|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 17 주차 | **기간** | 2020.04.20 ~ 2020.04.26 | **지도교수 이용희** | (서명) |
| 이번주 한일 요약 | -락 온 타겟 변경  -락 온 타겟 변경에 따른 미사일 유도  -프레임 레이트 고정 수정  -애프터 버너 효과 변경  -키매니저 버그 수정  -카메라 시선 고정  -구름 인스턴싱  -파일매니저 작성 및 미사일 폭발 효과 | | | | |

<상세 수행내용>

깃 주소 : <https://github.com/zzcx88/DKS_Project.git>

[](https://www.youtube.com/embed/uy2uWouOsBo?feature=oembed)

**김령운 :**

**락 온 타겟을 플레이어가 특정 키를 눌러서 직접 지정할 수 있게 하는 기능을 구현했다. 범위 내의 적 오브젝트들을 vector 자료구조에 저장하고 정렬하여 순서대로 접근할 수 있도록 했다.**

1. **락 온 타겟 변경**

**플레이어가 원하는 타겟을 특정 키로 지정할 수 있게 하기 위해서는 우선 타겟을 저장할 자료구조가 필요했다. 락 온 범위 내의 타겟이 얼마나 있을지 알 수 없기 때문에 배열 크기의 가변성을 보장해주는 vector를 저장할 자료구조로 사용했다.**

전 주차에 작성했었던 UIManager::MoveLockOnUI(오브젝트 매니저 클래스에 저장되어 있는 오브젝트들을 순회하며 락 온 UI를 그리는 함수)에 범위 내의 타겟들에게 순차적으로 접근하는 기능을 추가 구현했다.

적 오브젝트가 플레이어 조준 범위 내에 있을 경우 적 오브젝트를 벡터에 emplace\_back 하여 생성하고 월드 좌표 x 값 순으로 오름차순 정렬했다.

**그림1. Vector 원소 추가, 정렬**



 **그림1-1. Vector 원소 추가, 정렬**

오브젝트의 소멸을 판단하는 bool 값인 m\_isDead를 리턴하는 GetState() 함수를 통해 미사일과의 충돌로 오브젝트가 소멸했을 경우 소멸한 객체의 UI 락 온 텍스처에 접근하는 것을 방지하기 위해 조건문을 아래와 같이 설정했다.

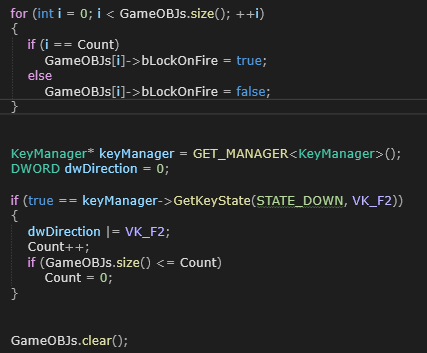
**그림1-2. 객체의 소멸 여부와 락 온 범위 내에 있는지 확인하는 조건문**

**그림1-3 CGameObject Class의 GetState() 함수**

****

그렇게 Vector에 정렬된 락 온 범위 내의 오브젝트들을 Vector의 사이즈만큼 반복문을 돌면서 접근한 오브젝트의 bLockOnFire bool 값을 true로 설정하고 이외의 오브젝트들의 bool 값은 모두 false로 설정했다.

**그림2. Vector 순회 반복문**

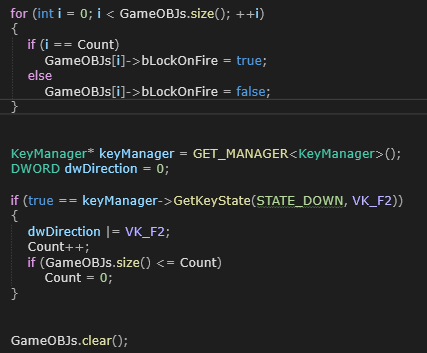
****

Vector의 원소에 접근하는 방법은 Vector의 사이즈만큼 반복문이 돌 때 변수로 만들어 놓은 Count의 값으로 접근하게 했다. Vector 키보드의 F2 키를 누를 때마다 Count 변수의 값을 0부터 숫자 1을 더해가도록 하고 Vector의 사이즈를 넘어갈 경우 Count의 값을 0으로 초기화 했다.

이제 F2를 누를 때마다 Vector의 원소에 순차적으로 접근할 수 있게 되었고 접근한 원소만 락 온을 판단하는 bool 값이 true로 바뀌게 설정했기 때문에 플레이어가 직접 원하는 타겟을 지정할 수 있게 됐다.

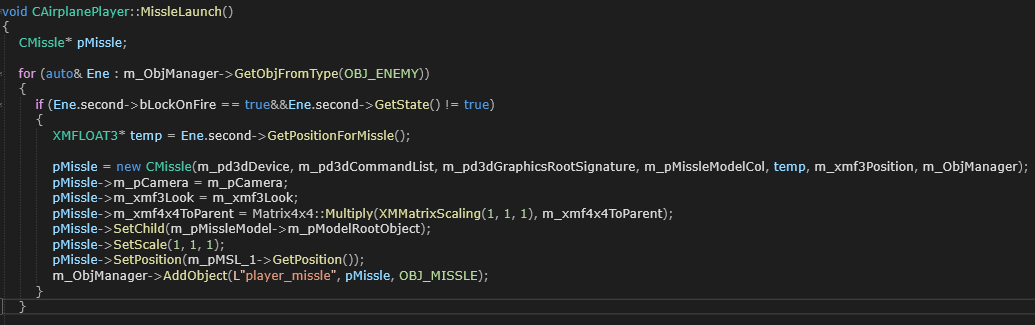
마지막으로 모든 오브젝트를 순회하고 함수가 끝나기 전에 Vector.clear()를 호출하여 벡터의 모든 원소를 삭제하도록 했다.

