CSED451 Assignment #4

3D DRAWING REPORT

Team: GPS

CSE 20202405 문민재 csmmj4594

CSE 20202728 김진수 fusion4268

**Overview of Program**

본 과제에서는 assn3에서 구현한 vertex shader와 fragment shader에 lighting, shading, 그리고 texture mapping 기능을 추가적으로 구현하였다. Lighting은 directional light, point light 2가지 광원이 존재하며, 키보드 입력을 통해 2 가지 광원이 모두 적용된 모드, directional light만 적용된 모드, point light만 적용된 모드로 전환할 수 있다. Shading은 hidden surface removal을 적용한 Gouraud shading과 Phong shading을 구현하였다. Texture mapping은 Gouraud shading의 경우 diffuse mapping을 Phong shading의 경우 diffuse mapping과 normal mapping을 모두 사용할 수 있도록 구현하였다.

**Programming Environment**

Visual Studio 2019

OpenGL – 4.6

GLSL – 4.60

Freeglut-MSVC-3.0.0

Glew-2.1.0

Glm-0.9.9.7

**Design and Implementation**

1. 자료구조 설계
2. Object (object.h)

맵 상의 오브젝트를 정의하는 클래스이다.  
멤버 변수로 type, color, x, y, z를 가지고 있다. 이들은 각각 오브젝트 타입(0일 경우 wall, 1일 경우 character), 오브젝트의 색깔, 오브젝트를 그릴 때 필요한 x, y, z좌표에 해당한다.

1. Character: public object (character.h)

Object를 상속받아 맵 상의 캐릭터 오브젝트(플레이어, 도둑)를 정의하는 클래스이다.  
character 오브젝트들은 type이 1에 해당하며, x, y, z는 torso의 중심 좌표를 나타낸다. 그리고 원의 반지름을 나타내기 위한 변수인 rad와 벽과의 충돌 여부를 나타내는 변수인 isCollided가 있다. 그 외에 인간형 캐릭터 구현에 필요한 여러 멤버변수들을 가지고 있다.

1. treeNode (treeNode.h)

인간형 캐릭터의 hierarchy를 표현하기 위해 필요한 Left Child Right Sibling(LCRS) tree를 구현하는 클래스이다.

현재 modelView matrix의 오른쪽에 곱해질 기본적으로 matrix인 mtx,  
기본 mtx transform에 추가적으로 행해질 transform을 나타내는 additionalTransform,  
이 node에 해당하는 신체 부위를 그려주기 위한 함수 포인터인 draw,  
그리고 sibling과 child 정보를 가지고 있다.

1. pose (pose.h)

플레이어와 도둑의 포즈를 정의한 클래스이다.  
캐릭터의 포즈에 따른 변화는 상체에서만 이루어지므로, 왼팔과 오른팔 각각 2가지씩 총 4가지 각도와 포즈에 따른 색을 변수로 갖고있다.

1. camera(camera.h)

이번 과제에서는 3가지 시점을 구현했는데, 각 시점에 해당하는 카메라 정보(camera position, reference point, up vector))을 저장하는 클래스이다.

1. Floor(Floor.h)

바닥 역할을 하는 격자를 그리기 위한 클래스이다.  
floor vertices와 floor indices를 변수로 가지며, GL\_LINES를 이용하여 여러 선들로 격자를 형성한다. 격자의 색은 검정색으로 한다.

1. Cylinder(Cylinder.h)

Wall과 character의 limb, torso를 그리기 위해 원기둥을 정의한 클래스이다. 생성자는 baseRadius, topRadius, height, sectorCount, stackCount를 parameter로 전달받는다. baseRadius는 원기둥의 밑면의 반지름을 의미하며, topRadius는 원기둥의 윗면의 반지름을 의미한다. baseRadius와 topRadius가 다를 때, 윈기둥이 아닌 밑면과 윗면의 크기가 다른 기둥을 생성할 수 있다. heigh는 원기둥의 높이를 의미한다. sectorCount와 stackCount는 각각 원기둥을 그릴 때 사용되는 sector와 stack의 수를 의미한다. 여기서 sector는 경도를 기준으로 원기둥을 몇 부분으로 분할할지를 의미하며, stack은 위도를 기준으로 원기둥 몇 부분으로 분할할지를 의미한다. 원기둥의 vertices를 GL\_TRIANGLES를 이용해 그리기 위한 indices를 변수로 가지며, 5가지 색을 구현하기 위하여 색에 따른 vertices\_(color) 변수를 갖는다.

1. Sphere(Sphere.h)

character의 head와 joint를 그리기 위해 구를 정의한 클래스이다. 생성자는 radius, sectorCount, stackCount를 parameter로 전달받는다. radius는 구의 반지름을 의미하며, sectorCount와 stackCount는 각각 구를 그릴 때 사용되는 sector와 stack의 수를 의미한다. sector, stack 그리고 원의 색과 관련된 설명은 원기둥과 같다.

