3D GameProgramming2 – 2nd report

2018180009 김시인

한국공학대학교

TECH UNIVERSITY OF KOREA

목차

실행 환경 : visual studio 2022, Release x64,

최신 C++ 초안의 기능(/std:c__latest)

CPU – AMD Ryzen 7 5800H, RAM 16GB, GPU - RTX 3070

조작키:

W, A, S, D – 수평이동 및 해당 방향으로 회전

space, ctrl - 상승/하강

마우스 드래그 – (3인칭인 경우)카메라 이동

Q – 미사일 발사

E – 미사일 폭우 발사

J – 카메라 인칭 변경(1인칭 <-> 3인칭)

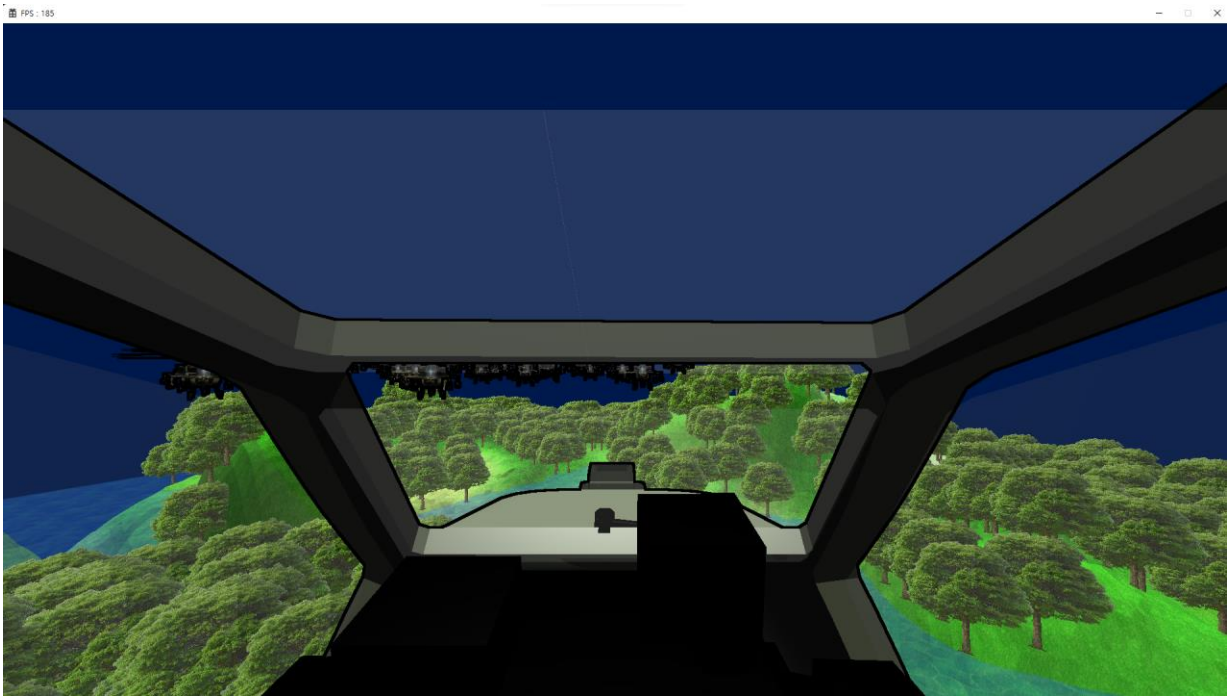
K - OOBB렌더링 on/off

L – 외곽선 렌더링 모드 변경(그리지 않음 -> 헬기만 -> 모든 오브젝트)

1. 카메라 1인칭 구현:

카메라를 헬기 안에 넣고 헬기의 창문을 반투명하게 렌더링하였습니다.

J 키를 통해 1인칭<->3인칭으로 변경이 가능하여지도록 만들었습니다.



또한 3인칭일경우 W, A, S, D가 그 방향으로 회전과 전진을 같이 하는 키가 되지만, 1인칭일 경우 S키가 후진으로 바뀌도록 만들었습니다.

<3인칭 시점>

2. 파티클 시스템 구현:

저는 성능을 높이기 위해 파티클을 인스턴싱하여 한꺼번에 그릴 수 있는 구조이면서 파티클을 쉽게 추가할 수 있는 방식을 구현해 냈습니다.