**그림3. 키 매니저 클래스를 통한 키 입력(Count 변수 증가)**

****

1. **타겟 변경 미사일 유도**

**그림4. CAirPlanePlayer Class의 MissleLaunch()함수**



타겟 변경이 될 때마다 유도 미사일을 타겟으로 발사하기 위해서 이재원 팀장이 작성했던 CAirPlanePlayer 클래스를 조금 수정하여 구현했다.

기존의 미사일 클래스는 특정한 적 오브젝트의 좌표를 받아와서 유도되는 미사일을 발사하도록 작성 돼있었다. 그래서 특정한 적 오브젝트가 소멸 시 다음 미사일 발사 때 좌표를 받아오지 못해 프로그램에서 오류가 났었다.

이를 해결하기 위해서 GetObjFromType 함수를 이용해 OBJ\_ENEMY 태그를 가진 오브젝트들을 가져와 범위기반 for 문을 이용해 락 온이 True이고 소멸 상태가 아닌 적 오브젝트를 찾아서 유도 미사일이 발사되도록 수정했다.

**그림5. ObjectManaget Class의 GetObjFromType()함수**



오브젝트 매니저가 소멸된 객체는 multimap에서 바로 삭제해주기 때문에 미사일과 적 오브젝트가 충돌 후 GetObjFromType 함수로 오브젝트 매니저에 저장된 적 오브젝트의 리스트를 가져온다면 소멸된 객체는 이미 리스트에서 삭제되고 없는 상태기 때문에 소멸되지 않은 오브젝트들 중에서 다음 락 온 타겟을 지정하고 유도 미사일 발사가 가능하다.

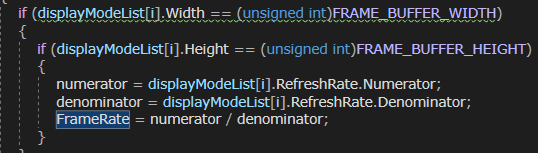
**이재원 :**

**1.프레임 레이트 고정 수정**

**모니터마다 주사율이 다름으로 각 모니터의 주사율에 대응되는 프레임 레이트를 계산하여 프레임 레이트를 고정한다.**

우선 계산되어 나온 고정 프레임 레이트 수치를 저장할 unsigned int 변수 FrameRate를 선언한다.

그 후 수직동기화를 위한 RefreshRate를 도출하고 나온 값들을 나누어 FrameRate에 저장한다.

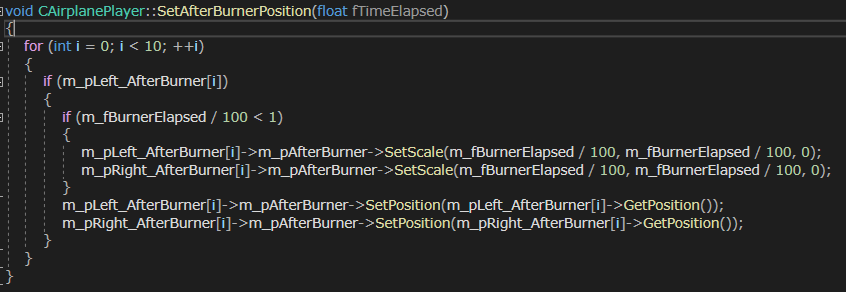


저장된 프레임 레이트를 게임 타이머에 적용시킨다.



**2. 애프터 버너 효과 변경**

기존에 만들었던 애프터 버너 효과를 가감속에 따른 스케일 값 변화를 통해 효과를 변경하였다.



애프터 버너의 개수가 좌,우측 에 각각 10개씩 존재함으로 for문을 통하여 처리를 한다.

m\_fBurnerElased는 플레이어가 키보드 키 W(가속), S(감속)을 누르는 동안 프레임마다 +1 혹은 -1이 누적되어 증감되는 수치이다.



따라서 W키를 누르는 동안 버너효과가 커지고 S키를 누르는 동안엔 줄어든다.

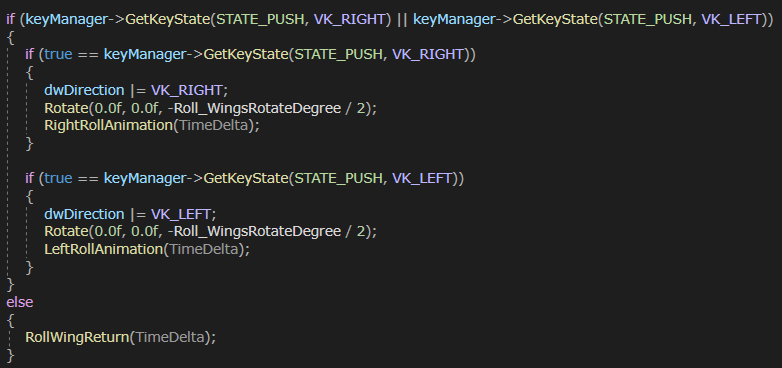
해당 함수는 플레이어의 Animate마다 매프레임 호출된다.

