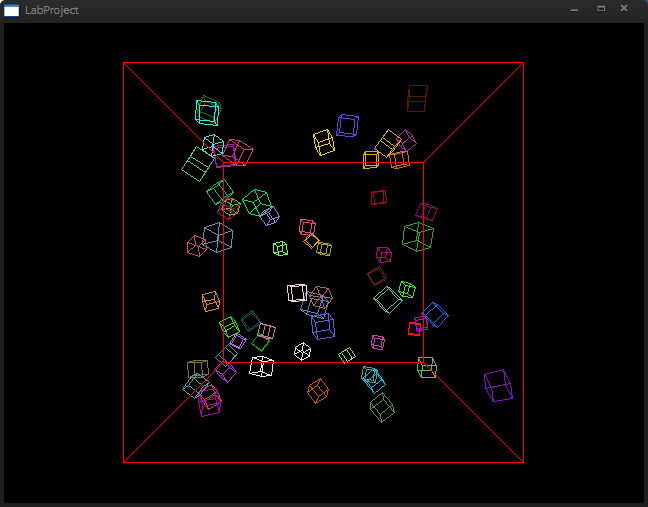
게임공학과 2012180004 권창현



1) 과제 완성 사진 및 과제를 완료하지 못한 부분

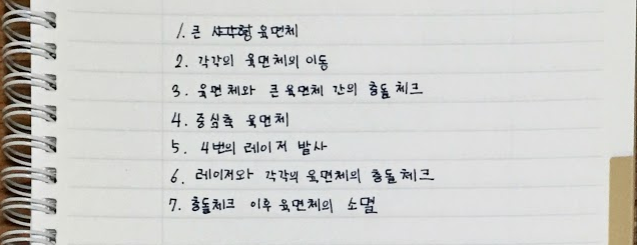
과제를 최종적으로 완료하면서 완료 하지 못한 부분은 레이저를 만들어 z축으로 회전하면서 쏘면 제일 먼저 충돌한 큐브가 사라지는 부분을 완성하지 못하였다.

2) 프로그래밍 과정

기본 파이프 라인에서 육면체의 크기를 줄이기 위하여 프로젝트의 cpp 파일들을 확인 하면서 육면체의 크기를 줄이기 위해 찾던 중 GameObject 에 CCubeMesh 라는 부분을 찾아서 숫자를 2에서 0.5로 변경을 하였다. 적당한 크기를 확인하였다.

그 다음 부분으로는 육면체의 개수를 50개 이상으로 늘리는 방법을 생각하였는데, CCubeMesh 에서는 도형 한 개를 그릴 수 있는 코드 라 서 아무래도 다른 부분에서 육면체의 개수를 늘리고 있는 것이 생각하여 다른 cpp를 확인하는 중 GameFramework 에서 m\_nObjects 라는 것을 확인하고 숫자를 수정하였지만 코드 자체에서는 배열을 직접 하나하나 입력을 해주는 것을 보고선 for문을 통하여 배열을 자동으로 생성을 하게 해주었다. 그 뒤 random 헤더를 불러와 랜덤 엔진을 사용하여 육면체의 위치를 여러 위치에 랜덤 하게 나타나게 한 뒤 배경 색상, 육면체 색상을 수정을 하였다.

과제를 진행하면서 어떤 순서대로 과제를 진행 해야지 좀 더 효율적인 코드를 작성하기를 생각하다가 과제의 내용을 생각하며 내가 지금 당장 해야하는 부분과 나중에 해야 되는 부분을 생각하면서 작성을 하였다.



사진과 같이 가장 우선적으로 해야할 부분부터 마지막 하는 부분까지를 노트에 작성을 하여 이대로 실천을 하기로 하였다. 우선 가장 첫번째인 큰 육면체를 만들기 위하여 육면체의 크기를 조정한 부분에서 크기를 수정을 해보려 하였으나, 이렇게 될 경우 따로 변수를 받아서 크기 값을 가지고 있어야하는 불편함이 생겨서 클래스 하나를 더 만들어서 따로 클래스를 불러오게 하였다.

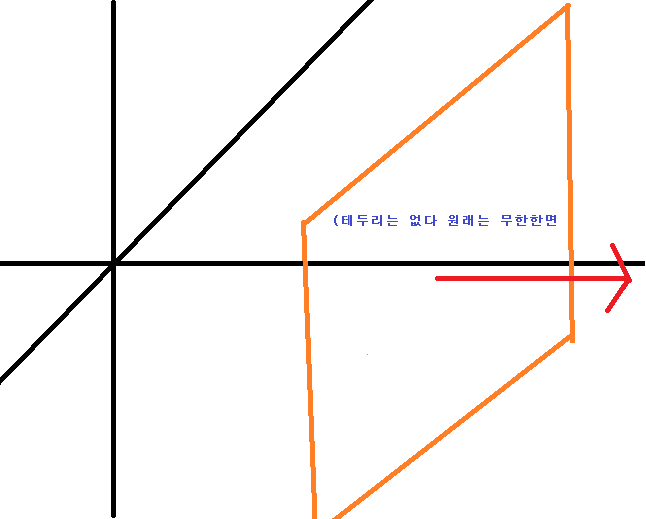
기존 육면체 작성과 동일하게 변수 명을 다르게 하여 BigCubeMesh를 추가하였고, GameFramework 에다가 BigCubeMesh를 불러올 수 있게 추가를 하였다. 그 뒤 각각의 육면체의 이동을 시작하기 이전에 50개 이상의 육면체를 움직이게 하면 확인도 하기 힘들 뿐 아니라 전부 확인을 할 수 없을 꺼 같아서 육면체를 1개로만 작업을 하였다.