1. Wall: public object (Wall.h)

Object를 상속받아 맵 상의 벽을 정의하는 클래스이다.  
Wall 오브젝트들은 공통적으로 type이 0이며, x, y, z는 벽의 위치를 의미한다. 상황에 따라 총 5가지 종류의 벽을 그린다. 5가지 벽의 종류로는 1) 원기둥, 2) 윗면이 더 큰 원기둥, 3) 밑면이 더 큰 원기둥, 4) 원뿔, 그리고 뒤집어진 원뿔이 있다. 5가지 종류의 벽은 모두 cylinder의 baseRadius와 topRadius를 조절해 그릴 수 있다.

1. matrixStack.h

matrixStack은 OpenGL fixed pipeline이 가진 glPushMatrix()와 glPopMatrix()를 대체하기 위해 구현된 헤더이다. modelViewStack과 projectionStack으로 기존의 fixed pipeline과 같게 stack을 이용해 구현하였으며, matrix의 계산을 직접 수행할 수 있도록 mtxView와 mtxProj를 변수로 가진다. vertex shader에서 matrixStack.h의 projection matrix와 modelView matrix를 활용하여 실제로 그려질 위치인 gl\_position을 계산한다.

1. shaderinfo(shaderinfo.h)

shader를 사용하기 위해 사용되는 함수들이 모두 선언 되어있다. shader를 initialize하고 각 변수들을 연결하여 값을 사용할 수 있게 하였다. texture의 경우 shaderinfo에서 texture 폴더에 있는 texture와 normal map을 가져오고, fragment shader의 texture와 normalMap에 연결해 사용할 수 있도록 구현하였다.

1. light.h

light는 lighting을 구현하기 위해 필요한 변수들을 선언하였다. ambient, diffuse, specular 3가지 모두 선언하였으며 diffuse와 specular는 point light과 directional light으로 구분하여 선언하였다. light의 위치나 shininess 또한 light.h에 선언하였으며, 위으 변수들을 값을 조절함으로서 빛의 위치나 특성을 변화시킬 수 있다.

1. stb\_image.h

텍스처 및 normal map을 로드 하기 위한 헤더이다. stbi\_load() 함수로 이미지 파일을 로드 할 수 있다.

1. 기본 기능 구현(이전 ASSN 관련)
2. 벽의 생성 및 이동

Assn3\_1과 동일하게 구현되었다.

1. 충돌 판정

Assn3\_1과 동일하게 구현되었다.

1. 플레이어 캐릭터의 조작

Assn3\_1과 동일하게 구현되었다.

1. 치트 기능

Assn3\_1과 동일하게 구현되었다.

1. Pass/Fail 시의 action

Assn3\_1과 동일하게 구현되었다.

1. Jump 구현

Assn3\_1과 동일하게 구현되었다.

1. 3D 시점 구현

Assn3\_1과 동일하게 구현되었다.

1. 시점 변환 및 줌 인/아웃

Assn3\_1과 동일하게 구현되었다.

1. Lighting & Shading

현재 프로그램에 구현한 광원은 directional light과 point light이 존재한다. Gouraud shading은 vertex shader에서 lighting과 관련된 연산을 수행하고 Phong shading은 fragment shader에서 lighting과 관련된 연산을 수행한다. 그래픽 환경에서 존재할 수 있는 light는 ambient light, diffuse light, specular light가 있으며 본 프로그램에서는 3가지 light를 모두 구현하였다.

Gouraud shading과 Phong shading을 구현하였다. Gouraud shading은 대부분의 연산을 vertex shader에서 수행하며, Phong shading은 대부분의 연산을 fragment shader에서 수행한다. VBO로 사용되는 변수들은 shader의 input으로 선언되었으며, lighting이나 shading과 관련된 변수들은 uniform 형태로 선언되었다.

1. Texture

캐릭터, 벽, 바닥에 다한 diffuse 텍스처 매핑을 구현하였다. 프로젝트 내의 texture 폴더에 각각의 texture를 저장하며, 텍스처 파일을 로드 하기 위하여 stb\_image.h 코드를 프로젝트에 추가하였다. stbi\_load 함수를 이용하여 캐릭터, 벽, 바닥의 텍스쳐와 normal map을 로드 한다. glGenTextures 함수로 각각의 texture를 생성하고 glTexImage2D로 해당 이미지를 shader에서 사용할 수 잇는 텍스처 형태로 변환한다. glActiveTexture 함수를 사용하여 texture와 normal map에 따라 GL\_TEXTURE0와 GL\_TEXTURE1으로 구분하여 저장하였다. 각각의 image 파일에 대한 텍스처로의 변환이 끝나면 stbi\_image\_free로 할당된 이미지를 해제한다. 해당 texture를 사용하고자 하면 glBindTexture()에 해당 texture의 location을 전달해 사용할 수 있다.  
Shader에서 texture를 사용하기 위하여 texture가 mapping될 좌표를 shader로 전달하였으며, sampler2D 형태로 texture와 normalMap 변수를 연결하였다. Gouraud shading과 Phong shading 모두 diffuse mapping의 경우, gl\_FragColor = fragColor \* texture2D(texture,texCoord); 식으로 color에 생성된 texture를 multiplication해 텍스쳐가 적용된 fragColor 값을 얻는다.