이 방식을 구현하기 위해서는 먼저 여러 리소스들을 생성해야 합니다.

uploadStreamInputBuffer : 추가할 파티클을 저장하기 위해 upload heap으로 생성한 버퍼 리소스. CPU에서 값을 쓰기 위해 map을 해준다. SO의 입력값으로 써야 하기 때문에 D3D12_VERTEX_BUFFER_VIEW를 만들어 연결해 준다.

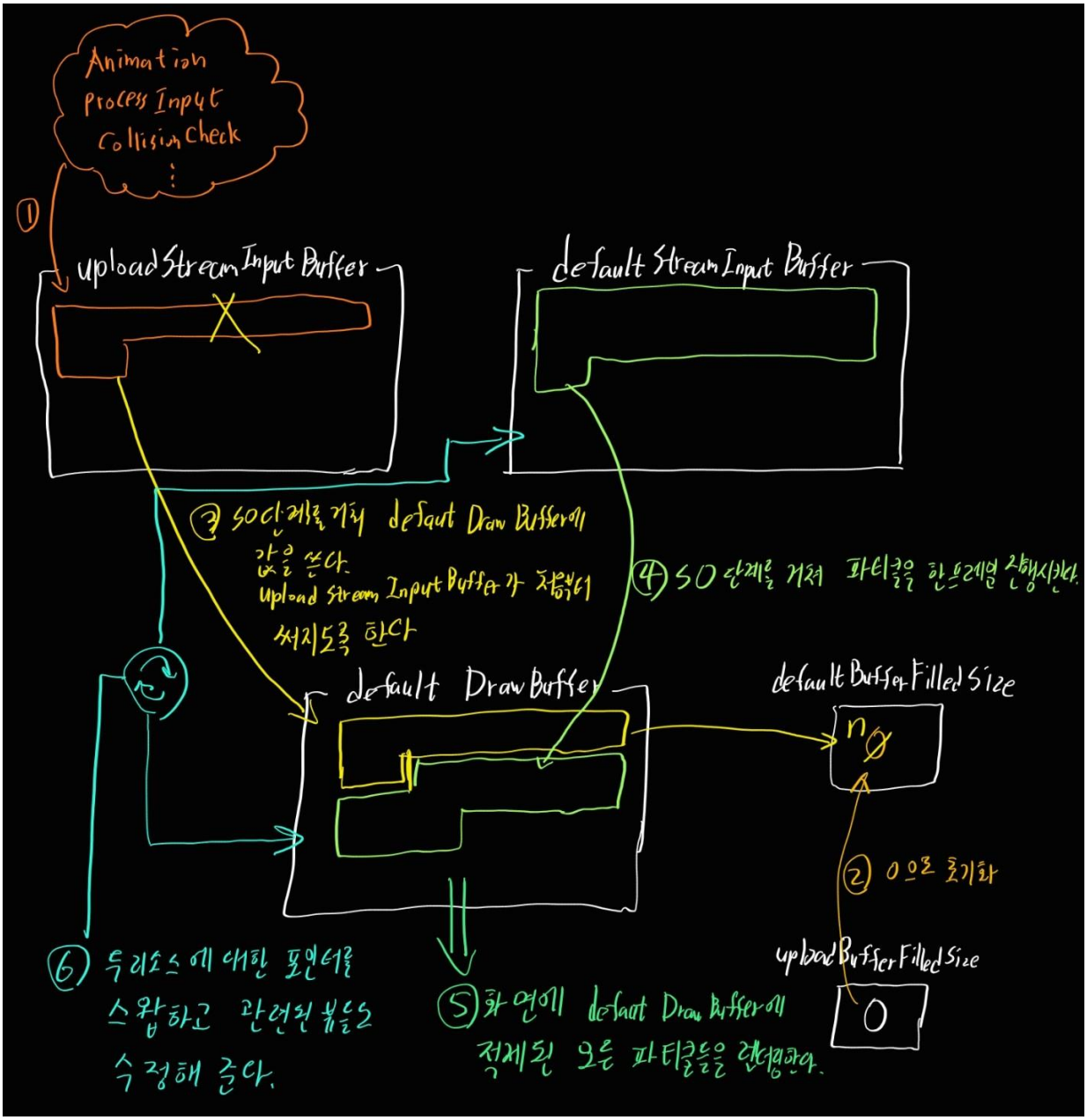
defaultStreamInputBuffer : uploadStreamInputBuffer와 같이 SO의 입력값으로 쓰일 리소스이지만 default heap으로 생성한다. 마찬가지로 D3D12_VERTEX_BUFFER_VIEW 도 만들어 연결해 준다.