교수님의 요구사항 중 행렬을 이용하여 코드를 완성하여라 라는 조건이 있었지만 처음 다이렉트X를 코딩하면서 행렬에 대한 이해도가 부족하여 기존 파이프라인의 변수를 그대로 사용하여 두번째인 각각의 육면체 이동을 시작하였다. 처음 육면체를 이동하는 것은 정말 간단하게 CGame Object 안의 변수인 m\_fxRotation, m\_fyRotation, m\_fzRotation에 더하기를 위하여 어디로 움직이게 할지 방향을 지정해주는 m\_vMovingDirection 함수를 만든 뒤 랜덤 엔진을 이용하여 방향을 지정해주고 해당 값과 움직이는 변수를 더하여 움직이게 하였다.

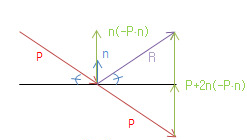
정상적으로 움직이는 것을 확인한 뒤 이제 세번째 인 육면체와 큰 육면체 간의 충돌 체크를 해야 하는데 지금까지 2D등 여러 프로그래밍 에서 충돌 체크를 할 경우 충돌 체크 위치까지의 x값 혹은 y값에 범위를 안 다음 if문을 통하여 그 범위에 도달하면 충돌 체크가 되었으니 다른 결과값을 진행하라 라는 방식을 사용했다.

이번 프로그래밍에서는 위와 같은 방법을 사용하지 말라 하여서 여러가지를 찾아보는 도중 육면체 간의 AABB 충돌 체크 와 같은 방법을 알게 되었으나, 전혀 사용방법을 알 수가 없어서 수업시간 끝나기 전에 교수님이 말씀한 법선 벡터, 반사 벡터를 이용하여 충돌 체크를 하라는 기억이 나서 해당 부분을 찾다 보니 3개의 점을 입력하여 평면을 만든 뒤 법선 과 포인트를 입력하여 플레인 을 생성하고 면과 위치 벡터 와 의 내적을 구하여 육면체가 입력된 평면보다 안에 있거나 같은 위치에 있거나 밖에 있는지를 판단할 수 있는 방법을 찾았다. 찾은 방식으로 적용을 하려면 일단 도형이 벡터로 저장이 되어있어야 좀더 편하게 작업을 할 수 있을까 라는 생각이 들어서 가장 우선적으로 SetPosition 함수의 받는 방식을 벡터로 받을 수 있게 x, y, z에 대한 각각의 변수를 제거하고 벡터로 변수를 받을 수 있게 수정을 하였다.

해당 부분만 수정을 하면 정상적으로 벡터로 받고 움직임 부분부터 다 정상적으로 될 줄 알았지만, 컴파일을 하였더니 오류가 나타났다. GameObject의 Draw 부분에서 float으로 받던 부분이 벡터로 변경하면서 변수를 지워버려 컴파일에서 오류가 난 것이다. 벡터를 불러오는 방법을 확인한 뒤 Draw 부분에 육면체의 float이 자리잡았던 부분을 전부 벡터 값으로 전환을 한 뒤 충돌 체크를 시작하였다. 육면체 한 개를 오른쪽 면과 충돌 시에 대한 부분을 작성을 하기 위하여 충돌 체크 부분에 대한 함수를 작성하였다. 처음 3개의 점을 이용한 평면을 생성하기 위하여 D3DXPlaneFromPoints 라는 다이렉트X의 함수를 이용하였다.



D3DXPlaneFromPoints 함수를 이용하면 위의 사진과 같이 임의의 평면을 생성하게 된다. 그 뒤 법 선과 포인트를 이용하여 플레인 을 생성해준다 그 뒤 면과 위치 벡터와의 내적을 구하기 위하여 현제 육면체의 x, y, z 값을 넣어준 뒤, 그 벡터를 D3DXPlaneDotCoord 에 플레인 과 함께 넣어주면 벡터의 내적을 계산하여 반환을 해준다. 그 반환을 float fResult로 받아와 현재 위치가 입력된 벽면과 얼마나 거리가 떨어져 있는지를 반환해준다. 그 반환 값을 바탕으로 이제 충돌이 되었는지 안되었는지 확인이 가능하다는 것을 확인 뒤 충돌을 하였을 경우 들어온 위치만큼 튕겨 나가게 하는 것을 코딩 하기 위하여 반사 벡터의 공식을 찾았다.