Phongshading의 normal mapping은 다음 식을 이용하여 계산된다.

vec3 normal = texture2D(normalMap, texCoord).rgb;  
 normal = normalize(normal \*2.0 - 1.0);  
 N = N \* normal;

기존의 N은 그려야 할 도형의 normal vector이며 normalMap은 해당 텍스처에 대한 normalMap이 된다.

**How to Run**

프로젝트 폴더의 \bin\x64 폴더로 이동하여 graphics\_assn4.exe를 실행시킨다.

키보드 입력 정리

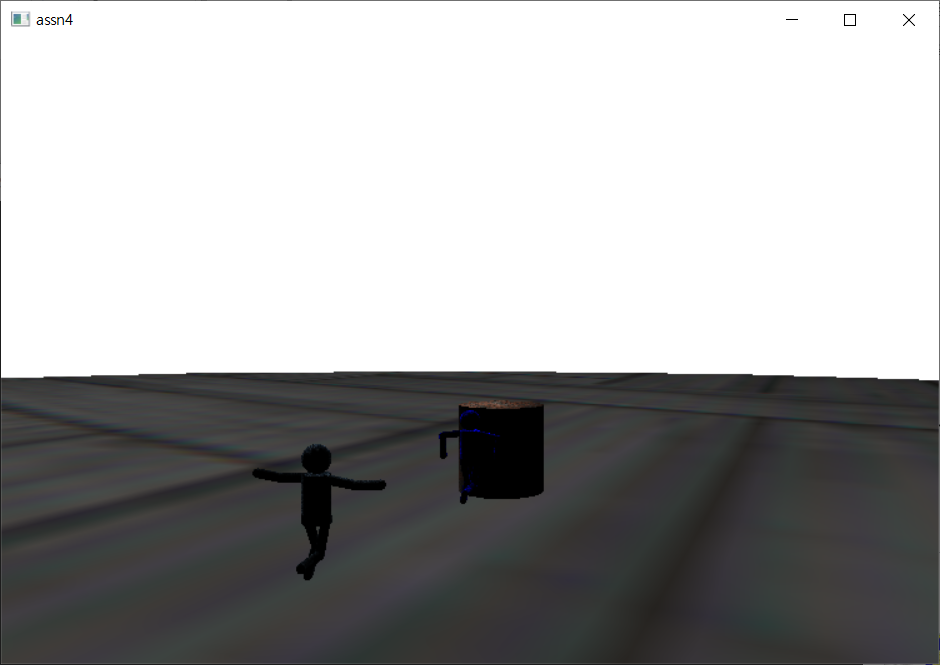
1. 방향키: 캐릭터의 색 및 포즈 변경
2. Space: 캐릭터 점프
3. C: All Pass
4. F: All Fail
5. Q: Gouraud Shading / Phong Shading
6. W: Directional Light + Point Light / Point Light / Directional Light
7. E: Texture 적용 / 미적용
8. R: Phong shading일 때, Diffuse Map / Normal Map
9. Keypad 1~8

* 다양한 카메라 시점을 구현하였다. 각 시점이 바라보는 방향이나 카메라의 위치는 다음과 같다.
* 1: 1인칭 시점
* 2: +x axis
* 3: 3인칭 시점
* 4: 3인칭 시점 2
* 5: - x axis
* 6: -y axis
* 7: Directional light position
* 8: -z axis
* 9: +z axis

