|  |  |  |
| --- | --- | --- |
| 이름 | 백성빈 | 손주은 |
| 학번 | 20170680 | 20170702 |
| 학과 | 컴퓨터공학과 | 컴퓨터공학과 |
| ID | sungbin0515 | jeson |

CSED 451 Computer Graphics—Assn3

**Assn 3: 피카츄 배구**

ASSN 1과 2에서 만들었던 2D 피카츄 배구 게임을 3D그래픽으로 구현한다.

# 개발 환경:

**IDE:** Visual Studio 2017 community

**Compile mode (VS 2017):** Release 32bit(x86)

**Dependency**: OpenGL 3.3 / GLFW / GLSL3.30.6(#version 330)

# **프로그램 설계**

1. **Vertex Shader와 Fragment Shader**

이번 과제에서는 앞선 과제들과 달리 Opengl의 내장함수들을 사용하여 Vertex를 그릴 수 없다. 따라서 우리는 GLSL(openGL Shading Language) 언어로 작성한 Programmable Shader, 그 중에서도 Vertex Shader와 Fragment Shader를 작성했다. Application에서 Model, View, Projection Matrix(Coordinate)를 다 곱한 하나의 Matrix를 Vertex Shader에게 전달하면 Shader는 이를 바탕으로 우리가 원하는 모델을 그려낸다. GLSL Shader는 Application에 포함되는 것이 아니라, 별도의 문서로 작성된 후에 App에서 컴파일이 이루어져야 한다. 이를 위해 Compile한 프로그램을 연동하는 코드를 외부 코드를 인용했으며, 그 출처는 코드와 보고서 하단에 병기되어 있다.

Vertex Shader에서는 하단의 Obj File Loader를 통해 Load한 3d mesh의 vertex 정보를 제공받고, 여기에 Application에서 제공한 Matrix를 곱한 값을 gl\_position 내장 변수에 저장한다. 이 정보는 Fragment Shader로 넘겨지고, 이는 역시 Application이 지정한 색을 입혀 Fragment의 색을 지정한다. 그리고 GLSL 안에서 이 fragment들이 pixel로 변환된다.

1. **GameManager와 RenderManager**

Object Rendering을 수동으로 구현해야 하지만, Game Logic과 Rendering 사이에는 별다른 주종관계가 없으므로, 두 기능을 분리하여 별도의 관리 Object를 생성했다. GameManager Class는 공을 생성하는 Logic이나 키보드를 누르면 캐릭터의 위치와 방향을 변경하는 Logic, AI의 위치를 공의 위치에 반응하도록 하는 Logic 등 Graphic과 무관한 Game Logic을 총괄하며, RenderManager는 후술할 ObjFileLoader를 통해 VAO(Vertex Array Object)에 연동된 GPU 내 버퍼안에 저장된 vertex 값을 활용해서 사용자가 원하는 모델을 원하는 위치나 방향에 그리는 역할을 한다.

1. **Loading Obj File**

.Obj 파일은 3D Object의 vertex 정보, UV mapping 정보와 normal vector 정보를 날 것 그대로 담고 있는 primitive한 형식 중 하나다. 즉 MAYA, Blender 등 특정 프로그램에 종속되어 있지 않다. 이는 .obj 파일을 읽는 방법이 공개되어 있으며, 또 그 방법이 쉽다는 것을 의미한다. 이번 Assignment에 필요한 3D Model 역시 라이선스 문제가 없는(무료배포) .obj 파일을 사용해서 간단하게 이브이 및 공의 3D Model을 불러올 수 있다.

3D model의 face 정보를 저장하는 방법은 여러가지가 있는데, obj 파일은 대부분 vertex pointer 방식을 사용한다. 이 방법은 explicit vertex 방식보다 메모리를 더 절약하지만, 구현하기 약간 더 어렵다는 단점이 있다. 따라서 우리는 구현을 쉽게 하기 위해 vertex pointer 방식으로 저장된 obj 파일을 해석해서 각 face triangle의 vertex 3개를 glm::vec3 으로 변환하여 하나의 list로 저장했다.

Obj file을 해석하여 변환하는 코드는 하단의 opengl-tutorial 사이트의 코드를 인용했으며, 코드상의 주석과 보고서 최하단에 출처를 병기하였다.