**3. 키매니저 버그 수정**

**키매니저의** KeyStateUp **상태, 즉 눌렀다가 떼었을 때의 상태가 매 프레임마다 TRUE, FALSE가 스위치 마냥 반복되는 버그가 발견되어 애니메이션이 프레임의 절반만 수행되는 버그를 수정**

플레이어에서 호출되는 키매니저의 호출 중 STATE\_UP을 사용하는 가속, 감속, 그리고 날개를 원위치 시키는 작업들이 매 프레임 true, false가 반복되어 프레임 레이트의 절반밖에 애니메이션이 실행되지 않고 있었다.

해당 버그를 수정하기 위해 다음과 같이 수정하였다.



롤 애니메이션과 롤 회전을 처리하는 부분을 예시로 보자면 오른쪽 혹은 왼쪽 방향키가 눌렸을 때 행위를 합치고 if문 안에서 행위를 나누었다. 위 방향키 중 만약 아무것도 눌리지 않았다면 RollWingReturn()이 호출되어 날개가 원위치 되도록 하였다.

다른 애니메이션도 마찬가지의 구조로 수정하였으며 핵심은 STATE\_UP을 이젠 눌렀다가 뗐을 경우에만 사용하게끔 바꾸었다.

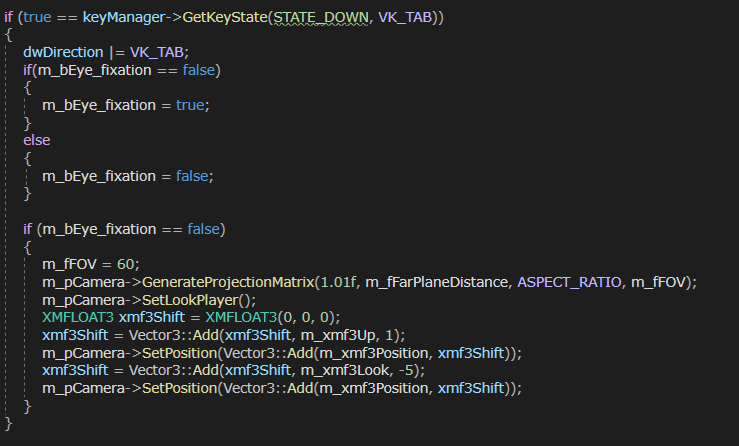
하지만 눌렀다가 뗐을 때 현재 사용하는 기능이 특별히 없음으로 사실상 STATE\_UP은 사용하지 않게 되었다.

**4. 카메라 시선 고정**

**고정된 적 타겟에 시선을 고정하는 기능을 추가했다.**

**3인칭 시점의 이점을 활용할 수 있는 기능이다.**

플레이어가 현재 시야에서 보이지 않는 타겟을 공격하고 싶거나 꼬리를 잡혀 위험한 상황이 되었을 경우 해당 타겟을 주시 할 수 있는 기능을 추가하였다.

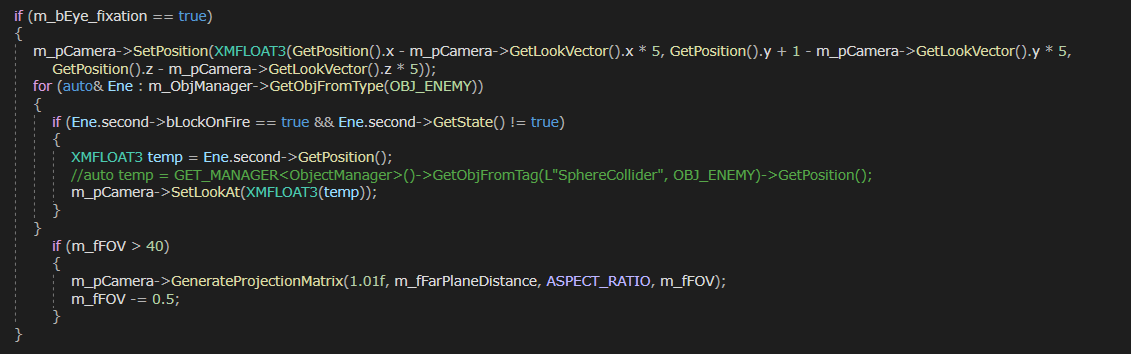


우선 해당 기능을 TAB키로 설정하였다.

TAB키를 누르면 m\_bEye\_fixation 이라는 bool형 변수가 이전 상태에 따라서 true 혹은 false가 된다.

False가 되면 후술할 수정된 카메라 위치와 바라볼 타겟, 그리고 줄어든 FOV를 원상태로 복귀시킨다.

