 객체에 텍스쳐 맵핑을 할 시 보간 문제를 줄여주는 필터임.

glGenerateMipmap(GL_TEXTURE_2D);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_BASE_LEVEL, 0);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_LEVEL, 4);

texture[1] = initTexture("opengl.jpg", GL_REPEAT, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR);

6) **위 코드에서 텍스쳐 매핑의 확대, 축소 필터링(Filtering)이 무엇인지, texture[0], texture[1], texture[2]로 바인딩하여 그렸을 때 나타나는 차이점이 무엇인지 자세히 설명하라. (10점)**

http://dis.dankook.ac.kr/lectures/cg13/entry/OpenGL-Texture-Mapping-Filtering-Environment-Parameters

확대 필터 (MAG_FILTER): 텍스쳐 이미지를 확대하여 보다 넓은 면적의 다각형 영역에 매핑하고자 할 때 사용되는 필터링

축소 필터 (MIN_FILTER): 텍스쳐 이미지를 축소하여 보다 작은 면적의 다각형 영역에 매핑하고자 할 때 사용되는 필터로, 선형 (GL_LINEAR) 필터링

확대(MAG_FILTER)와 축소(MIN_FILTER)와 밉맵(MIPMAP_FILTER)에 다른 필터링 사용

Texture[0]은 확대필터와 축소필터에 포인트 샘플링 방식인 최근점 필터링 (GL_NEAREST) 사용하여 에일리어싱(aliasing) 문제가 발생함

Texture[1]는 확대필터와 축소필터에 선형필터링을 사용하고 밉맵필터로 선형필터링를 추가함

Texture[2]은 확대필터와 축소필터에 최근점 필터링 대신 텍셀의 이웃을 포함한 텍셀 그룹의 가중 평균을 사용한 선형 필터링 (GL_LINEAR)를 사용해서 에일리어싱 문제를 줄여줌

7) 위 코드에서 texture[2], texture[3], texture[4]로 텍스쳐 바인딩하여 그렸을 때 나타나는 차이점을 그림으로 그려서 자세히 설명하라. (10점)

http://dis.dankook.ac.kr/lectures/cg13/entry/OpenGL-Texture-Mapping-Filtering-Environment-Parameters

GL_REPEAT는 텍스쳐 좌표가 [0, 1]를 넘쳤을 때 텍스쳐를 반복 사용.

GL_MIRRORED_REPEAT는 텍스쳐 좌표가 [0, 1]를 넘쳤을 때 텍스쳐를 반사하면서 반복 사용.

GL_CLAMP_TO_EDGE는 텍스쳐 좌표가 [0, 1]를 넘쳤을 때 텍스쳐를 더 이상 그리지 않음.



8) 위 코드에서 setSquareData() 함수 내에 다음 부분이 아래와 같이 변했을 때, drawTextureQuad(texture[3]); 출력결과를 그려라. (5점)

```
squareVertices.push_back(glm::vec3(-0.5f, -0.5f, 0.0f));
squareVertices.push_back(glm::vec3( 0.5f, -0.5f, 0.0f));
squareVertices.push_back(glm::vec3( 0.5f, 0.5f, 0.0f));
squareVertices.push_back(glm::vec3(-0.5f, 0.5f, 0.0f));
squareTextureCoords.push_back(glm::vec2(-1.0f, -1.0f));
squareTextureCoords.push_back(glm::vec2( 0.0f, -1.0f));
squareTextureCoords.push_back(glm::vec2( 0.0f, 0.0f));
squareTextureCoords.push_back(glm::vec2(-1.0f, 0.0f));
```

9) 위 코드에서 draw() 함수 내에 *이탤릭체 부분*이 어떻게 동작하는지 자세히 설명하라. 이 부분에서 *glDepthFunc(GL_LEQUAL);*는 왜 필요한 지? (10점)

http://dis.dankook.ac.kr/lectures/cg13/entry/OpenGL-Multi-Texturing

블렌딩을 이용한 다중패스 멀티 텍스쳐링 (multi-pass multi-texturing)으로 제작한 라이트 매핑이다. 이 방식은 동일한 다각형 자체를 여러 번 렌더링하는 것으로, texture[5] (배경이미지)를 그리고 난 후, 동일한 다각형 자체에 texture[6] (라이트매핑 이미지)를 합성(blending)하여 그려서 multi-pass multi-texturing하는 것이다.

glDepthFunc(GL_LEQUAL); 함수는 같은 평면에 있는 fragment마다 두 개의 텍스쳐 이미지로 합성할

수 있도록 하기 위하여 사용한다. Z-buffering이 활성화되어 있는 경우 반드시 필요하다.

10) 위 코드에서 draw() 함수 내에 *이탤릭체 부분*에서 *glBlendFunc(GL_ONE, GL_ONE);*는 어떤 블렌딩 함수인가? 만약 배경 이미지만 나타나는 블렌딩 (background only)을 원한다면 어떻게 바꿔야 하는가? (5점)

http://dis.dankook.ac.kr/lectures/cg13/entry/OpenGL-Blending-Filter

glBlendFunc(GL_ONE, GL_ONE) 블렌딩 함수는 덧셈 블렌딩 (add blending)을 해서 배경 이미지(Cd)와 라이트매핑 이미지(Cs)가 합산된 결과로 나타난다.