1. **Camera Setting**

이번 Assign에서는 총 3가지 모드의 카메라를 구현해야 한다. 첫번째는 캐릭터의 시야와 완전히 동일한 위치에서, 두번째는 캐릭터의 뒤통수에서 캐릭터가 정면을 바라보는 시점, 마지막은 플레이어가 돌아다닐 수 있는 영역의 상단에서 자유롭게 움직일 수 있는 카메라(시점은 원점 고정)가 있다. Camera도 결국 vertex에 transform을 적용하는 것과 같으며, 여기에 필요한 transform matrix는 glm library에서 제공하는 glm::mat4 lookAt(vec3 pos, vec3 dir, vec3 up) 함수를 활용해서 계산할 수 있다.

1. **Hidden Line Removal**

3D Model을 렌더링 할 때 다른 오브젝트나 Face에 가려져 그려지지 않는 Line까지 그리는 것은 큰 메모리 낭비다. 이를 방지하기 위해 다양한 Hidden Line Removal Algorithm이 연구 되었으며, 우리는 그 중 가장 간단하며 하드웨어 상으로 구현되어 있는 Z-buffer Algorithm을 사용했다.

Z-buffer Algorithm은 각 Fragment를 그릴 때 각 Pixel별로 이미 그려져 있는 Fragment의 Z축 깊이를 측정하고, 기존에 Z축 깊이 값이(depth value) 있다면, 새로 그릴 pixel의 z축 깊이가 더 작을 때(모니터에 더 가까울 때)만 그리고 아니라면 버리는 알고리즘을 의미한다. OpenGL에는 Z-depth를 설정할 수 있는 기능이 있으며, 이를 활용해서 같은 3D Model을 첫번째는 배경색과 동일한 GL\_FILL 모드(면을 채우는 모드)로 그리고, 두번째는 배경색과 크게 차이나는 GL\_LINE(면을 채우지 않고 선만 그리는 모드) 모드로 그림으로써 Object의 Face에 의해 가려지는 Line을 그리지 않는다.

키보드의 ‘H’키를 입력함으로써 hidden line removal mode와 wireframe mode 사이에서 전환할 수 있다.

1. **충돌 판정 및 점수 계산**

3D 물체 간의 충돌을 판정하는 방법은 여러가지가 있다. 그 중에서 가장 복잡하지만 가장 현실적인Mesh Collider를 비롯해 다양한 Collider가 존재한다. 하지만 이번 Assign은 그래픽적 요소가 메인이며 게임적 요소는 크게 중요하지 않으므로, 매우 간단한 근사 충돌을 구현했다. 각 Object에는 원기둥 모양의 Collider가 있다고 가정하고, 두 Object 사이의 물리적 거리를 계산해서 Collider가 겹쳤다고 판단되면 공의 speed를 바꾸어 유사 탄성 충돌을 구현했다.

Ball의 z좌표가 boundary로 정해놓은 지점을 넘을 때, 캐릭터가 공을 막지 못한 것으로 판정하여 상대의 점수를 1점 추가한다. 한 플레이어의 점수가 10점을 넘어가면 그 플레이어의 승리로 판정하고 게임을 종료한다.

1. **점수 및 승패 판정 표기**

이전의 Assign에서는 OpenGL 내장 함수를 통해 손쉽게 텍스트를 화면에 출력할 수 있었다. 하지만 GLSL을 활용해서 직접 텍스트를 그리기 위해서는 어떤 라이브러리를 사용하든 결국 텍스쳐 렌더링의 영역에 들어가야 한다. 따라서 이번 Assign에서는 화면에 점수를 띄우는 대신 cmd창에 점수를 띄우는 방식을 택했다.

