

# Assignment #3-1: 3D Drawing

---

CSED451 Computer Graphics (Spring 2021) Assignment #3

컴퓨터화상처리

최은수 (컴퓨터공학과, 20180050, ches7283),

권민재 (컴퓨터공학과, 20190084, mzg00)

## 개요

---

이번 과제는 기존의 ASSN2에서 구현했던 탄막 슈팅게임을 3D로 확장하여 재구현 하는 것을 목표로 한다. 게임 내부 기능(backend)은 거의 동일하지만 object 들을 2D에서 3D로 확장하는 동시에 와이어프레임 렌더링과 3D 카메라를 제어해보는 것을 주요 목표로 삼는다. 이전 과제를 확장한 형태의 과제이었기 때문에, **본 보고서에는 ASSN2에서 변경 및 확장된 부분에 대해서만 자세히 다루고 있으며 기존의 게임 구현과 동일한 부분 (class, method, 실행방법 등)은 작성하지 않았다.**

## Character

- 3D 우주선 모델을 load하여 표현한다.

## Background

- Ground를 grid로 표현한다.
- 행성계의 천체들을 모두 구(sphere)로 표현한다.

## System

- 총알과 아이템도 3D 구(sphere)로 표현한다.
- Ground의 일정부분을 게임 boundary로 설정하여 기존 ASSN2의 window처럼 우주선과 총알이 나가지 못하는 경계선으로 설정한다.

## Viewing

두 개의 Viewing mode 를 구현하여 'v' key 를 누르면 Viewing mode 가 전환되게 한다.

- 삼인칭 시점: 카메라가 플레이어 뒤의 적절히 위치 (Default)
- 일인칭 시점: 카메라가 플레이어의 앞에 적절히 위치

## Rendering

두 개의 Rendering mode 를 구현하여 'r' key 를 누르면 Rendering mode 가 전환되게 한다.

- 모드 1: Hidden Line Removal이 적용되지 않은 와이어프레임 렌더링 (Default)
- 모드 2: Hidden Line Removal이 적용된 와이어프레임

## 프로그래밍 환경

---

## 개발 환경

- Visual Studio 2019

## 라이브러리 버전

- OpenGL 4.1
- FreeGLUT 3.2.1
- GLEW 2.2.0
- GLM 0.9.9.9

## 프로그램 설계 및 구현

### Load .obj

`obj` 파일은 아래와 같은 구성 요소를 포함하고 있다.

Header	Format	Content
<code>v</code>	<code>x y z</code>	Vertex Coordinate
<code>vt</code>	<code>x y</code>	UV Mapping
<code>vn</code>	<code>x y z</code>	Normal vec
<code>f</code>	<code>v/vt/vn</code>	Face (Vertex Index)

해당 파일을 효율적으로 파싱하고 그 데이터를 저장하기 위해서 `Model` 클래스를 이용한다. 이번 과제에서는 vertex의 좌표 정보와 face만 파싱하면 와이어프레임을 출력할 수 있지만, 차후 VBO를 사용할 때를 대비해서 모든 정보를 파싱하도록 만들었다. class `Model`에서는 우선 파일의 경로를 입력 받으면, 해당 경로의 파일을 읽어서 위와 같은 형식으로 데이터를 파싱하여 저장한다. 이후, 임시로 저장한 Vertex 정보를 Face에 적혀있는 인덱스 순으로 정렬하여 `GL_TRIANGLE_STRIP`으로 출력할 수 있는 형태로 클래스 멤버 변수에 저장한다. 이후 다른 클래스에서 해당 변수의 값을 이용해서 모델을 출력할 수 있다.

### New Shape

#### Grid

Grid는 배경을 표현하기 위한 클래스이다. 가로 세로 칸 수와 각 칸의 높이와 너비를 입력받아서 그에 상응하는 그리드를 `GL_LINE`을 이용해서 그릴 수 있도록 했다. row에 대해서 가로 선을 그리고, column에 대해 세로 선을 그려서 나타냈다. 이 클래스를 이용해서 기본 배경을 그리고, **추가적으로 플레이어가 움직일 수 있는 범위에 대해서 파란색으로 별도의 그리드를 표시했다.**

# Sphere

Sphere는 latitude와 longitude를 나눠서 `TRIANGLE_STRIP`으로 출력하는 방식을 채택했다. 크게 2단계에 걸쳐서 vertex vector를 설정하였으며, 해당 벡터를 기존 `Shape` 클래스의 `modelFrame`에 설정하여 출력할 수 있도록 하였다.