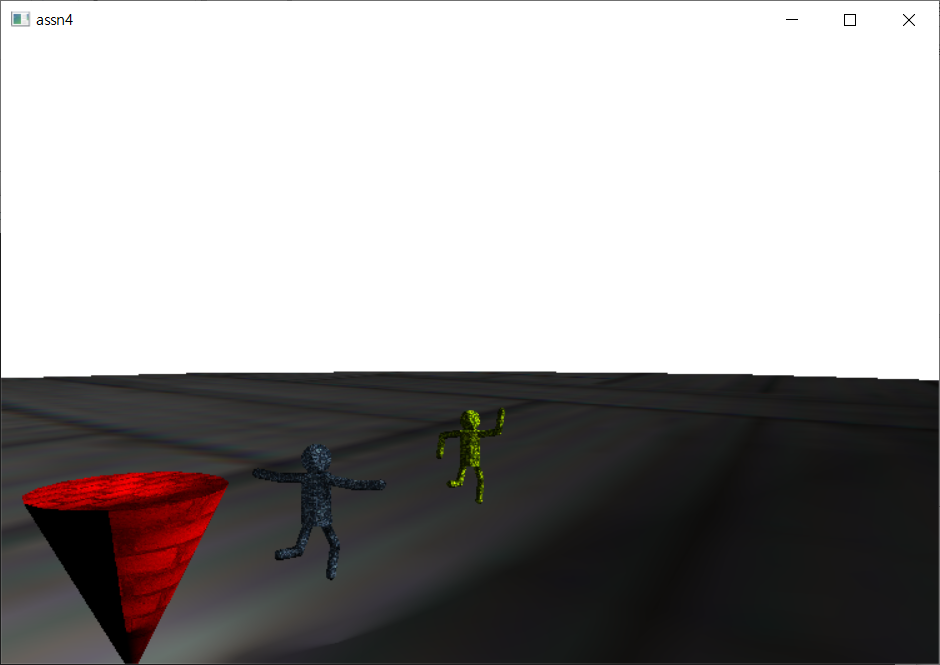
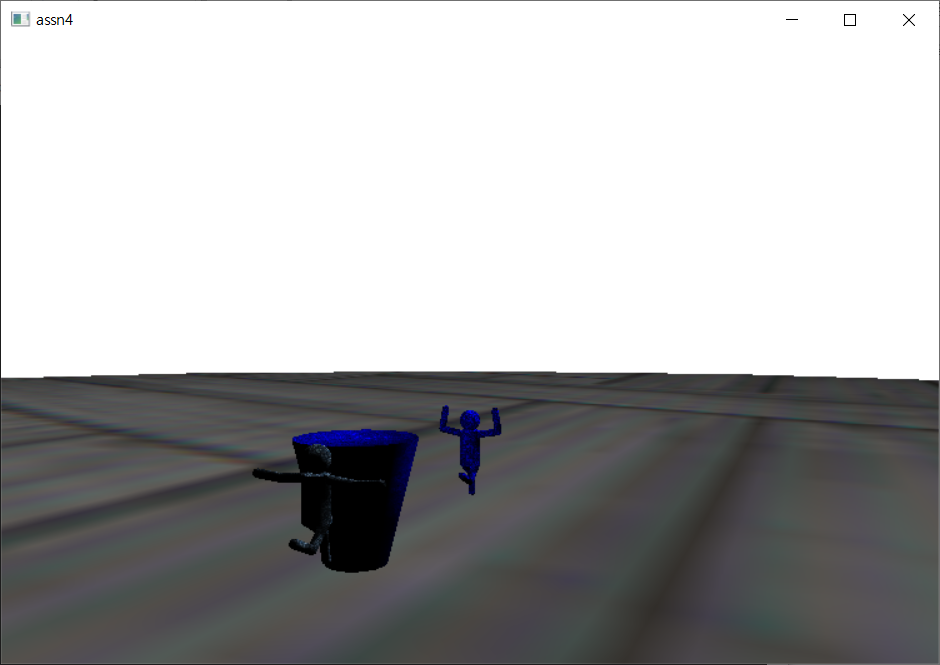
**Example**

기존 Assn3와 중복되는 부분은 제외하였다.

