1. 이번 과제의 목표

저는 이번 과제에 대해서 다이렉트 3D와 특히, DirectMath에 익숙해지고, 교수님께서 제공하신 프레임워크를 온전하게 이해하는 것을 목표로 잡았습니다.

1. 프로그램 가정

저는 이 프로그램을 먼저 비행기가 날아가는 것 이기 때문에 최대한 제한이 없어야 한다고 생각하여 전진 후진에 대한 제한을 두지 않았습니다. 그리고 그것을 위하여 플레이어가 진행하는 방향에서 장애물 오브젝트가 생성되도록 하였습니다.

1. 프로그램 제작 순서

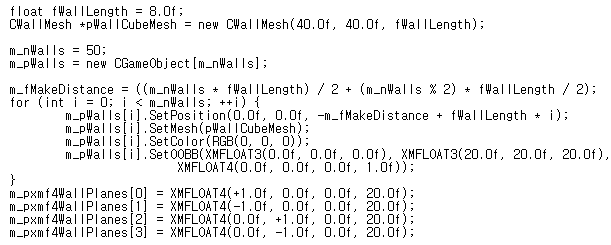
저는 프로그램의 조건을 만족시키기 위해서 다음과 같은 순서로 진행 하였습니다.

1. 사각형으로 표현된 무한한 공간 만들기
2. 플레이어가 무한한 사각형 밖으로 나가지 않도록 하기
3. 장애물의 방향 종류를 상하, 좌우, 랜덤으로 나누어 생성하기
4. 장애물을 플레이어가 움직이는 것에 따라 무한히 생성하기
5. 장애물이 부딪혔을 때, 방향과 색상 처리
6. 플레이어 조작 방법 개선
7. 총알 발사 및 총알과 벽, 장애물과의 충돌 처리
8. 플레이어와 장애물과의 충돌처리
9. 오브젝트의 파괴 시 파티클 생성
10. 코드 설명

-최대한 간결하게 작성하기 위하여 원본 코드에서 바뀌지 않은 내용은 적지 않았습니다.

1. 사각형으로 표현된 무한한 공간 만들기

* 먼저 벽을 생성할 때 플레이어 앞, 뒤로 벽이 만들어지게 하기 위하여 BuildObject()를 수정하였습니다.



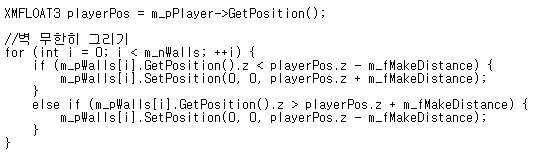
위의 코드에서 변수 fWallLength는 벽의 길이 조절을 쉽게 하기 위하여 추가하였으며,

m\_nWalls는 벽의 개수를 변경할 경우 전체 코드에 반영하기 위하여 추가하였습니다.

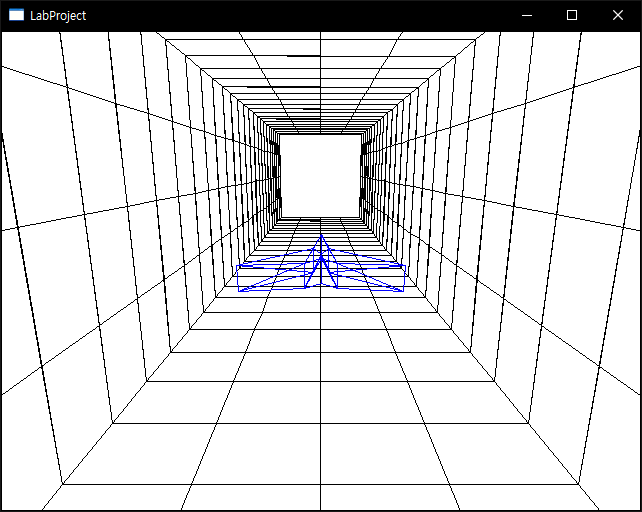
m\_fMakeDistance 벽의 m\_nWalls개 있을 때, 플레이어가 중앙에 위치할 수 있는 거리를 계산하였습니다. 이 변수는 플레이어를 기준으로 가장 멀리 떨어질 수 있는 벽과 플레이어 사이의 거리를 나타냅니다.

m\_pxmf4WallPlanes의 경우 기존 6개에서 4개로 감소시켰는데, 이것은 벽이 Z축으로 무한하게 생성되어야 하기 때문에 z축 충돌을 없애기 위하여 수정하였습니다.