Vertex를 설정하기 위해 우선 vertex들의 위치를 설정하였다. 반지름의 길이를  $r$ , latitude의 개수를  $A$ , longitude의 개수를  $O$ 라고 하자. 우선  $z$  좌표를 latitude에 따라 설정할 것이다. 현재 설정하고자 하는 좌표가  $a$ 번째 latitude 위의 점이라고 하자. 이때  $\theta = \frac{a}{A}\pi$ 라고 하면, 이 점의  $z$  좌표는  $r \cos \theta$ 와 같다. 이제 이 latitude 위에서 longitude에 따른 좌표를 확인하자. 이 구를 위에서 구한  $z$  좌표를 지나는  $xy$  평면에 평행한 평면으로 잘랐을 때 볼 수 있는 원을 생각해 보자. 이때,  $\phi = \frac{o}{O} \times 2\pi$ 라고 하면 이 원의 반지름  $r' = r \sin \theta$ 라고 할 수 있다. 이때, 이 점의  $x, y$  좌표는  $(r' \cos \phi, r' \sin \phi)$ 라고 생각할 수 있다. 이러한 아이디어를 이중 반복을 이용해서 적용시키면 구를 이루는 vertex들을 구할 수 있다.

이제 이 vertex들을 `TRIANGLE_STRIP`으로 출력할 수 있도록 순서대로 배치해야 한다. 각 vertex를 (`latitude`, `longitude`)와 같이 표현해 보자. 이때,  $a$ 번째 latitude 위의  $o$ 번째 점에 대해  $(a, o)$ 와  $(a + 1, o)$ 를 순서대로 계속 찍으면 `TRIANGLE_STRIP`으로 구를 출력할 수 있다. 그렇기 때문에 위와 같은 순서로 vertex를 배치하고 shape의 `modelview`로 설정하면 구를 출력할 수 있다.

## Viewing

**frontCamera** : 카메라의 시점 변환을 위한 1 또는 -1인 계수

(1) 사용자가 키보드에서 'v'를 눌렀을 경우 `frontCamera`를 `-1 * frontCamera`로 수정한다.

(2) camera 이동

매 frame display 될 때 `gluLookAt`을 이용하여 카메라 위치를 설정해 준다.

카메라의 위치 벡터의 경우

- $x$  : 우주선의  $x$ 좌표
- $y$  : 우주선의  $y + \text{frontCamera} \times \text{distance}$ 로 설정하여 시점에 따라  $y$ 값이 변경되도록 구현한다

이렇게 구현할 경우 우주선이 움직이면 그에 따라  $x, y$  값도 바뀌어 적용되므로 카메라가 우주선 위치에 종속적으로 움직인다.

## Rendering

**isHiddenLineRemoval** : hidden line removal 모드인지를 저장하는 boolean 변수.

(1) 사용자가 `r`을 눌렀을 경우에, `isHiddenLineRemoval`의 값을 `!isHiddenLineRemoval`로 변경한다.

(2) polygon offset & fill

먼저 rendering 모드에 상관 없이 모든 object는 자신의 color와 vertex들을 가지고 `GL_FRONT_AND_BACK`, `GL_LINE` 모드를 이용해 wire frame으로 그려진다.

그리고 hidden line removal 모드 일 경우,

`glPolygonOffset`를 설정하고, 이미 그려진 object들의 vertex들을 `GL_FRONT_AND_BACK`, `GL_FILL` 모드를 이용해 각 면을 채워 준다.

각 면의 색을 검정색으로 칠하게 되면 뒤쪽 object들은 면에 가려지기 때문에 hidden line removal이 적용된 wire

frame 이 display 된다.

## 프로그램 실행 방법

### 우주선 이동

- 키보드의 **상하좌우** 키를 누르면 그에 따라 우주선이 상하좌우로 이동한다.
- 우주선은 선체의 일부분도 window 밖으로 나갈 수 없으며 적이 있는 위치까지 제한없이 이동 가능하다.