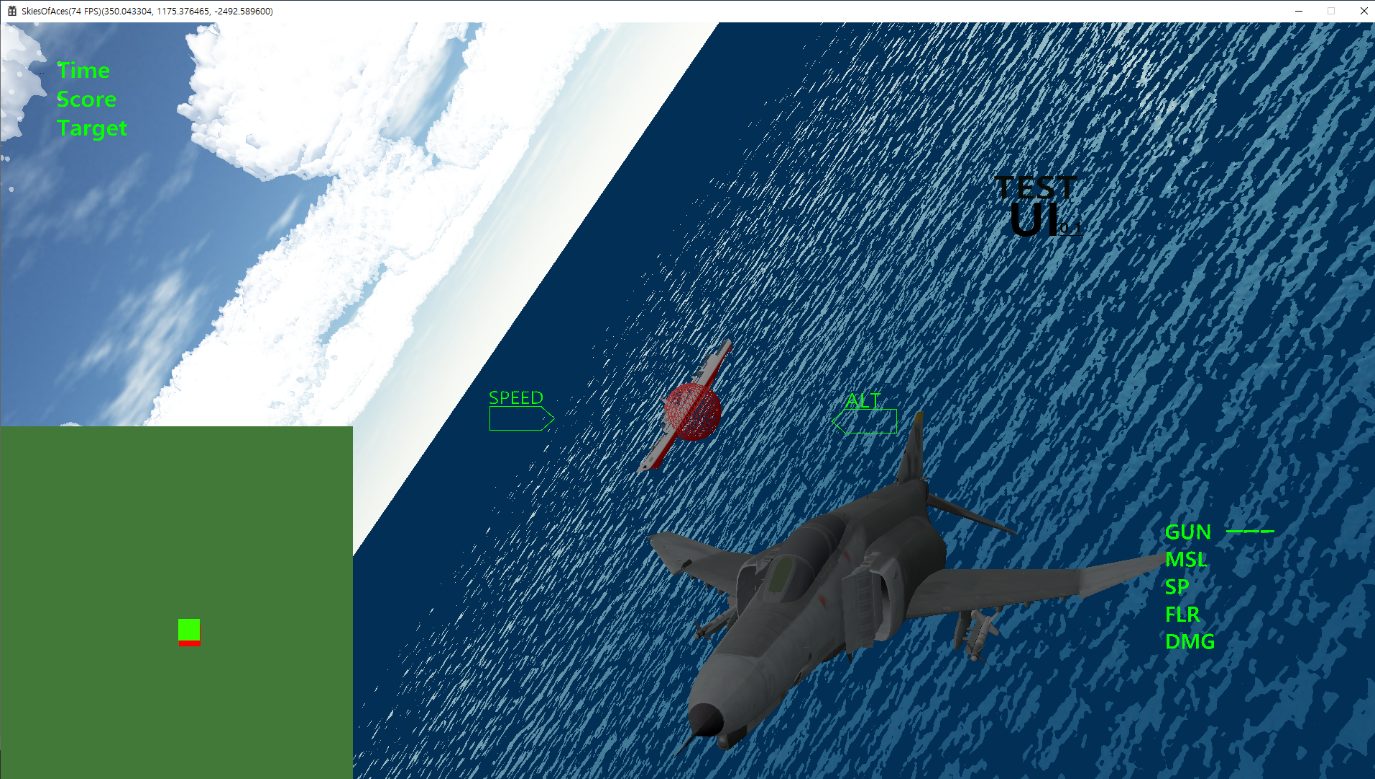
만약 true가 된다면 다음과 같은 코드를 통해 플레이어의 위치가 어디 있던간에 적을 바라보게 한다.



먼저 카메라의 위치(카메라 오프셋을 생각하면 된다.)를 플레이어와 타겟 둘 다 잘 보이는 위치로 재 설정한다.

그리고 락온 된 적을 바라보게끔 해당 타겟의 위치를 받아와 카메라의 SetLookAt 함수에 전달해 주어 카메라가 타겟을 바라보게끔 한다.

카메라는 FOV가 40이 될 때까지 서서히 줄어든다.



이로써 줌인 과 동시에 타겟을 바라보며 곡예를 할 수 있게 되었다.

**5. 구름 인스턴싱**

**빌보드로 된 수많은 구름을 특정 위치에 집중 배치 시켜서 뭉게구름을 만든다.**

우선 인스턴싱을 위한 구름 텍스쳐를 준비한다.

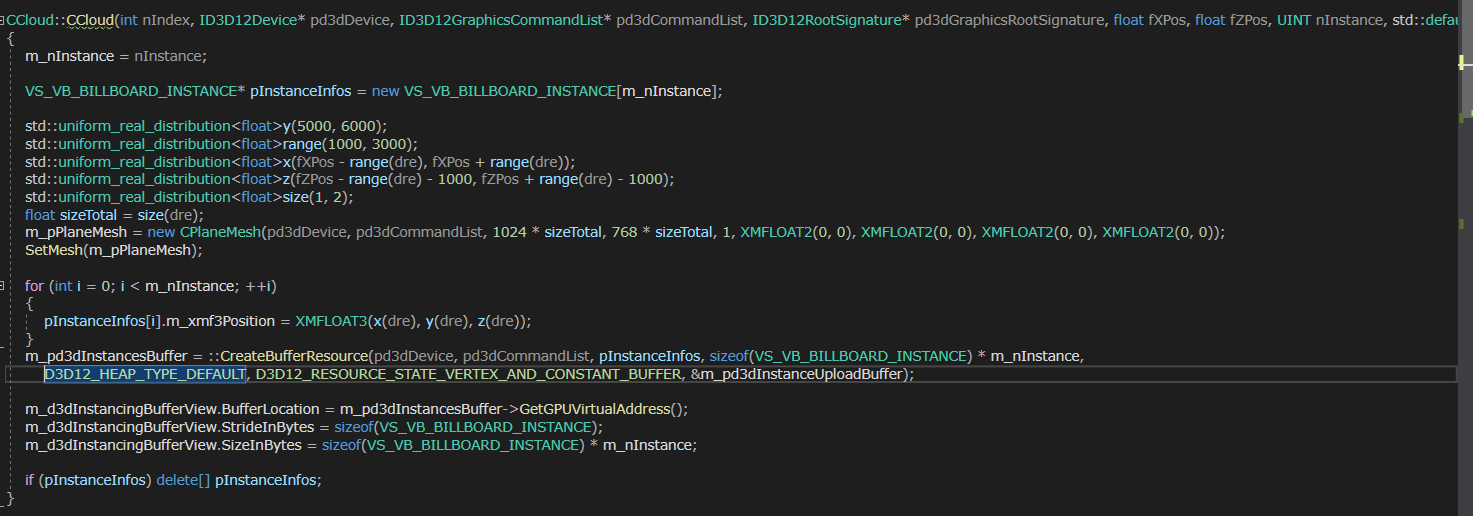
(cloud\_1.dds)