# **시연**

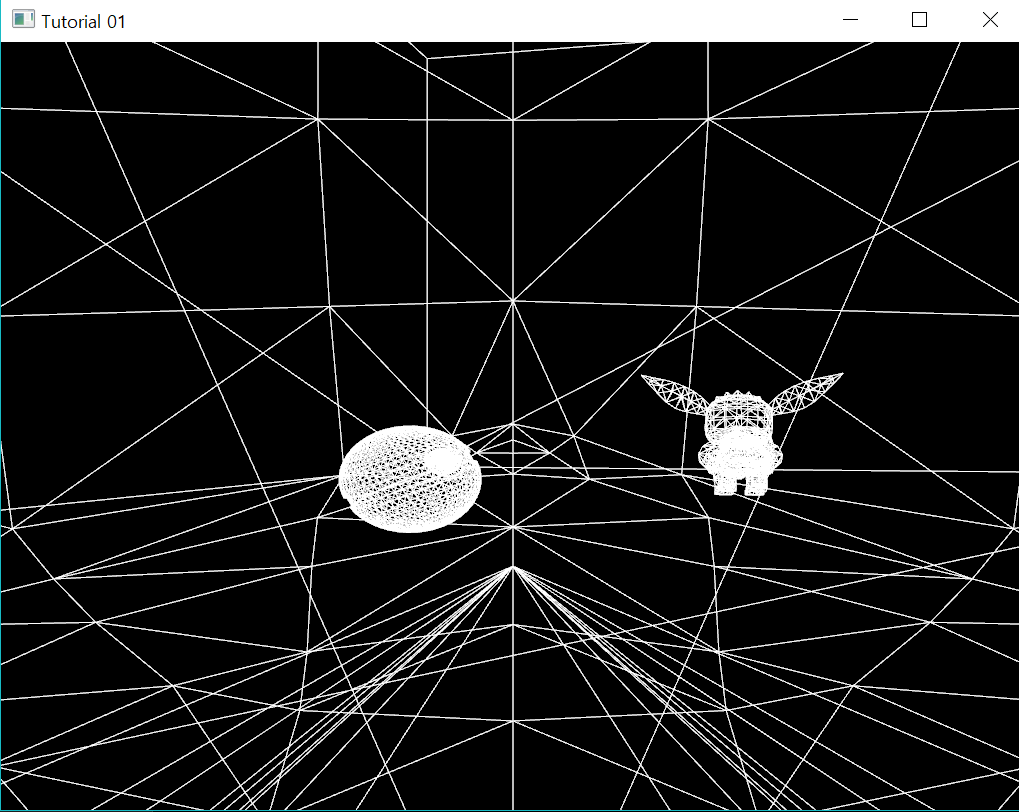


Figure 1 View 1을 적용한 게임 플레이 화면이다

스크린샷이(가) 표시된 사진

자동 생성된 설명

Figure 2 View 2를 적용한 게임 플레이 화면