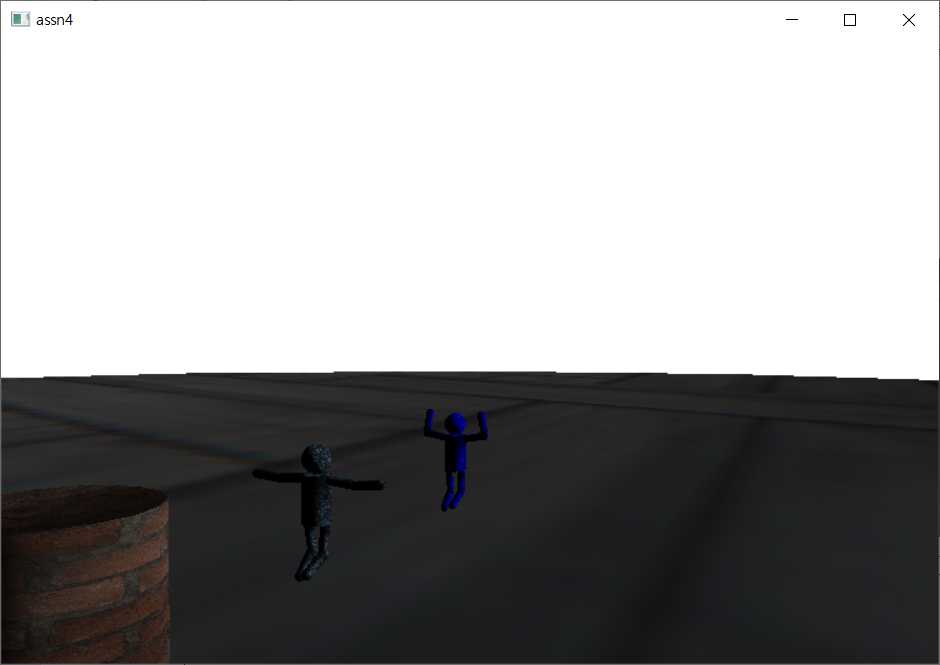
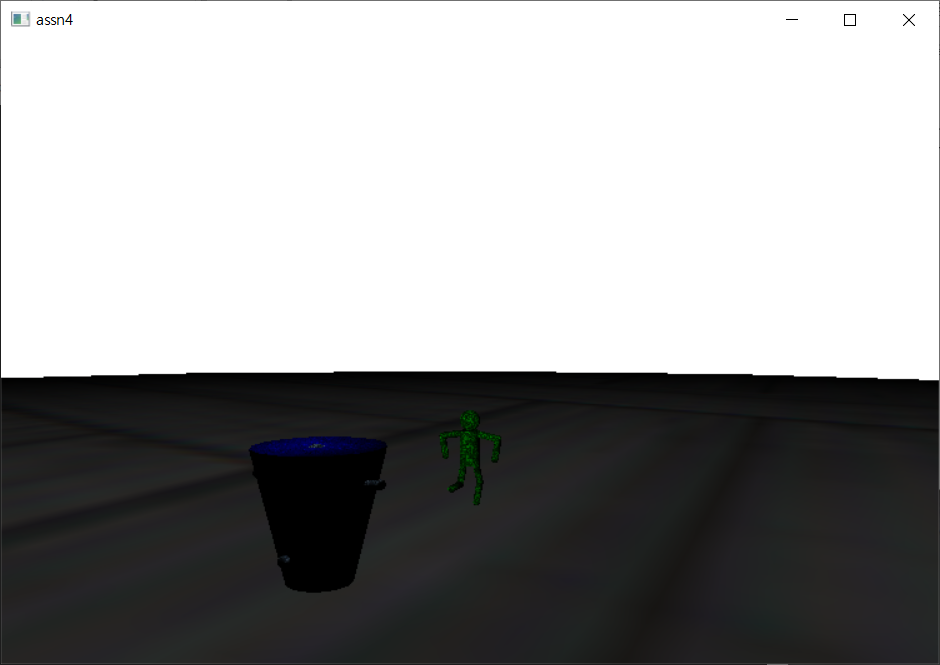
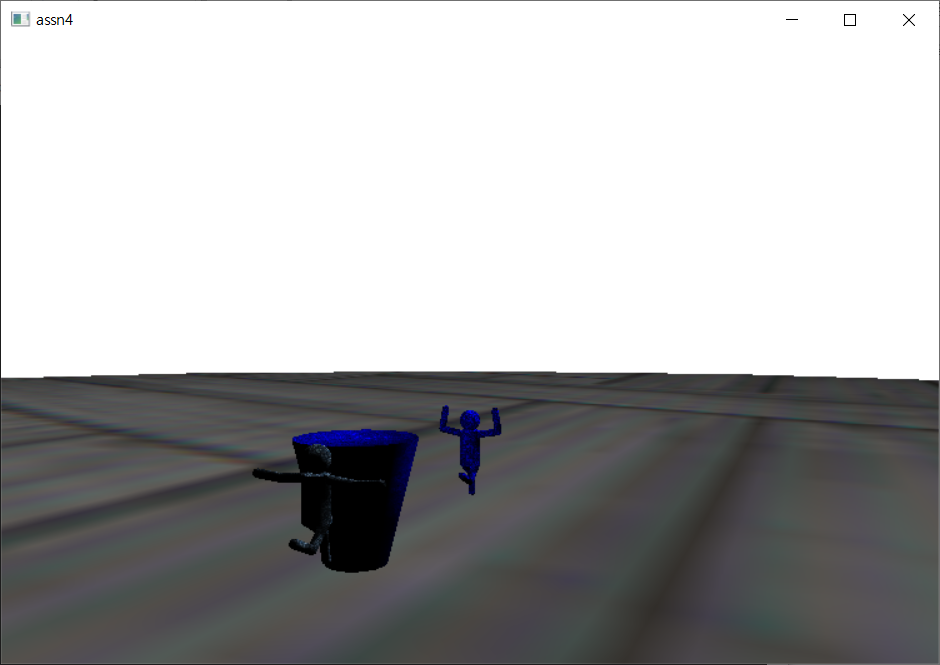
프로그램 실행 시 화면으로, 카메라가 플레이어의 오른쪽 뒤에 있는 3인칭 뷰로 시작한다.  
Shading: Gouraud Shading  
Lighting: Directional Light + Point Light  
Texture Mode: Diffuse Map



아래의 화면은 각각 Gouraud shading과 Phong shading을 적용했을 때의 화면이다. 키보드의 q 입력으로 전환할 수 있다.