### 총알 발사

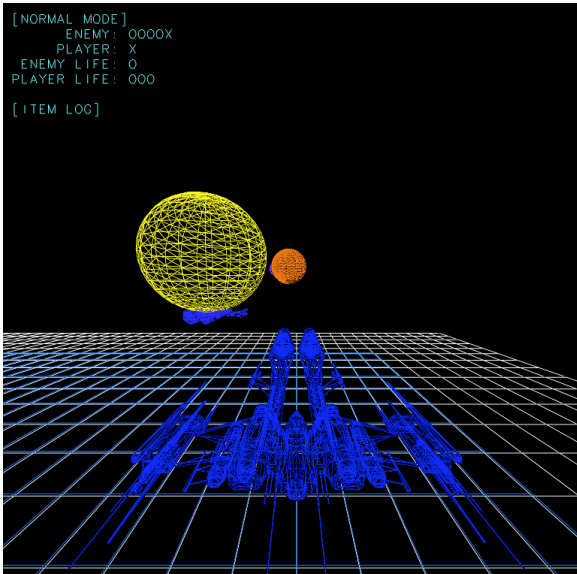
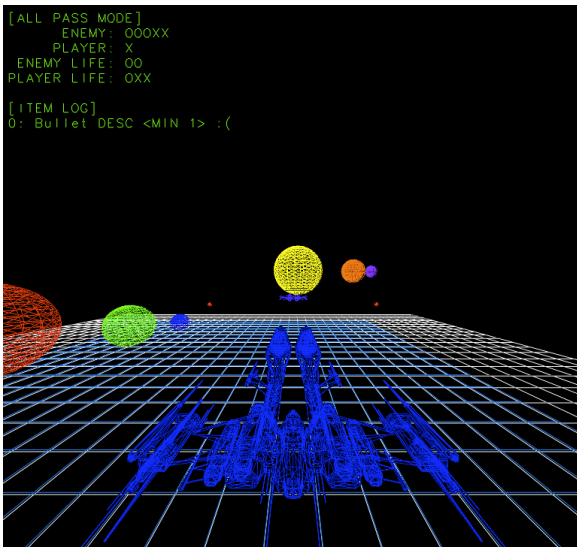
- 키보드의 **space bar** 를 누르면 삼각형의 위쪽 꼭짓점을 통해 총알을 발사할 수 있다.
- 발사 가능한 총알 개수에는 제한이 없으며 이동과 동시에 총알 발사도 가능하다.

### 모드 변경

- **C** : 키보드의 'c' 키를 누르면 올패스 모드로 전환된다.
- **F** : 키보드의 'f' 키를 누르면 올패일 모드로 전환된다.
- **R** : 키보드의 'r' 키를 누르면 rendering mode가 변경된다.
- **V** : 키보드의 'v' 키를 누르면 viewing mode가 변경된다.

## 예제

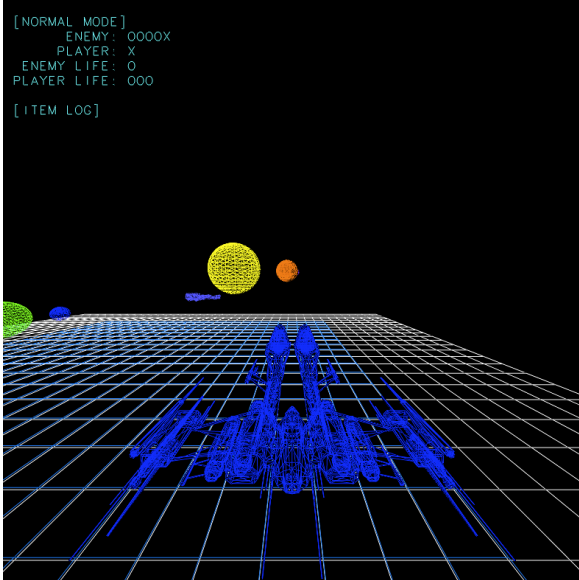
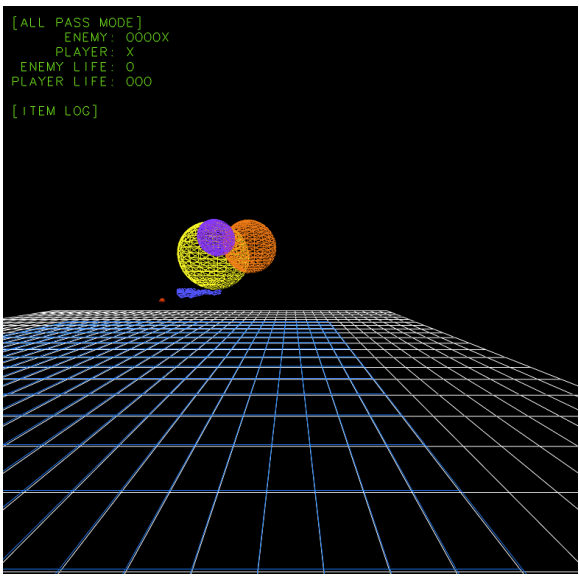
### background