defaultDrawBuffer : SO의 결과들을 쓰고, 렌더링하기 위해 사용되는 리소스로 default heap으로 생성한 리소스, SO의 결과를 쓰기위해 D3D12_STREAM_OUTPUT_BUFFER_VIEW 와 렌더링의 입력으로 사용하기 위해 D3D12_VERTEX_BUFFER_VIEW 를 만들어 준다.

defaultBufferFilledSize, uploadBufferFilledSize, readBackBufferFilledSize : 각각 SO 단계에서 몇 바이트나 쓰였는지 저장/쓰기/읽기 위한 버퍼

그 외 map을 통해 값을 읽거나 쓰기 위한 변수와 파티클의 개수를 저장하는 변수, 텍스처를 읽어 저장할 리소스와 파이프라인단계에 연결하기 위한 변수들이 필요.

전체적인 진행방식을 아래와 같습니다.



<파티클 수행 과정>

다음장에 글로 자세하게 설명되어 있습니다.

1. Animation, ProcessInput, CollisionCheck 등 여러 상황에서 AddParticle()함수를 호출한다. AddParticle()함수는 파티클의 정보를 uploadStreamInputBuffer에 저장하고 현재까지 몇 번째 인덱스까지 파티클이 추가되었는지도 함께 저장한다.

2. uploadBufferFilledSize에는 항상 0의 값이 쓰여 있고, defaultBufferFilledSize의 값을 0으로 초기화하는 역할로 사용한다. defaultBufferFilledSize의 값을 0으로 초기화하여 다시 SO단계를 진할 준비를 한다.

3. uploadStreamInputBuffer를 입력조립기에 연결하고 defaultDrawBuffer를 SO의 출력으로 연결하여 파이프라인을 수행한다. 그후에 readBackBufferFilledSize를 이용하여 몇 개의 파티클이 쓰였는지 알아낸다. D3D12_STREAM_OUTPUT_BUFFER_VIEW 의 BufferLocation을 알아낸 파티클의 개수 * 파티클하나의 크기 만큼 더해주어 그 다음 위치부터 쓰여지도록 한다. 그리고 uploadStreamInputBuffer의 내용을 옮겼으므로 다시 AddParticle()함수를 부를 때 처음부터 써지도록 만든다.

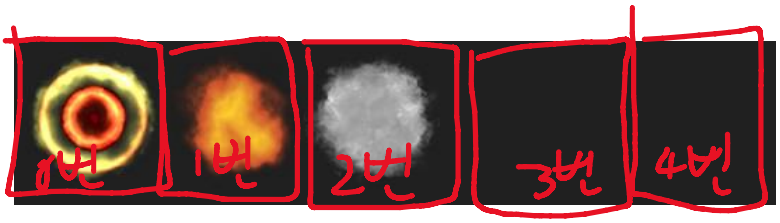
4. defaultStreamInputBuffer를 입력조립기에 연결하고 defaultDrawBuffer를 SO의 출력으로 연결하여 파이프라인을 수행한다.

5. defaultDrawBuffer의 상태를 D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER로 바꾼다. defaultDrawBuffer를 사용하여 3번과 4번에서 쓰여진 파티클의 개수만큼 한번에 렌더링을 수행한다.