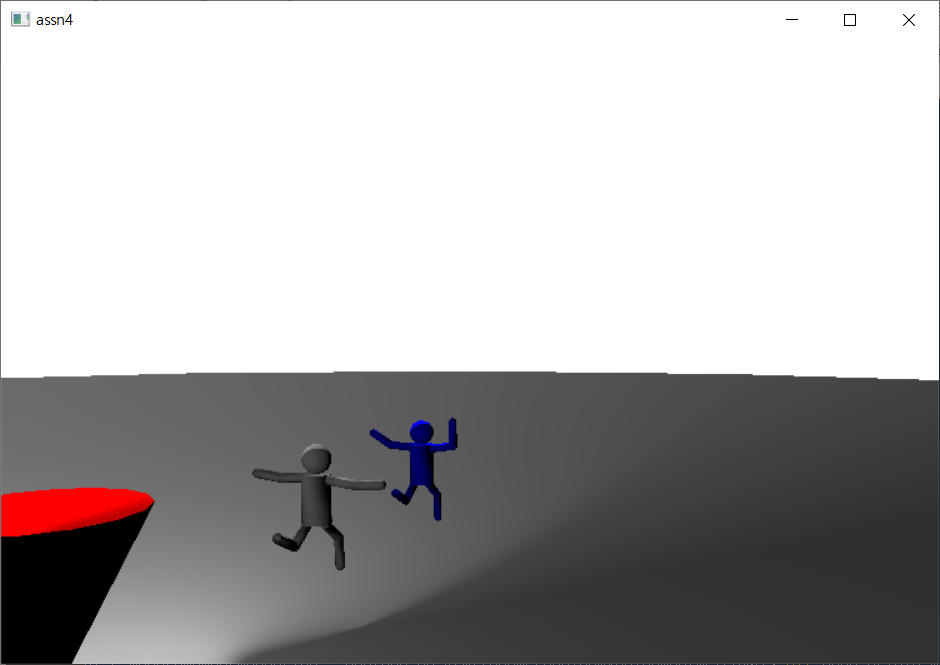
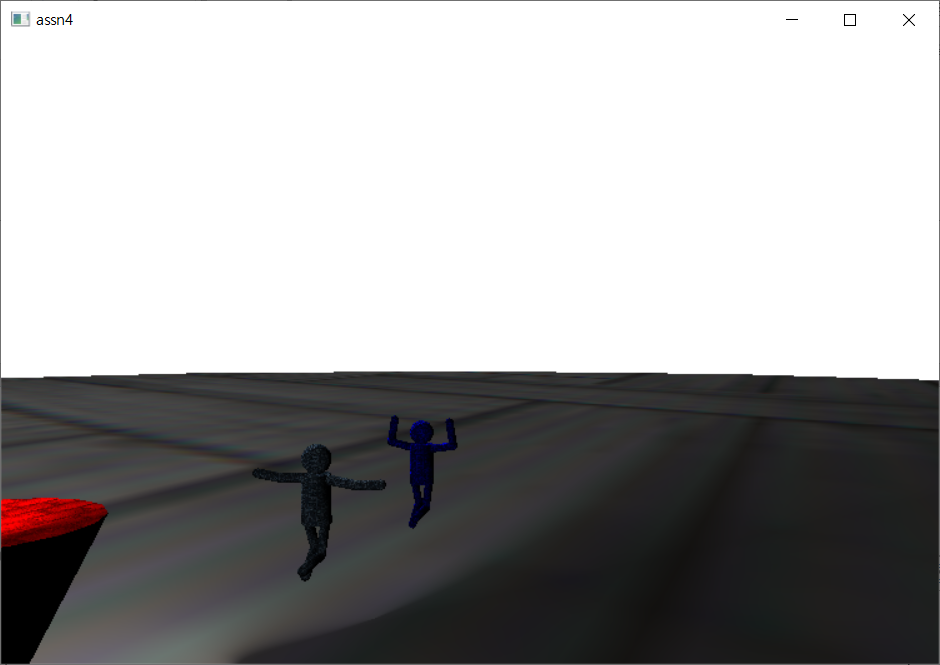


키보드의 w 입력으로 Lighting을 Directional Light + Point Light, Point Light, Directional Light로 변환할 수 있다.