위와 같은 종류의 구름 텍스쳐 8장을 준비한다.

그리고 CCloud 클래스를 생성하고 텍스쳐를 불러오고 쉐이더 및 쉐이더 리소스 뷰를 생성한다.



모두 불러왔으면 이제 인스턴싱을 위한 생성자를 새로 작성한다.

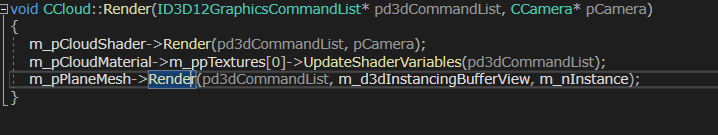


m\_nInstance에 매개변수로 받아온 인스턴싱 할 오브젝트 개수를 저장하고 랜덤엔진을 사용하여 구름이 생성될 위치와 범위를 지정한다.

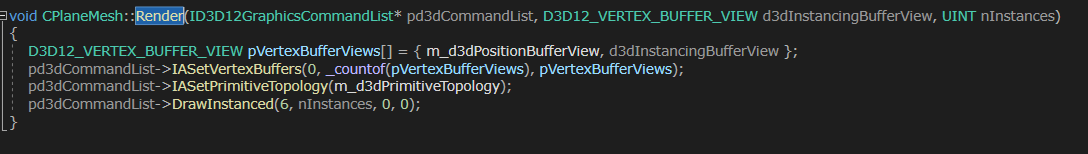
VS\_VB\_BILLBOARD\_INSTANCE 구조체 포인터 변수를 m\_nInstance만큼 동적할당하여 생성한후

그리고 구름 메쉬 크기를 지정하여 생성한 후 쉐이더 코드에 넘겨줄 인스턴싱 포지션을 랜덤으로 지정하여 구조체 포인터 변수에 저장한다.

그리고 포지션 값을 가지고 있는 구조체인 VS\_VB\_BILLBOARD\_INSTANCE 구조체와 구조체 포인터 변수를 가지고 인스턴스 버퍼와 인스턴스 버퍼 뷰를 생성한다.



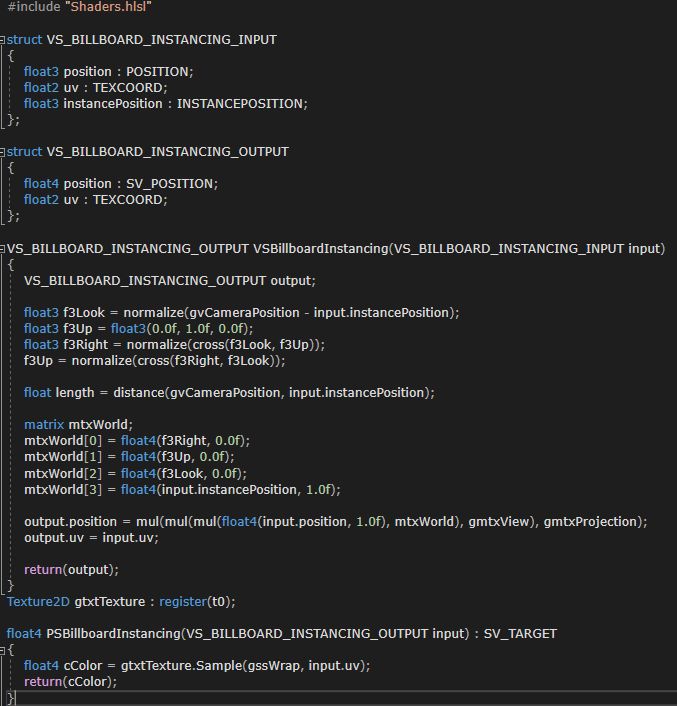
생성된 인스턴스 버퍼 뷰와 인스턴싱 될 개수를 m\_pPlaneMesh의 렌더의 매개변수로 넣어주고



메쉬를 만들 때 같이 만들었던 포지션 버퍼 뷰와 매개변수로 받아온 인스턴싱 버퍼 뷰를 가지고

입력 조립기 단계에 버퍼를 물려 준 후 DrawInstacned() 함수를 통해 인스턴스를 생성할 개수를 가지고 Draw한다.