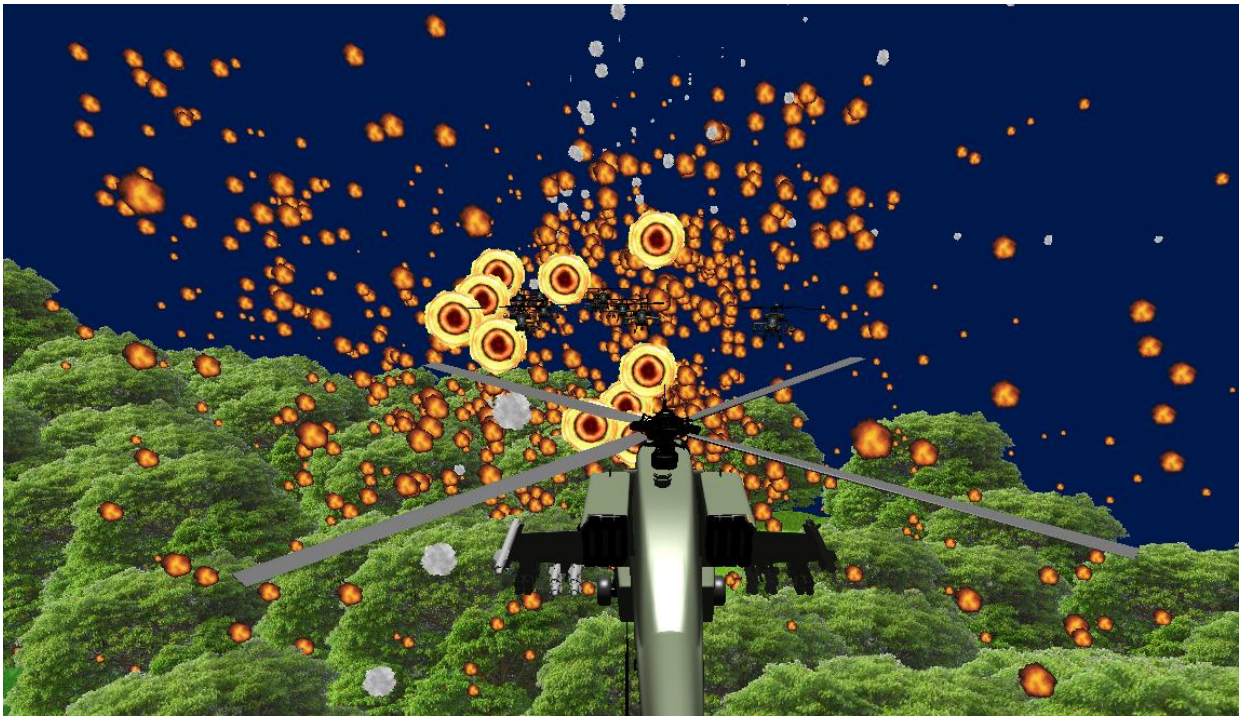
6. defaultDrawBuffer와 defaultStreamInputBuffer를 바꾼다. 관련된 뷰들도 함께 바꾼다.

허나 이렇게 구현하였을 때 readBackBufferFilledSize에 defaultBufferFilledSize를copy하고 map함수를 통해 값을 읽어오는 과정에서 copy를 하는 것은 CommandList를 통해 GPU에서 수행되기 때문에 즉시 이루지지 않고 map함수를 통해 값을 읽어오는 것은 CPU에서 바로 이루어지기 때문에 결과적으로 값을 읽어온 후 copy하는 문제가 발생합니다.

이 문제를 해결하기 위해 5번과 6번 과정을 먼저 수행하고 1~4번 과정을 수행하도록 만들어 1프레임이 차이 나지만 위의 문제가 발생하지 않도록 만들었습니다. (한 프레임 끝나면서 CommandList에 넣어둔 모든 명령이 수행되고 다음 프레임에서 값을 읽어 오기 때문에 해결된다.) 한 프레임차이는 플레이어가 거의 느낄 수 없기 때문에 문제가 되지 않는다고 판단했습니다.



파티클을 그리기 위한 텍스처는 하나의 이미지에 모든 파티클들을 해당하는 위치에 그려 넣고 파티클의 타입에 따라 uv값을 조절하여 텍스처를 입히도록 만들었습니다.



적군 헬기를 파괴할 경우 10개의 0번 파티클이 생성됩니다. 0번 파티클은 랜덤 한 방향으로 이동하게 되고 중력의 영향을 받습니다. 만약 생명주기가 끝날 경우 1번 파티클을 10개 생성합니다.

1번 파티클은 시간이 지날수록 크기가 점점 작아지며 사라집니다. 0번 파티클과 마찬가지로 중력의 영향을 받고 0번에 의해 생성될 때 0번이 가지고 있던 속도를 어느정도 가지고 생성됩니다.