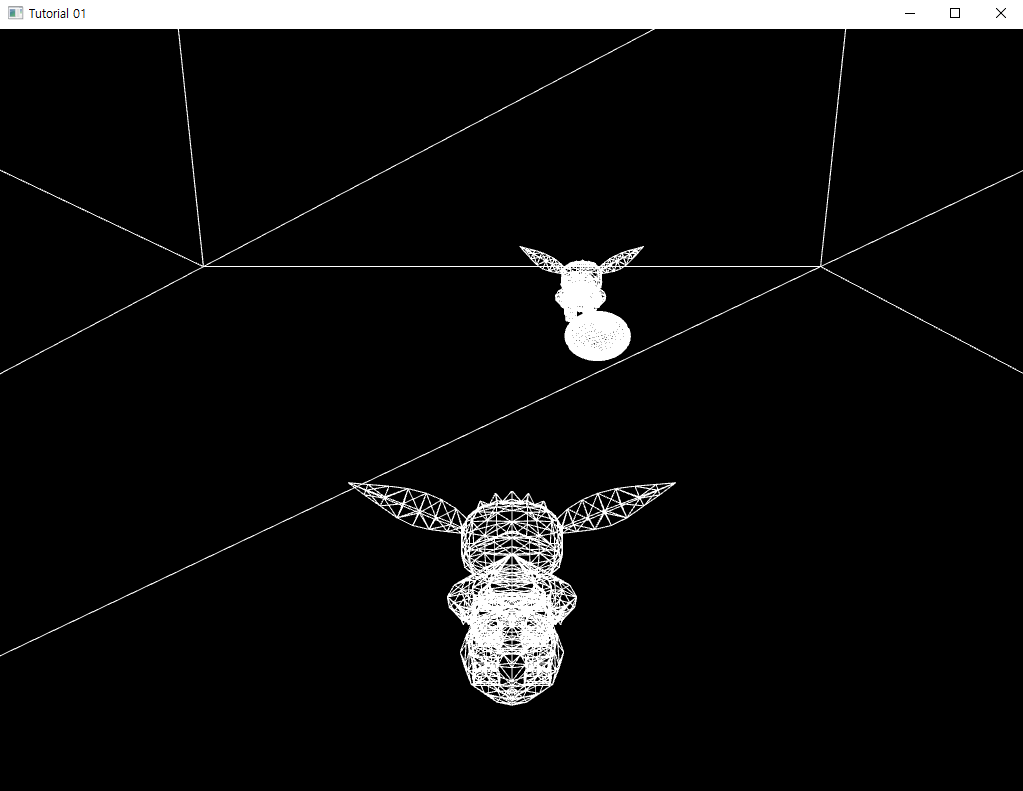


Figure 3 View 3을 적용한 게임 플레이 화면이다.

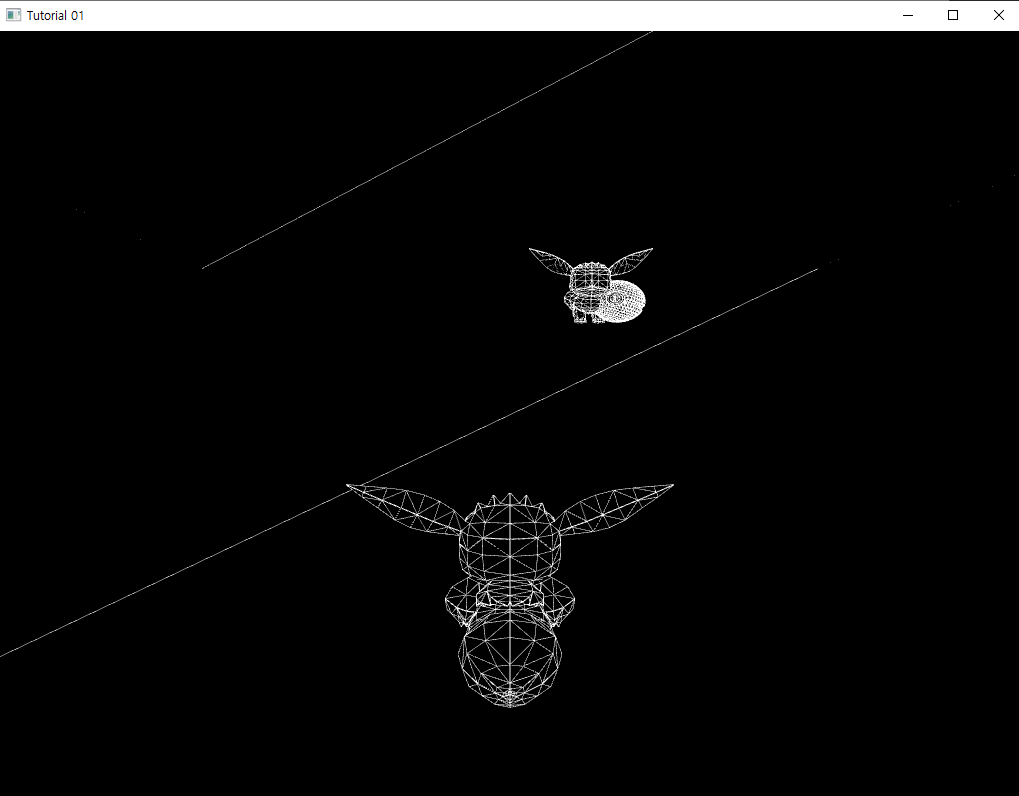


Figure 4 ‘H’키를 눌러 hidden line removal을 적용한 상태이다.

