Computer Graphics Project 02

제출자 : 20193494 문서형

실행한 결과물은 다음과 같습니다. 움직임, Texture, Lighting 이 구현되어 있습니다.

```
pvmMat = projectMat * viewMat * frontLegMat_1[i];
glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
glDrawArrays(GL_TRIANGLES, 0, NumVertices);

pvmMat = projectMat * viewMat * frontLegMat_2[i];
glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
glDrawArrays(GL_TRIANGLES, 0, NumVertices);

pvmMat = projectMat * viewMat * frontLegMat_3[i];
glUniformMatrix4fv(pvmMatrixID, 1, GL_FALSE, &pvmMat[0][0]);
glDrawArrays(GL_TRIANGLES, 0, NumVertices);
```

project 01 에서 변경한 점입니다. 기존엔 uniform variable 에 vertex 를 넣고 순차적으로 여러 번 그리는 식으로 구현을 했습니다. texture 를 적용하기엔 그렇게 하기에 좀 번거롭다고 판단해서 sphere 예제처럼 Horse 객체를 정의했습니다. 객체 생성시 vertex, texture coordinate, normal vector 의 array 를 생성하도록 구현했습니다.

```
class Horse {

public:
    vector<glm::vec4> verts;
    vector<glm::vec4> normals;
    vector<glm::vec2> texCoords;
```

```
void idle()
{
    static int prevTime = glutGet(GLUT_ELAPSED_TIME);
    int currTime = glutGet(GLUT_ELAPSED_TIME);
    float t = abs(currTime - prevTime);
    cycleTime += abs(currTime - prevTime);
    if (cycleTime > moveCycle)
        cycleTime = 0;
    horse.move(cycleTime);
    glBufferSubData(GL_ARRAY_BUFFER, 0, vertSize, horse.verts.data());
    glBufferSubData(GL_ARRAY_BUFFER, vertSize, normalSize, horse.normals.data());
    glBufferSubData(GL_ARRAY_BUFFER, vertSize + normalSize, texSize, horse.texCoords.data());
    glutPostRedisplay();

    if (isRotate && abs(currTime - prevTime) >= 20)
    {
        float t = abs(currTime - prevTime);
        float speed = 360.0f / 10000.0f;
        modelMat = glm::rotate(modelMat, glm::radians(t*speed), glm::vec3(1.0f, 1.0f, 0.0f));
        prevTime = currTime;
        glutPostRedisplay();
    }
    else if (!isRotate)
    {
        prevTime = currTime;
    }
}
```

기존 프로젝트에서 matrix multiplication을 이용해서 구현한 시간에 따른 다리, 꼬리 등을 움직이는 구현은 `void move(int time)` 함수에 담아두었습니다. 이를 idle함수에서 호출해 buffer의 데이터를 변형한 데이터로 대체, 다시 그리는 식으로 구현했습니다.

normal vector와 vertex coordinate의 설정 방법은 다음과 같습니다.

```
void Horse::quad(int a, int b, int c, int d) {

    //add normal vector setting
    glm::vec4 vec_ab = vertices[b] - vertices[a];
    glm::vec4 vec_ac = vertices[c] - vertices[a];
    glm::vec3 normal_vec3 = glm::cross(glm::vec3(vec_ab), glm::vec3(vec_ac));
    glm::vec4 normal_vec4 = glm::vec4(normal_vec3, 0.0f);
    normal_vec4 = glm::normalize(normal_vec4);

    baseModel[Index] = vertices[a]; baseNormal[Index] = normal_vec4; baseTexCoord[Index][0] = 0.0f; baseTexCoord[Index][1] = 0.0f; Index++;
    baseModel[Index] = vertices[b]; baseNormal[Index] = normal_vec4; baseTexCoord[Index][0] = 1.0f; baseTexCoord[Index][1] = 0.0f; Index++;
    baseModel[Index] = vertices[c]; baseNormal[Index] = normal_vec4; baseTexCoord[Index][0] = 1.0f; baseTexCoord[Index][1] = 1.0f; Index++;
    baseModel[Index] = vertices[a]; baseNormal[Index] = normal_vec4; baseTexCoord[Index][0] = 0.0f; baseTexCoord[Index][1] = 0.0f; Index++;
    baseModel[Index] = vertices[c]; baseNormal[Index] = normal_vec4; baseTexCoord[Index][0] = 1.0f; baseTexCoord[Index][1] = 1.0f; Index++;
    baseModel[Index] = vertices[d]; baseNormal[Index] = normal_vec4; baseTexCoord[Index][0] = 0.0f; baseTexCoord[Index][1] = 1.0f; Index++;
}
```

기본 cube의 normal vector, texture coordinate 설정을 quad function에서 해주었습니다.
기본 cube를 몸의 각 부분으로 변형시켜주는 matrix를 곱해 vertex, normal vector에 넣어주는 식으로 구현했습니다.

```
void Horse::pushTransformedModel(glm::mat4 transMatrix) {

    for (int i = 0; i < 36; i++) {
        verts.push_back(transMatrix * baseModel[i]);
        normals.push_back(transMatrix * baseNormal[i]);
        texCoords.push_back(baseTexCoord[i]);
    }
}
```

texture의 로드의 경우 제시된 코드를 변형하지 않았고, 인터넷에서 찾은 texture 이미지를 동일한 형식인 1024 x 1024 resolution, 24 bit depth, bmp 형식으로 변형해서 넣어주었습니다.

lighting 상수, light 위치, viewer 위치의 사소한 변경도 있었습니다.