2번 파티클은 연기를 표현하기 위해 만들었습니다. 미사일이 이동할 때 0.2초마다 생성되고 시간이 지나면 하늘로 올라갑니다.

3. 외곽선 구현:

다중 렌더 타겟을 이용하여 Texture2D에 깊이 값을 저장합니다.

그 다음 깊이 값이 써진 텍스처를 연결하고 화면 전체를 메우는 사각형(삼각형2개)을 렌더링 합니다.

버텍스 쉐이더에서는 정점정보와 uv정보를 그대로 픽셀 쉐이더로 전달합니다.

픽셀 쉐이더에서는 uv값이 0~1의 값을 가지고 있으므로 각각 텍셀값(0~화면의 가로/세로 크기)으로 변환시켜줍니다. 그렇게 구한 텍셀값을 이용하여 주변의 색상들을 알아내어 소벨필터 알고리즘을 이용하여 외곽선인지를 판단합니다. 소벨필터 알고리즘을 거쳐 나온 값에 절대값을 취해 일정 수치를 넘을 경우 검정색(외곽선)을 그리고 그렇지 않을 경우 discard를 통해 파이프라인에서 제외하도록 만들었습니다.

기본적으로 헬기만 외곽선으로 그리며 지형이나 빌보드로 그린 나무 등을 투시하여 볼 수 있도록 되어있습니다.

L키를 눌러 모든 오브젝트에 외곽선이 적용되게 하거나 외곽선을 그리지 않도록 모드를 변경할 수 있도록 구현하였습니다. 모든 오브젝트에 외관선을 적용할 때는 따로 지형을 관통하여 외곽선이 보이거나 하는 기능은 넣지 않았습니다.

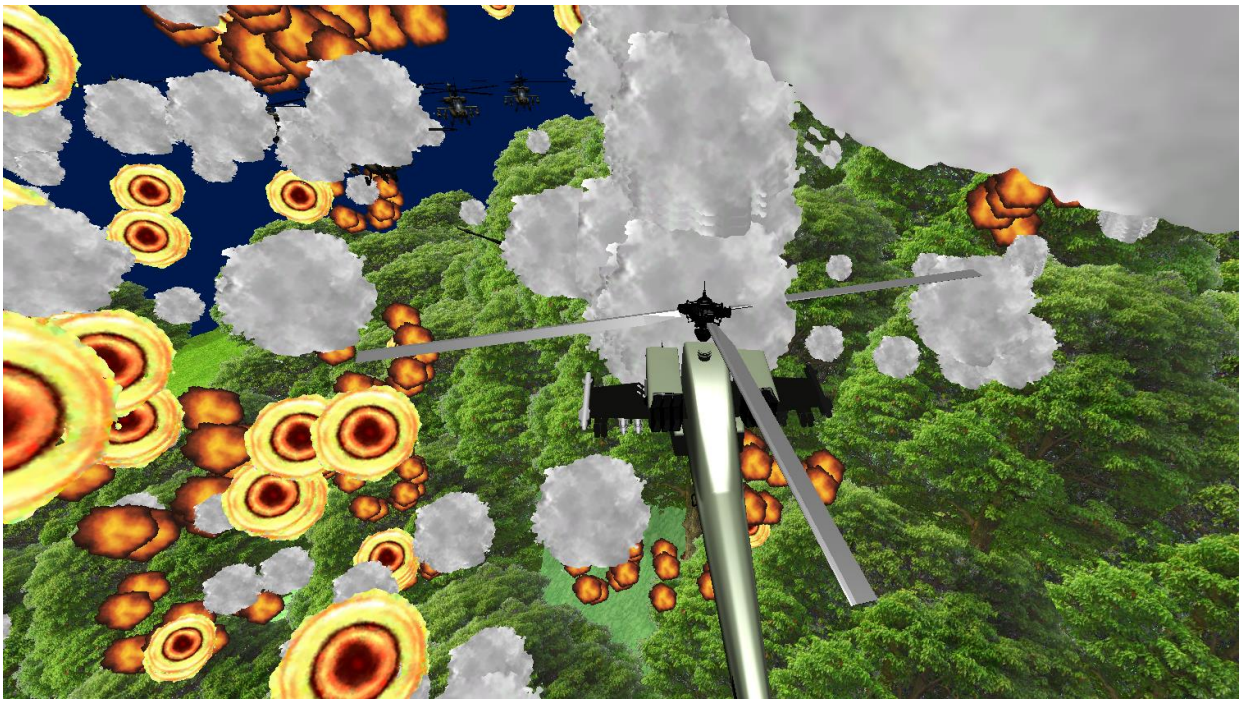
<헬기만 외곽선 그리는 모드>

K키를 눌러 바운딩 박스 렌더링을 off할 수 있습니다.