* 플레이어가 이동할 시 벽이 무한히 보이는 효과를 위하여 ProcessInput()을 수정하였습니다.



플레이어와 벽사이의 거리가 m\_fMakeDistance보다 멀어진 경우 그 벽의 위치를 플레이어 위치 에서 진행방향으로 m\_fMakeDistance만큼 떨어진 위치로 재 설정 합니다. 이것으로 인하여 플레이어가 이동해도 벽이 무한히 존재하는 효과를 적용할 수 있었습니다.



무한한 공간 생성 완료

1. 플레이어가 무한한 사각형 밖으로 나가지 않도록 하기

* BuildObject() 에서 플레이어에 OOBB를 추가하였습니다.
* ProcessInput()에서 벽과 플레이어의 충돌을 추가하였습니다.

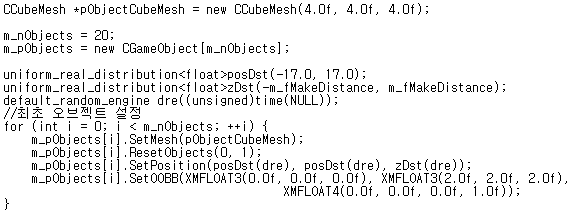


충돌 검사에서 벽의 Z축의 검사를 제외하여 모든 벽을 따로 검사하는 대신 하나의 벽의 OOBB검사로 모든 벽의 충돌을 확인할 수 있습니다.

충돌이 된 경우 어느 면에 충돌된 것인지 확인하여 그 면의 노멀 벡터를 얻어옵니다. 그리고 그 노멀 벡터에 현재 플레이어의 속도를 투영하여 벽에서 반사되는 벡터를 구하고 그 벡터의 크기를 length로 불러와 플레이어를 노멀 벡터 방향으로 이동시켜 벽을 통과하지 못하도록 하였습니다.

1. 장애물의 방향 종류를 상하, 좌우, 랜덤으로 나누어 처리하기

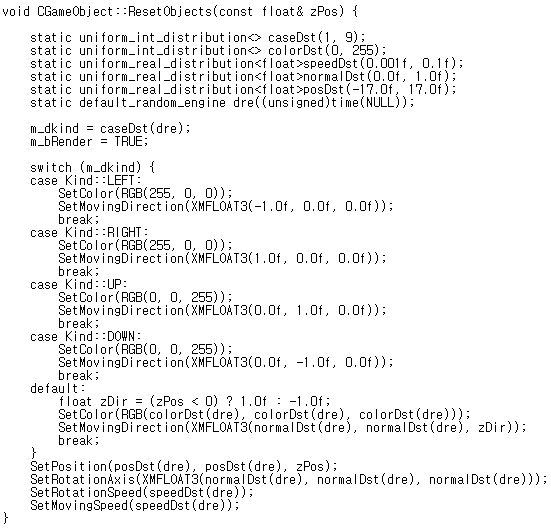
* BuildObject()에서 오브젝트의 최초 상태를 for문으로 처리하였으며, 랜덤하게 위치하도록 하기 위하여 random을 이용하였습니다.



벽의 생성과 마찬가지로 장애물의 효율적 관리를 위하여 m\_nObjects를 추가하였습니다.

1. 장애물을 플레이어가 움직이는 것에 따라 무한히 생성하기

* 장애물 또한 무한히 생성되는 효과를 주기 위하여 오브젝트에 멤버 함수로 ResetObjects()를 추가하였습니다.



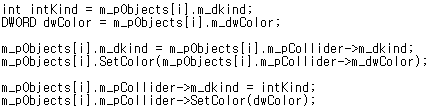
랜덤 엔진을 static으로 선언하여 시간이 경과함에 따라 랜덤으로 나오는 값이 바뀌도록 하였습니다.

플레이어의 방향에 따라 zPos는 (+, -)m\_fMakeDistance를 받아오며, RANDOM 방향의 경우 플레이어 쪽으로 z값을 설정하기 위하여 zPos의 부호를 조사하여 z값을 설정하였습니다.