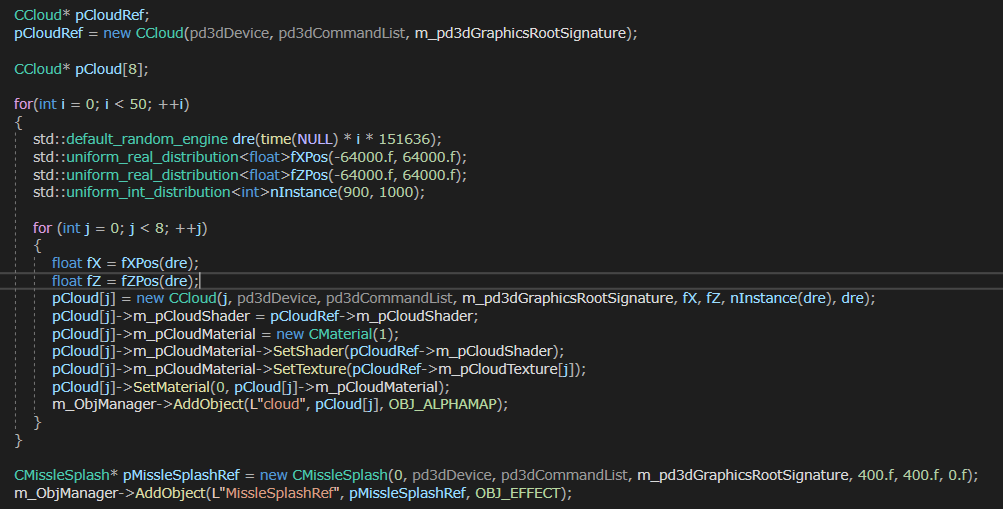
구름의 쉐이더 코드는 다음과 같다.



구름은 상, 하, 좌, 우 모든 방향에서 플레이어를 바라보아야 함으로 Up벡터를 새로 계산해 준다.

포지션은 인스턴싱 정보를 저장했던 인스턴스 포지션을 가지고 지정한다.

실질적으로 CCloud를 활용하여 구름을 생성을 호출하는 것은 CTestScene의 BuildScene()함수에서 실행한다.

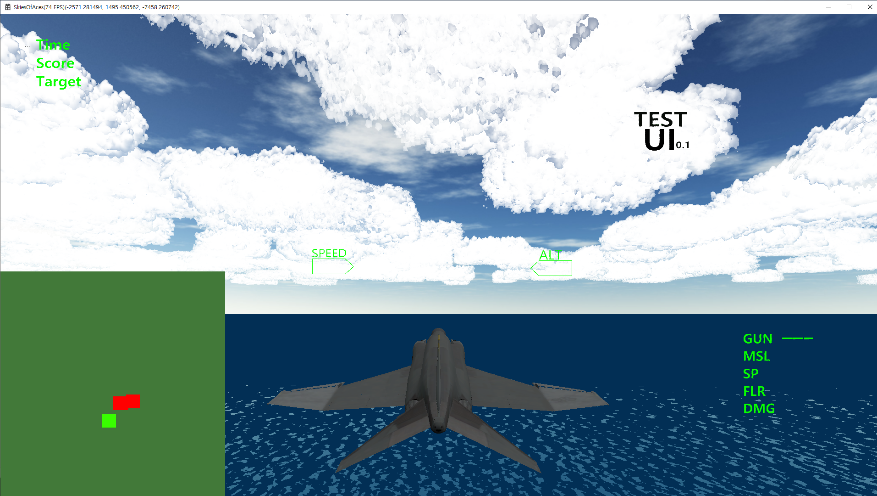


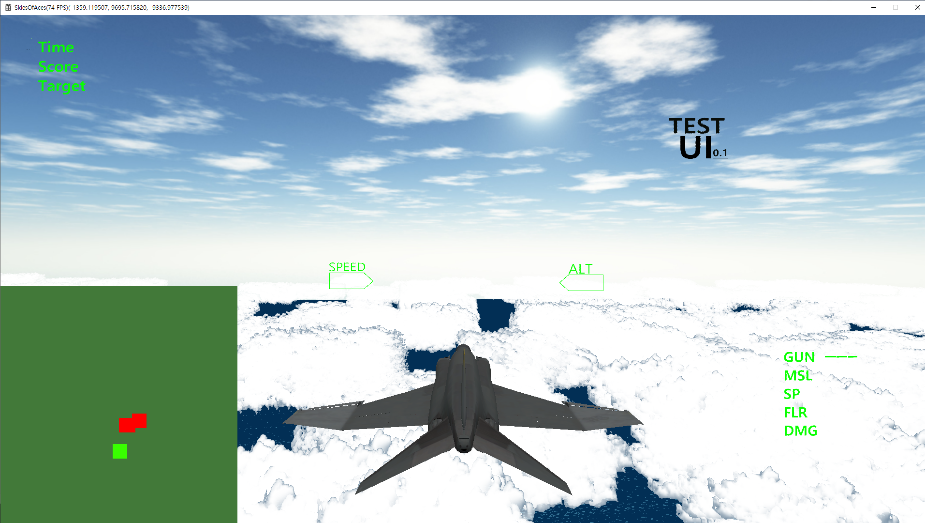
pCloudRef 변수는 텍스쳐 및 쉐이더, 쉐이더 리소스 뷰를 만드는데 쓰이는 래퍼런스 변수이고