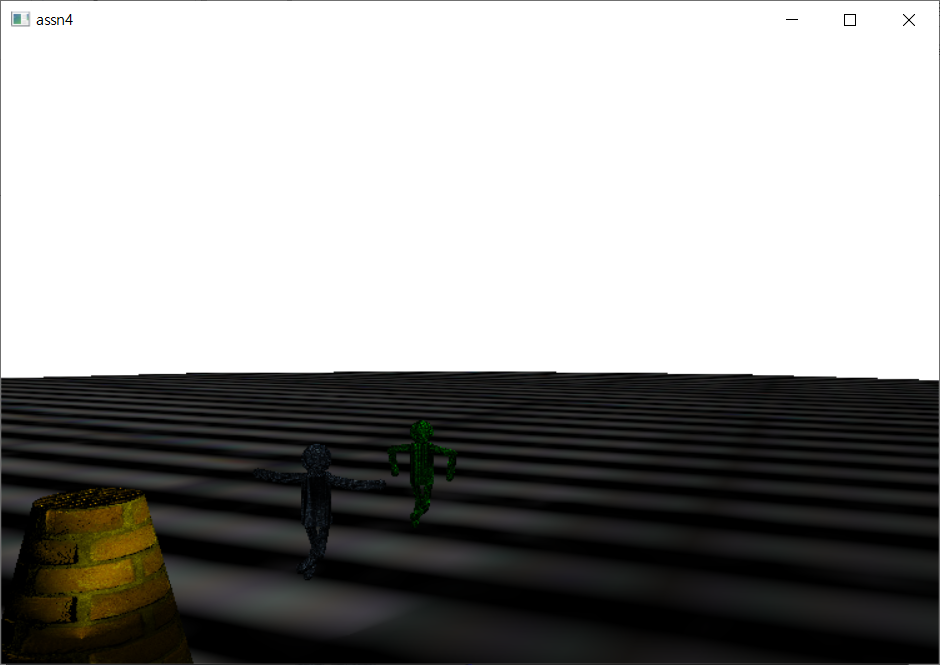
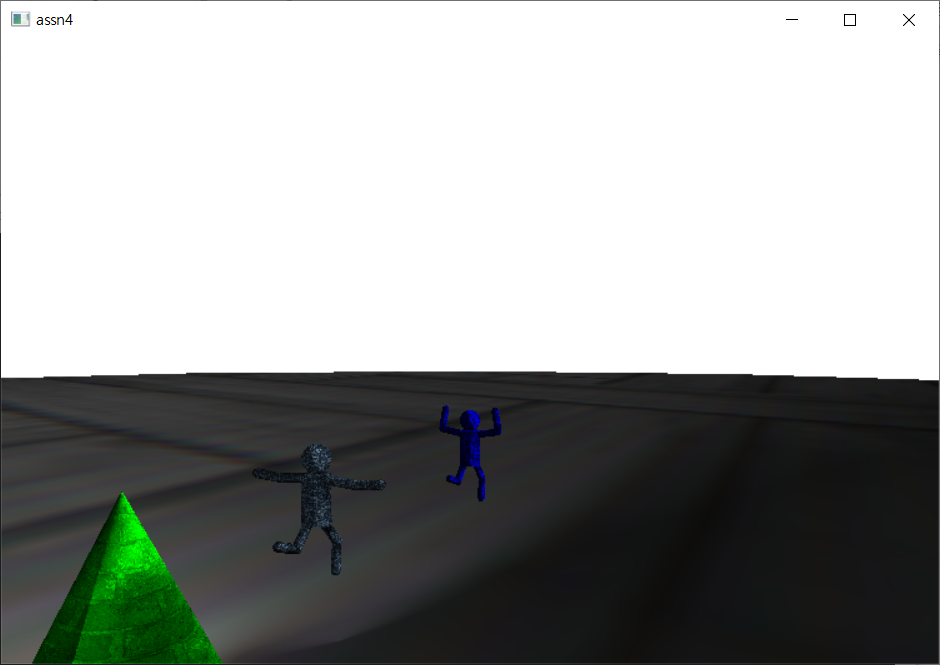


각각 Directional Light + Point Light, Point Light, Directional Light이다. Point Light만 적용되어 있는 경우 벽 위에 존재하는 광원을 확인할 수 있다.

키보드의 e 입력으로 텍스쳐가 적용된 화면과 적용되지 않은 화면을 전환할 수 있다. 각각 Phong shading에 Directional Light와 Point Light이 모두 적용되었을 때 texture를 적용한 화면과 적용하지 않은 화면이다.



키보드의 r 입력으로 Phong shading일 때 diffuse mapping이 적용된 경우와 normal mapping이 적용된 경우를 전환할 수 있다. normal mapping을 적용하였을 때 벽의 texture가 보다 입체적으로 보이는 것을 확인할 수 있다.



**Discussion**

1. Lighting & Shading

Lighting과 Shading 기능을 추가하면서 shaderinfo.h와 shaderinfo.cpp 코드를 생성하였고 shader와 관련된 함수들을 위 코드에 선언 및 실행하였다. shader의 모든 변수들을 linking하는 과정을 수행하였으며 위 과정에서 하나의 변수라도 linking이 제대로 수행되지 않았을 때 구현하고자 하는 방향으로부터 크게 벗어난다는 것을 확인할 수 있었다.

Light.h에 선언한 lighting에 관련된 변수들과 관련해 값을 할당하는 데 어려움을 겪었다. 여러 변수에 할당된 값들을 변화해 가며 어떤 차이가 발생하는지 확인하였고, 최적의 값을 찾는 과정에서 여러 시행착오를 겪었다. Shininess에 적절한 값이 할당되지 않았을 때 전체적으로 잘못된 lighting이 구현되었으며, 이 문제를 할당된 값이 아닌 구조에서 문제 원인을 찾다가 많은 시간이 소요되었다. 그 과정에서 lighting과 관련된 각각의 값들이 어떤 역할을 하는지 확인할 수 있었다.

1. Texture

Texture를 구현하면서 시간이 가장 많이 소요되었던 부분은 여러 개의 texture를 가져오는 부분이었다. 각각의 texture를 다른 location에 저장해야 하며, 어떠한 객체를 draw하느냐에 따라 필요한 location을 bind해야 알맞은 texture를 mapping할 수 있었다. 또한 texture 뿐만 아닌 normal map도 구현하기 위하여 glActivateTexture() 함수를 사용하였으며, GL\_TEXTURE0에 texture와 관련된 텍스처를, GL\_TEXTURE1에 normal map과 관련된 텍스처를 저장하였다. shader에 texture 관련 연산을 추가하는 과정은 이미지로부터 텍스처를 생성하고 알맞게 할당하는 과정에 비해 쉽게 구현할 수 있었다. 기존의 fragColor에 적용해야 할 texture vector를 곱했을 때 텍스처가 적용되는 것을 확인할 수 있었다.