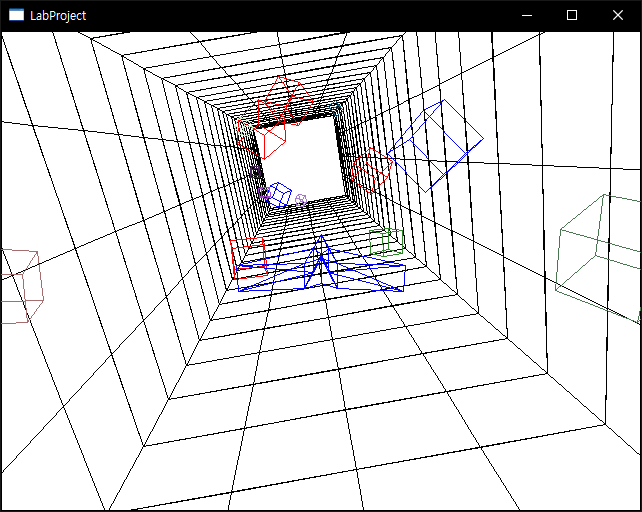
* ProcessInput()에서 벽과 마찬가지로 플레이어와 장애물의 거리가 m\_fMakeDistance보다 커진 경우 ResetObject()로 재설정 합니다.

1. 장애물이 부딪혔을 때, 방향과 색상 처리

AnimateObjects()에서 장애물끼리 충돌했을 때 좌우, 상하의 경우 고정된 색을 가져야 하기 때문에 색상과 종류를 교환하는 코드를 추가하였습니다.



이 부분의 경우 교수님께서 제공하신 프레임 워크에서 충돌 처리가 두 물체가 부딪혔을 때 방향과 속도를 단순히 교환하는 방식으로 되어있다는 것을 발견하였고, 그에 맞춰 색상과 종류 또한 바꿔주면 되겠다는 생각으로 진행하였습니다.



장애물 설정 완료

1. 플레이어 조작 방법 개선

기본적으로, 비행기 게임이기 때문에 방향키로 조작하기에는 굉장히 불편하다고 느꼈으며, 최근 인터페이스 강의에서 UI가 중요하다는 강조가 있어서 바꾸게 되었습니다.

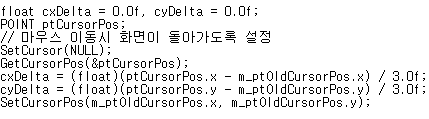
- 키보드

기본 FPS에서 사용하던 방식을 가져와 WASD로 기본 이동을 하고 Q와 E를 각각 위와 아래 이동으로 변경하였습니다. 이때, GetKeyState()를 사용하여 비트계산으로 처리하였습니다.

-마우스

화면의 마우스를 이용한 전환 또한 클릭하여 이동하는 대신 마우스 이동을 읽어와서 처리하는 방식으로 변경하였습니다.

* CGameFramework() 생성자에 m\_ptOldCursorPos를 화면 중앙으로 변경하였습니다. 이 값이 기존처럼 (0, 0) 인 경우 마우스의 좌, 상 이동이 먹히지 않는 문제가 있었습니다.



마우스의 경우 기존에 SetCapture()가 윈도우로 잡혔을 경우에만 마우스 회전처리를 하던 것을 if문을 제거하며 제한을 없애 항상 마우스의 이동회전 처리를 하도록 변경하였습니다.

추가적으로 좌, 우 클릭으로 회전을 구분하였던 것을 없앴으며, 그로 인해 비행기 게임이라는 컨셉에 더 어울리는 화면 움직임을 얻었습니다.

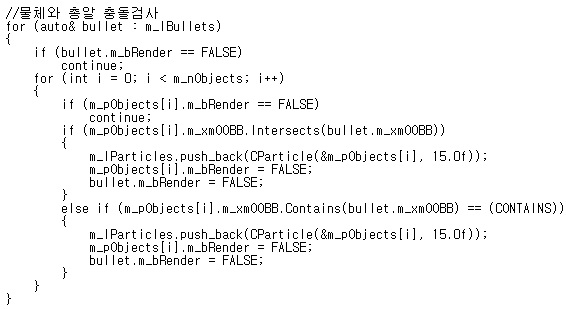
1. 총알 발사 및 총알과 벽, 장애물과의 충돌처리

* ProcessInput()에서 총알 발사 처리의 경우 마우스 회전을 클릭으로 처리하지 않게 되었기 때문에 남는 좌 클릭 버튼을 이용하였습니다. GetAsyncKeyState()를 이용하여 좌클릭 이벤트를 받았으며, 연속해서 총알이 나가지 않도록 ShootRate를 설정하였습니다.