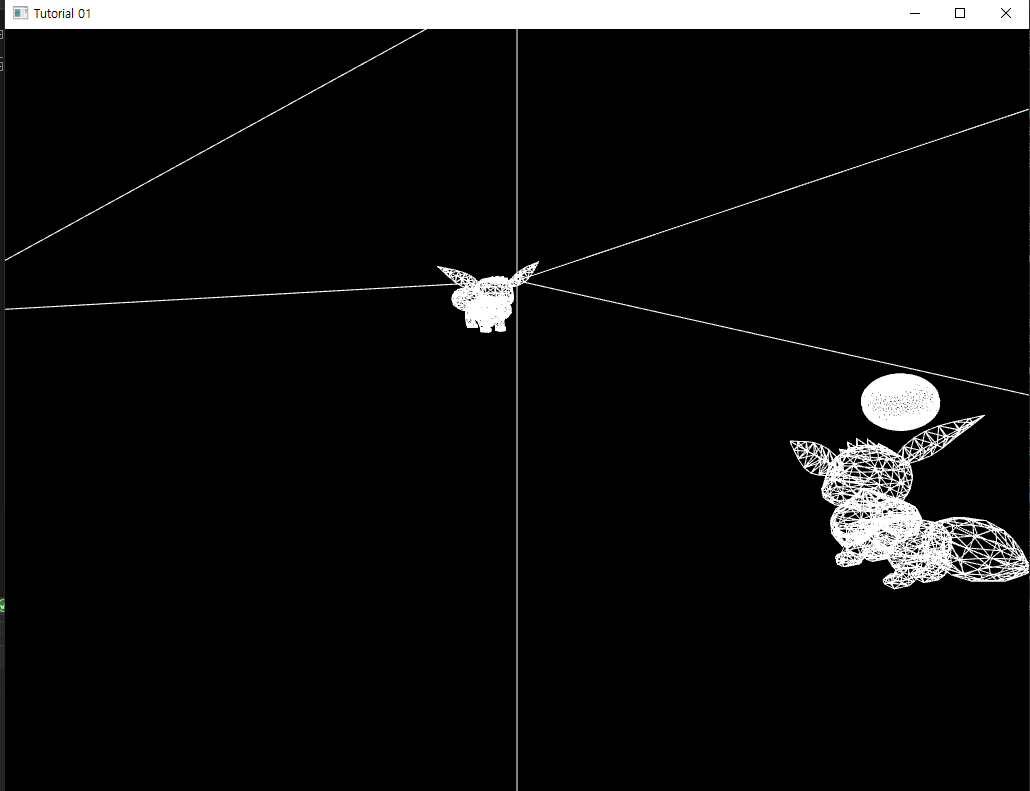


Figure 5 View 3 모드에서 카메라의 위치를 이동한 모습이다.

# **토론**

이번 Assign에서는 Assn2에서 구현한 것 처럼 계층 구조와 애니메이션을 전혀 사용하지 않고 딱딱한 이브이 모델링을 사용하여 포켓몬 게임에서 흔히 볼 수 있는 귀여운 이브이와는 거리가 먼 모습을 보여주었다. Assn4에서 이브이 텍스처를 사용하고 또 계층 구조를 활용해서 꼬리가 흔들리거나 귀가 쫑긋 거리는 애니메이션을 만든다면 더 귀여운 이브이 모델링이 가능할 것이다.

# **결론**

이번 과제에서는 Graphics Pipeline의 Vertex Shader와 Fragment Shader를 직접 program해 Game 화면을 그려보았다. 이전과는 달리 transformation matrix를 직접 관리함으로써 여러 카메라 모드를 구현할 수 있었다. 또한 wireframe mode와 hidden line removal을 적용한 Render를 구현할 수 있었다.

# **개선 방향**

이전에 OpenGL의 내장 함수를 사용해서 그렸을 때는 `glutBitmapCharacters’와 같은 함수를 이용해 비교적 쉽게 점수를 비롯한 텍스트를 화면에 출력할 수 있었는데 내장 함수를 사용하지 않고 구현하는 것은 어떤 라이브러리를 사용하는 가와는 무관하게 텍스처 렌더링이 들어가서 구현할 수 없었다. 다음 과제 때 Texture Mapping을 배우고 하면 비교적 쉽게 구현할 수 있을 것이다.

# **참고자료**

Vertex Shader / Fragment Shader Binder : <http://www.opengl-tutorial.org/kr/beginners-tutorials/tutorial-2-the-first-triangle/>

Obj File Loader : <http://www.opengl-tutorial.org/kr/beginners-tutorials/tutorial-7-model-loading/>