grid와 sphere	item/총알 sphere
<pre>[NORMAL MODE] ENEMY: 0000X PLAYER: X ENEMY LIFE: 0 PLAYER LIFE: 000 [ITEM LOG]</pre> 	<pre>[ALL PASS MODE] ENEMY: 000XX PLAYER: X ENEMY LIFE: 00 PLAYER LIFE: 0XX [ITEM LOG] 0: Bullet DESC &lt;MIN 1&gt; :{</pre> 

- 행성계의 천체들과 총알, 아이템 모두 3D 구를 이용하여 출력

- 바닥은 Grid로 표시하되, 파란색으로 boundary를 별도로 출력

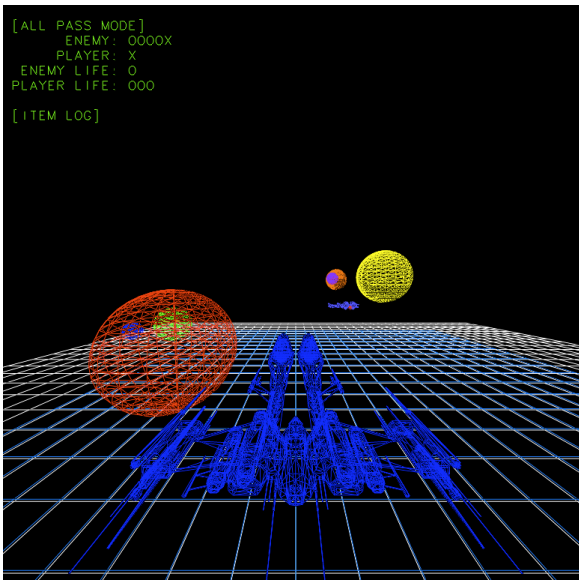
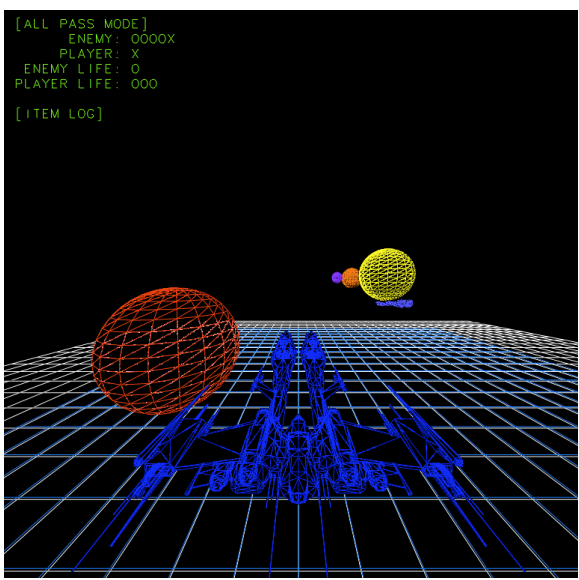
## Viewing

3인칭 view	1인칭 view
<pre>[NORMAL MODE]   ENEMY: 0000X   PLAYER: X   ENEMY LIFE: 0   PLAYER LIFE: 000 [ITEM LOG]</pre> 	<pre>[ALL PASS MODE]   ENEMY: 0000X   PLAYER: X   ENEMY LIFE: 0   PLAYER LIFE: 000 [ITEM LOG]</pre> 

**v** 키를 눌러 viewing mode를 변경할 수 있다. 그리고 player가 이동함에 따라 그 위치에 종속적으로 카메라도 이동한다.

- 3인칭 view : player 우주선의 뒤쪽에 위치
- 1인칭 view : player 우주선 앞쪽에 위치

## Rendering

normal rendering	hidden line removal
<pre>[ALL PASS MODE]   ENEMY: 0000X   PLAYER: X   ENEMY LIFE: 0   PLAYER LIFE: 000 [ITEM LOG]</pre> 	<pre>[ALL PASS MODE]   ENEMY: 0000X   PLAYER: X   ENEMY LIFE: 0   PLAYER LIFE: 000 [ITEM LOG]</pre> 