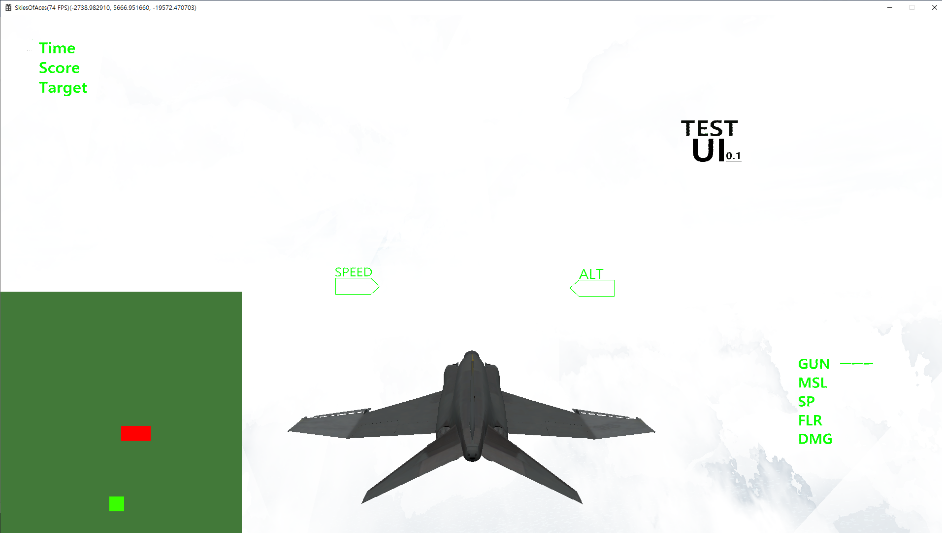
해당 변수의 정보를 사용하여 for문을 통해 pCloud변수로 실제 구름을 생성한다.

이중 포문 중 하위 포문을 돌때마다 j번째 텍스쳐를 사용한 구름 빌보드가 지정한 범위에 랜덤하게 900~1000개가 생성되게끔 만들었다.

이로써 최소 360,000개에서 최대 400,000개의 구름이 랜덤하게 생성된다.

구름 아래

구름 위

구름 속

**6. 파일매니저 작성 및 미사일 폭발 효과**

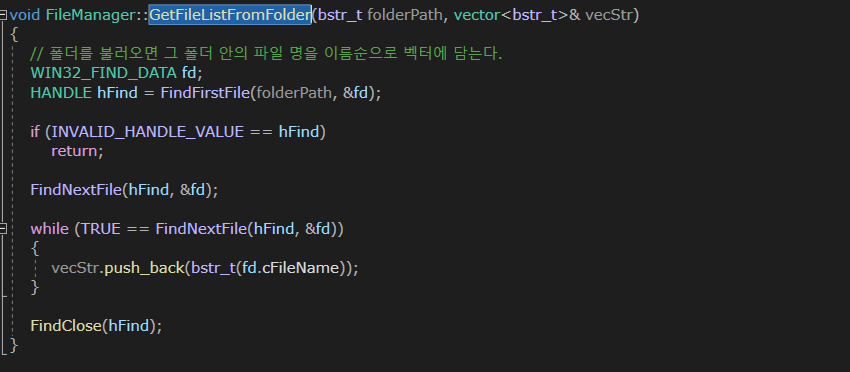
**미사일 폭발 효과를 만드는데 30장 이상의 텍스처를 사용된다.**

**빌보드 애니메이션을 만들 때 텍스처 경로를 일일 히 입력하는 것이 번거로움으로 폴더 내 파일을 한 번에 읽어오는 파일 매니저와 파일 매니저를 사용하여 텍스처를 로드 하는 매니저 클래스를 만든 후 미사일 폭발 효과를 만든다.**

우선 폴더 내 파일들을 이름 순으로 벡터에 담는 함수를 가진 파일 매니저 클래스를 작성한다.

WinAPI의 함수를 사용하여 별도의 순서를 직접 작성하지 않아도 된다.

파일의 순서의 마지막이 존재하지 않을 때 까지 반복문을 통해 벡터에 정보를 담는다.



그리고 폴더의 경로를 매개변수로 받아 파일 매니저에서 불러왔던 파일 목록과 폴더 경로를 합쳐 텍스처를 로드 하는 로드 텍스처 매니저 클래스를 작성한다.

해당 파일이 몇 개인지 센 만큼 그 센 수도 리턴해준다.