만약 라이트매핑 이미지는 안 나타나고, 배경 이미지만 나타나는 블렌딩으로 바꾸고자 한다면,

glBlendFunc(GL_ZERO, GL_ONE);

2. 다음은 조명 (lighting)에 관한 문제이다. 아래의 질문에 답하시오. (30점)

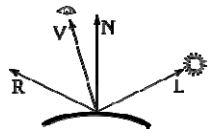
$$I = K_a I_a + \sum_{i=0}^{m-1} f_{att}(d)\left\{K_d I_d (N \bullet L) + K_s I_s (R \bullet V)^n\right\} + E$$

1) 퐁 직접 조명 모델 (Phong Reflection Model)에서 환경반사 (ambient reflection), 난반사 (diffuse reflection), 정반사 (specular reflection)가 무엇인지, 위의 공식에서 어느 부분인지 설명하라. (5점)

K_aI_a : 환경반사 (ambient reflection)는 광원에 직접 노출되지 않는 면에 밝기를 부여함
K_dI_d (N · L): 난반사 (diffuse reflection)는 면의 밝기가 입사각 (광원벡터와 법선 벡터의 사이각)의 코사인에 정비례한다는 램버트 코사인 법칙에 따라서, 물체의 색에 면이 서 있는 방향에 따라 차등적 밝기를 부여함
K_sI_s $(V \cdot R)^n$: 퐁 모델은 Reflection Vector, R을 사용하여, shiny surface에서 정반사 (specular reflection) 경면광(물체의 색이 아니라 광원의 색)을 계산해주며, 시점이 정확히 반대방향일 때 보임

2) 정반사(specular reflection)에 사용하는 반사 벡터가 무엇인지 자세히 설명하라. (5점)

정반사 (specular reflection)에서 퐁 모델의 경우 Ks Is $(R \bullet V)^n$ 을 사용하는데, 광원을 향하는 벡터 L이 표면의 법선 벡터 N에 정반사된 벡터 R을 모든 정점에 대해 계산해야 한다.



3) Per-pixel lighting(일명, Phong shading)이 무엇인지 자세히 설명하라. 이것이 per-vertex lighting과의 차이점은 무언인가? (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 경면광을 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 법

선벡터가 완전히 일치하지 않기 때문) 뭉개지는 현상이 나타남.

4) 다음은 프래그먼트 쉐이더(fragment shader)코드의 일부를 보여주고 있다. 밑줄 친 부분의 코드를 자세히 설명하라. (10점)

```
void main()
{
    vec3 MaterialDiffuseColor = texture2D(gTextureSampler, TexCoordPass).rgb;
    vec3 MaterialAmbientColor = vec3(0.1,0.1,0.1) * MaterialDiffuseColor;
    vec3 MaterialSpecularColor = vec3(0.3,0.3,0.3);
    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 cosAlpha = clamp(dot(V,R), 0, 1);
    Color = MaterialAmbientColor +
        MaterialDiffuseColor * gLightColor * cosTheta / (distance*distance) +
        MaterialSpecularColor * gLightColor * pow(cosAlpha,5) / (distance*distance);
}
```

texture2D는 텍스쳐 이미지 샘플러(gTextureSampler)와 텍스쳐 좌표(TexCoordPass)로부터 텍스쳐 매핑을 수행
distance는 광원 위치 (gLightPosision)와 정점 위치(PositionWorldPass) 간의 거리
cosTheta는 법선벡터 (N)과 광원벡터 (L)간의 내적은 두 벡터간의 입사각의 코사인과 비례
R은 L이 N을 중심으로 반사된 벡터
Color (픽셀당 최종 색)은 ambient + (diffuse +specular)/distance*distance 공식을 이용하여 최종 색을 계산

3. 다음은 moglclass의 SimpleMobile 클래스의 변형인 SimpleObject 클래스이다. 이 클래스가 동작하는 그 출력 결과를 그림으로 그려라. (extra 10점).

```
void SimpleObject::init()
{
  part1 = Cube(4.0f);
  part2 = Cylinder(2.0f, 4.0f, 16);
  part3 = Torus(1.5f, 0.5f, 32, 16);
  part4 = Sphere(2.0f, 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 * bodyTransform;
    p->setUniform("gModel", m1);
    part1.draw();

    glm::mat4 m2 = m1 * partTransform[0];
    p->setUniform("gModel", m2);
    part2.draw();

    glm::mat4 m3 = m2 * partTransform[1] * partTransform[3];
    p->setUniform("gModel", m3);
    part3.draw();

    glm::mat4 m4 = m2 * partTransform[2] * partTransform[3];
    p->setUniform("gModel", m4);
    part4.draw();
}

bool SimpleObject::update(float deltaTime)
{
    angle = angle - 180.0f * (float) (deltaTime) * 0.0001f;
    bodyTransform = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0, 1, 0));
    partTransform[0] = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -4.0f, 0.0f));
    partTransform[1] = glm::translate(glm::mat4(1.0f), glm::vec3(-2.0f, -4.0f, 0.0f));
    partTransform[2] = glm::translate(glm::mat4(1.0f), glm::vec3(2.0f, -4.0f, 0.0f));
    partTransform[3] = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0f, 0.0f, 1.0f));
    return true;
}
```

http://dis.dankook.ac.kr/lectures/cg13/entry/SimpleMobile-Class