**r** 키를 눌러 Rendering mode를 변경할 수 있다.

- normal : hidden line removal이 적용되지 않아 뒤쪽 다른 object들이 보이는 와이어프레임
- hidden line removal : hidden line removal이 적용되어 뒤쪽에 위치한 object들이 보이지 않는 와이어프레임

## 토론

### GetPosition from World-coordinate

PA1, PA2에서는 modelframe을 다양한 transformation으로 world-coordinate에 맵핑한 후, vertex를 그릴 때의 modelview를 저장해 놓았었다.

그 후 특정 object의 WC 좌표가 필요할 때 ModelView \* Vertex를 해서 좌표를 구했었다.

이는 2D나 카메라가 (0, 0)에 고정되어 있을 때는 전혀 문제가 되지 않았다.

하지만 이번 PA3처럼 camera가 이동하는 경우 glLookAt 함수를 사용하는데, glLookAt함수는 ModelView Matrix를 사용하므로 이전처럼 좌표를 구할 수 없다.

그래서 worldcoordinate 좌표를 Translate node의 (x, y)를 기준으로 구하도록 변경하였으며, 총알처럼 각도를 가지고 움직이는 object의 경우는 direction(x, y)를 추가하여 해당 방향으로 translate 하도록 구현하였다.

하나 다양한 transformation이 적용된, 복잡한 WC 좌표를 구하는데 한계가 있을 것 같아 어떻게 WC좌표를 구하는 것이 효율적이고 좋은 디자인인지에 대한 논의를 하였다.

## 추가 구현

### 총알 Reflection

총알이 2개 이상 나갈 경우, 총알의 x좌표의 값도 변하게 된다. 이 때 window 좌우에 총알이 부딪히게 되는데 이때 입사각과 같은 각으로 총알이 반사되어 나가도록 구현하였다.

총알이 반사될 경우, 총알이 움직이는 direction의 y성분은 변하지 않고 x성분만 반대로 (부호가 반대로) 변하게 된다. 따라서 direction(x, y) -> direction (-x, y) 로 바꿔주면 계속 총알이 앞으로 나가되, 반사되어 나가게 된다.

### 스코어보드

기존 과제에서 구현하였던 스코어보드를 카메라의 위치에 상관 없이 출력할 수 있도록 구현을 수정하였다. 우선 좌표축과 projection matrix를 초기화하고 스코어보드를 원하는 위치에 출력했다. 해당 작업 이후에, perspective를 설정하는 matrix들을 적용시키고 게임에 필요한 오브젝트들을 출력시켜서 게임 오브젝트들에는 카메라의 시점이 적용되지만 스코어보드에는 카메라 시점이 적용되지 않도록 했다. 일련의 과정을 통해 언제나 같은 위치에 스코어보드 정보가 표시될 수 있도록 구현하였다.

### 멀티 키 입력

기존 과제에서 구현했던 멀티 키 입력을 그대로 구현하였다. 자세한 사항은 이전 과제의 보고서에서 확인할 수 있다.

## 결론

---

1. obj 파일을 파싱하고 출력하는 방법을 익힐 수 있었다.
2. 3D polygon 을 drwaing 하는 다양한 방법 및 특성을 활용하여 triangle mesh로 3차원 물체를 그릴 수 있었다.
3. 카메라와 perspective 등 viewing의 개념과 구현을 익힐 수 있었다.
4. polygon offset을 활용하여 hidden line removal을 구현할 수 있었다.

## 개선 방향

---

### 카메라 시점 변경

현재 구현은 시점이 1인칭 혹은 3인칭 시점 2가지로만 구현되어 있고 각각은 player에 종속되어 이동하게 된다.

하지만 시중의 다른 RPG 게임들의 경우 카메라 시점과 view frame을 원하는대로 자유롭게 이동시키며 게임을 할 수 있다.

그래서 마우스의 휠로 zoom in/out이나 카메라의 위치를 유저 인풋에 따라 자유롭게 변형시키는 것을 구현하면 좀 더 심미성이 높은 게임이 될 수 있을 것으로 기대한다.

## 참고문헌

---

- OpenGL Docs (<https://www.khronos.org/opengl/>)