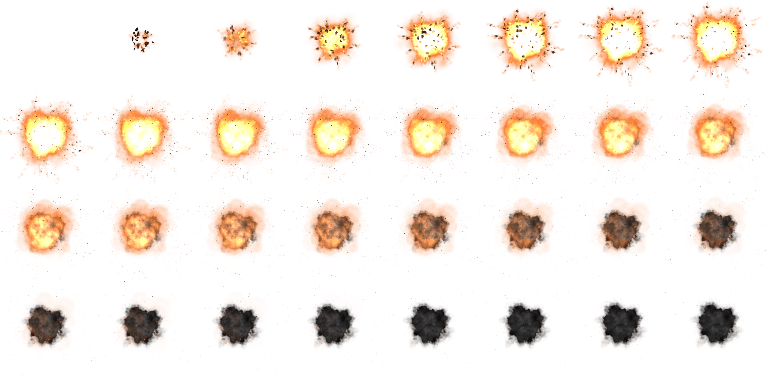


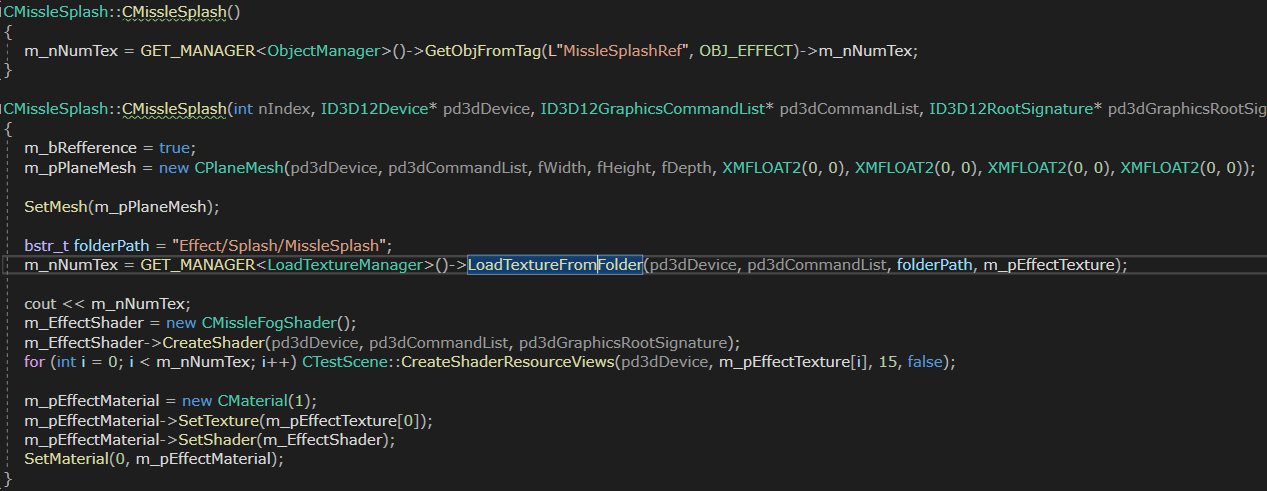
이로써 앞으로는 폴더내에 여러 장의 dds 파일이 있으면 다음 코드를 통해 한번에 텍스처를

로드 할 수 있다.



파일을 한번에 불러오는 작업이 가능함으로 이제 아래의 스프라이트를 각 이미지 별로 분리하여 dds로 만들어 텍스처를 생성한후 불러온다.





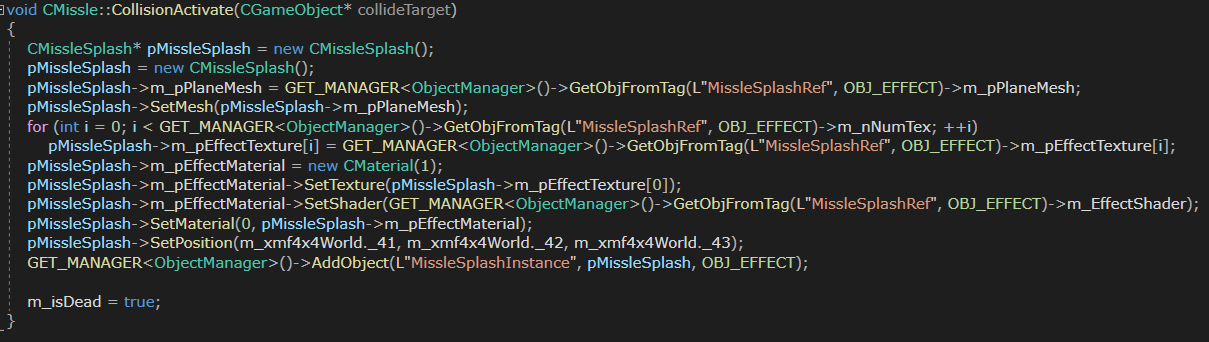
m\_nNumTex는 스프라이트 개수다. 텍스처를 불러올 때 리턴 받는 텍스처 개수를 저장한다.

기본 생성자는 실질적으로 생성되는 미사일 폭발 효과의 생성자로 쓰이며

아래의 생성자는 래퍼런스 오브젝트를 생성할 때 쓰인다.

구조는 MissleFog와 같다.

빌보드이며 스프라이트 애니메이션이 끝나면 소멸한다.



미사일 폭발 효과의 생성은 충돌 되었을 경우 실행되는 함수인 CollisionActive함수에서 호출된다.

미사일이 어딘 가에 충돌했을 경우 생성되는 것임으로 CMissle 클래스의 CollisionActive에서 실행된다.



|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** | 플레이어가 먼 거리에서 미리 조준할 타겟을 지정하고 가까운 거리에 왔을 때 자동으로 락 온 되는 기능이 없음  아직 적 오브젝트의 행위가 준비되어 있지 않음 | | |
| **해결방안** | 락 온 UI가 깜빡거리는 애니메이션을 이용해 먼 거리에서도 미리 타겟을 지정하고 가까운 거리에 왔을 때 빨간 조준 가능 표시가 뜨도록 구현  기본적인 AI를 먼저 제작하거나, 행동트리를 우선적으로 수행 | | |
| **다음주차** | 18 주차 | **다음기간** | 2020.04.27 ~ 2020.05.03 |
| **다음주 할일** | 김령운 : 타겟 애니메이션, 고도/속도/시간/점수 UI 애니메이션  이재원 : Behavior 툴 제작, 기타 자잘한 작업 해결 | | |
| **지도 교수**  **Comment** |  | | |