이미지 파일을 로드 해 texture로 변환할 수 있는 여러 가지 방법을 확인하였으며 본 프로그램에서는 stb\_image.h 코드를 이용하여 그 과정을 수행하였다. normal map image는 외부의 normal map으로 변환해주는 링크를 사용하여 해당 texture의 normal map을 얻을 수 있었다.

**Conclusion**

1. Lighting & Shading

Lighting mode와 Shading mode에 따른 shader를 구현하면서 키보드 입력을 shader에 반영하는 방법을 알 수 있었다. 기존 assn3에서 구현하였던 VBO에 shading을 구현하기 위해 필요한 VBO들을 추가하고 그 VBO를 활용하여 lighting과 shading을 구현하는 과정에서 VBO의 구조에 대해 보다 자세히 이해할 수 있었다. Shader에서 shading에 필요한 계산과 정을 이해할 수 있었으며 각 lighting의 개념들에 대하여 이해할 수 있었다. 또한, Gouraud shading과 Phong shading의 차이를 이해하고 구현하면서 shader에서 구조적으로 어떤 차이를 갖고 있는지 알게 되었다.

Ambient, diffuse, specular reflection의 차이를 알 수 있었으며, light.h에서 각각의 값들을 조절하며 그 개념이 실제로 어떻게 적용되는지 확인할 수 있었다.

1. Texture

이미지를 로드하고 텍스처를 생성하는 과정에서 openGL에서 텍스처가 생성되는 과정을 알 수 있었다. glActiveTexture()를 통해 shader에 여러 texture를 동시에 사용하는 방법을 알게 되었으며, glBindTexture()로 사용하고자 하는 texture를 텍스처를 가져올 수 있었다. Normal map을 생성하고 해당 normal map을 fragment shader에 적용하는 방법을 알 수 있었다. Normal map을 fragment shader에 적용하기 위해서는 normal vector, tangent vector, bitangent vector가 필요하다는 것을 알 수 있었다. 다만, phong shading의 연산에 어떻게 적용되는지 확실히 이해하지 못해 normal mapping이 부족하게 구현되게 된 것에 대해 아쉬움이 남았다.

**Direction of Improvement**

1. Normal map

Phong shading에서의 normal map이 완전히 구현되지 않은 것으로 보여진다. tangent vector, bitangent vector, normal vector 3가지 종류를 사용한 방식이 아닌 normal vector와 normal map 이미지만으로 구현하고자 하였으나, 기존의 normal vector와 normal map을 연결하는 과정에서 오류가 있는 것으로 보여진다. 3가지 종류의 vector를 모두 사용하는 방식을 적용한다면 물체의 입체감을 더욱 살릴 수 있을 것이다.

1. Texture quality

floor의 경우 texture의 해상도로 인해 깨져 보이는 현상이 발생하며 character의 경우 해당 모델에 알맞게 적용할 수 있는 texture가 존재하지 않아 규칙성을 가진 texture를 적용하였다. 위의 문제들을 해결한다면 심미적으로 보다 나은 결과물을 얻을 수 있을 것이다.

1. Brightness

lighting을 구현하는 과정에서 초기 변수를 잘못 설정하여 전체적인 밝기가 어둡게 나타난다. 이 문제를 해결한다면 물체를 보다 선명하게 식별할 수 있을 것으로 보여진다.

**참고문헌**

<https://alleysark.tistory.com/260>: 이번 과제의 구현에서, 파일로 된 vertex/fragment shader를 string으로 읽어오는 코드(readShaderSource 함수)와,  
program에 vertex/fragment shader를 attach 완료한 후 program을 점검하여 문제가 있을 경우 로그를 띄워주는 CheckProgram 함수를 참고하였다.

<https://heinleinsgame.tistory.com/7>: GLFW에서 shader를 적용시켜 삼각형 하나를 출력하는 기본적인 코드로, 이번 과제에서 shader를 사용하기 위한 setup(VAO, VBO, program setting 등)을 참고하였다.

<https://github.com/g-truc/glm/blob/master/manual.md#section5>: glm으로 기존 openGL의 fixed pipeline 상에서의 matrix 계산을 대체하는 방법을 참고하였다.

<http://www.songho.ca/opengl/gl_cylinder.html> - Cylinder

<http://www.songho.ca/opengl/gl_sphere.html> - Sphere

Cylinder와 Sphere를 구현하는 부분의 위의 링크를 참고하여 구현하였다. 위의 링크에서는 현재 프로그램에서 구현되어 있는 vertices와 indices 뿐만 아닌 normal vectices와 texture까지 구현한다. 과제 프로그램은 이를 간소화하여 vertices와 indices를 구하는 부분만 프로그램에 추가하였다.

<https://heinleinsgame.tistory.com/19>: Directional light 참고

<https://heinleinsgame.tistory.com/15?category=757483>: Phong shading 참고

<https://heinleinsgame.tistory.com/9?category=757483>: Texture & Normal map 로드 참고

<https://learnopengl.com/Advanced-Lighting/Normal-Mapping>: Normal mapping 참고

<https://www.smart-page.net/smartnormal/>: Normal map 이미지 생성