생성된 총알 객체는 동적 컨테이너 중 중간 데이터의 삭제가 다른 자료구조에 비해 덜 부담스럽기 때문에 리스트를 사용하여 저장하였습니다.

* AnimateObjects()에서 총알의 업데이트 및 충돌처리를 하였습니다.

= 벽과 의 충돌 처리의 경우는 장애물과 벽의 충돌과 동일한 방식으로 처리하였고, 충돌한 총알을 제거하기 위해 m\_bRender를 False로 만들었습니다.



= 장애물과 총알의 처리는 총알이 장애물 내부에 들어간 경우와 장애물에 걸친 경우로 가정하여 진행하였으며, 총알이 장애물에 충돌한 경우 장애물과 총알 모두 m\_bRender를 False로 만들었습니다.

= 총알 또한 플레이어와의 거리가 m\_fMakeDistance보다 큰 경우 m\_bRender를 False로 변경하였습니다.

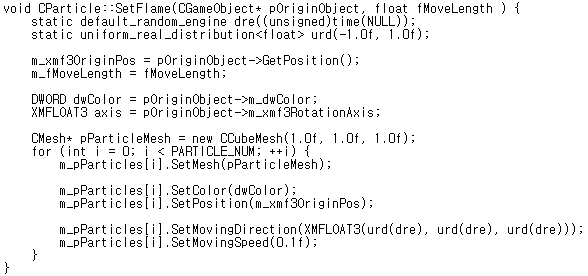
= 총알의 제거는 리스트의 기능 중 remove\_if()를 사용하여 구현하였으며, m\_bRender가 False인 것을 제거하였습니다.

1. 플레이어와 장애물과의 충돌처리

-AnimateObjects()에서 장애물 배열을 for문으로 돌리며 각각 충돌 처리를 하였습니다. 이때, m\_bRender가 False인 장애물은 제외하여 충돌이 일어나지 않도록 하였습니다.

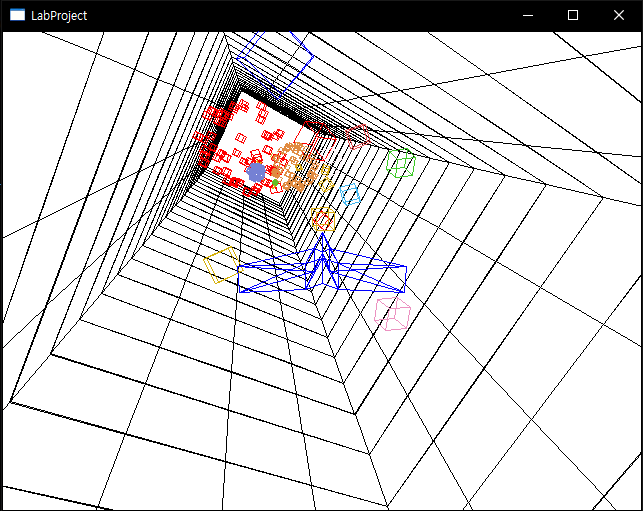
1. 오브젝트의 파괴 시 파티클 생성

-50개 이상의 박스로 구성된 파티클을 구현하기 위하여 CParticle 클래스를 새롭게 만들었습니다. 이때 멤버 변수에 CGameObject \*m\_pParticles로 동적 배열을 할당하여 원하는 숫자의 파티클을 생성할 수 있도록 하였습니다. 또한, 이동 생성자로 구현하여 리스트에 삽입할 때 복사가 일어나 속도가 느려 지지 않도록 하였습니다.



파티클이 터지도록 하는 SetFlame()의 경우 기존 도형의 정보를 받아오고 그 중 색상과 위치를 현재 파티클 그룹에 적용하고 랜덤 엔진으로 방향을 설정하였습니다. 매개변수 fMoveLength는 파티클이 살아있을 수 있는 거리를 적용하는 것인데, 이것은 플레이어가 죽을 경우 다른 파티클과 차별을 주기 위해 더 긴 시간동안 유지시킬 수 있도록 함수의 입력 값으로 설정한 것입니다.

파티클 생성의 경우 기존에 충돌처리 후 장애물 혹은 플레이어의 m\_bRender를 False로 설정한 위치에서 리스트에 추가하였으며, AnimatieObjects()에서 Update하며 최대 생존 거리보다 많은 거리를 이동한 경우 파티클의 m\_bRender를 False로 하여 총알의 제거처럼 remove\_if()를 사용하여 제거하였습니다.



최종 실행 화면

1. 조작 방법