인터넷을 찾아본 결과 왼쪽과 같은 이미지가 나타나며, 공식은 “R = P + 2n(-P \* n)” 라는 공식을 찾아냈다 이 공식을 대입 하려 하다 보니 “R = 움직이는 방향 -2 \*(내적\*벡터)” 라는 방식이 나와서 공식에 맞게 대입을 한 후 코드를 돌려보니 정말 그대로 반사가 되어서 튕겨져 나온다. 오른쪽 한 면만 작업을 한 것이기에 육면체 모두를 똑 같은 방식으로 코드를 작성하여 함수가 총 6개가 되었다.

충돌 체크를 완료한 뒤 이제 Draw부분에 대해서 행렬로 변환을 시작하였다. 지금까지 행렬을 사용하지 않고 교수님께서 제공해주신 파이프 라인에 육면체와, 충돌 체크 부분만 행렬을 사용하여 값을 넣어주어 정상적으로 작동하게 만들었으니, 본격적으로 행렬 작업을 시작하였다.

처음 시작은 모든 CGameObject의 변수들을 전부 벡터로 변경하여, Draw시 행렬로 받을 수 있게 작업을 하는 것 이다. 처음부터 행렬로 받으면 편하긴 하지만 바로 할 자신이 없어서 우선 가장 만만한 벡터로 값을 받아 놓는 것이다. 변수를 수정하면서 CGameObject 와 CCamera의 변수 명이 동일한 것을 보고선 변경을 해주었다. 변수 명이 같아도 클래스가 다르기 때문에 상관은 없지만 코드를 작성하면서 변수가 같으면 조금 헷갈릴 꺼 같아서 CCamera 같은 경우 \_뒤 c를 붙여서 카메라 라는 것을 알게 하였고 CGameObject 같은 경우 기존 float 형의 f를 그냥 그대로 두어 변수를 수정하였다. 그 뒤 GameObject 에서 Draw 부분을 행렬로 변환 하는 작업을 시작하였다.

각각의 Rotation, Scale 에 대한 Matrix 행렬을 만들어 D3DXMatrixIdentity 사용하여 모두 단위 행렬로 변경하였다. 단위 행렬로 변경을 하지 않고 그냥 사용 할 경우 쓰레기 값이 이미 들어있거나 할 경우를 대비 하여 전부 단위 행렬로 변경을 해주었다. 행렬을 넣어주면서 생각을 하게 된 것이 매번 교수님이 말씀하신 월드 좌표 라는 말이 생각이 나서 아무리 각각의 행렬을 다 만들어 줘도 월드 좌표계가 없으면 표시가 안되겠구나 라는 생각을 하고 mtxWorld 라는 월드 좌표 계를 만들어서 Scale 면 mtxWorld에 Scale을 할 값들을 넣어줘서 mtxWorld에 넣어주고 Rotation 같은 값들을 행렬에 넣어준 뒤 D3DXMatrixMultiply 행렬의 곱을 통하여 값을 적용 시켜주었다.

다 적용을 한 뒤 컴파일을 하였지만 정상적인 출력이 나오는 것이 아닌 화면이 아무것도 나오지 않았다 카메라 뷰에 대한 부분이 제대로 작동 하지 않는다는 것을 생각하여 여러 가지 방향으로 계속 수정을 하다가 CCamera에 m\_Viewport의 값이 너무 크거나 작아서 그려진 육면체가 이미 카메라 범위에서 벗어나거나 너무 안으로 들어와서 보이지 않았던 것이다. 해당 부분을 무한 대입 식으로 값을 수정하여 정상적으로 보이는 부분을 찾은 뒤 너무 멀어져 있어 보여 z값을 0.0f 가 아닌 좀더 앞으로 30.0f까지 당겨 왔다. 그 뒤 키보드를 통하여 좌, 우, 앞, 뒤를 이동하였는데 움직이는 방향이 정 반대로 움직이는 것이다. 아무래도 뷰 포트를 설정하면서 좌 우가 뒤집어서 출력이 된 것 같다. 컴퓨터 그래픽스에서 동일한 문제가 있었는데 이 부분은 z축으로 어느 한계 이상 당겨졌을 경우 반대로 출력이 된다는 것을 듣고선 수정을 하려 하였지만, 정확하게 어느 부분이 문제가 되는지를 몰라서 키보드의 값을 바꿔주는 것으로 수정을 완료하였다. 그 뒤 레이저 포인트를 이용한 육면체를 지우는 방법을 진행 하려 하였으나 아무리 생각을 하여도 좋은 방안이 떠오르지 않아 처음에는 육면체의 위치 좌표를 알고있으니 레이저가 쏘면 그 쏜 위치 좌표와 육면체의 위치 좌표가 동일 하면 충돌 체크 검사를 하지 않고선 해당 위치에 있는 육면체를 지우는 방법을 하려고 하였다. 하지만 해당 방법은 편법 임과 동시에 레이저가 지나갈 때 마다 육면체와의 위치를 모두 한번씩은 검사를 하게 되면 프레임레이트 가 매우 떨어지는 것을 생각하여 결국 생각만 하다가 코드 작성을 하지 못하고 과제를 미완성으로 제출을 하게 되었다.

3) 조작 방법

레이저 포인트를 사용하지 않기 때문에 파이프라인과 동일한 키보드 상, 하, 좌, 우 로 이동만 할 수 있다.