>>스크린샷의 왼쪽 위를 보시면 지형 뒤에 있는 헬기의 외곽선만 그려진 것을 볼 수 있습니다

<모든 오브젝트에 대해 외곽선 그리는 모드>

>>헬기 뿐만 아니라 지형과 이펙트까지 외곽선이 그려진 모습



<외곽선을 적용하지 않을 경우>

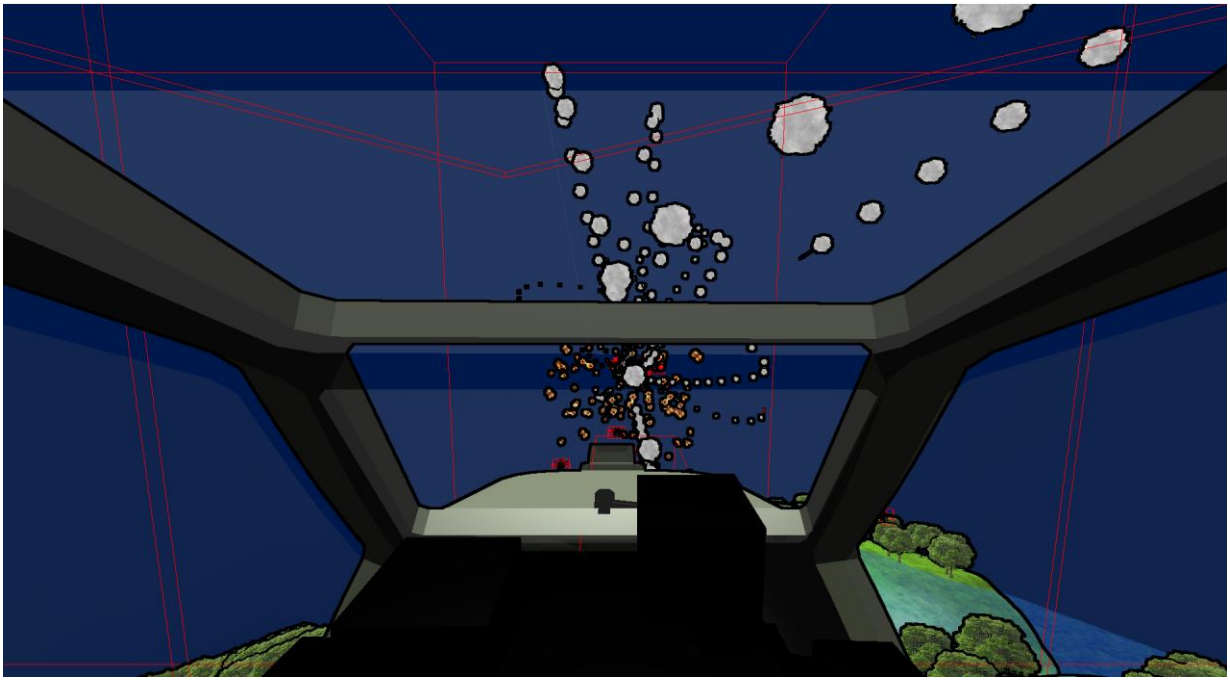
4. 스킬 구현:

E 키를 눌러 유도 미사일을 연속으로 15발 발사하는 스킬을 구현하였습니다.

15발의 미사일이 한꺼번에 발사되는 것이 아니라 짧은 시간에 연속적으로 발사되도록 만들었습니다.

각 미사일은 일정 거리 안에 적이 있는지 찾고 적이 있다면 그 적을 타겟으로 설정하고 따라갑니다.

타겟이 죽어서 사라지면 새로운 타겟을 찾습니다.
스킬은 5초의 쿨타임이 존재합니다.



<게임 실행 화면>