3D 프로그래밍1

과제 02

게임공학과

2013180003 김나단

1. 이번 과제의 목표

따라하기 15번의 지형이동 예제를 바탕으로 평지에 집을 만들고 내부를 미로로 만들고 내부에는 적이 존재하여 일정 거리 내에서 내를 향해 굴러오고, 픽킹을 사용하여 총알을 발사하고 총알에 맞은 적은 파티클을 남기고 사라지도록 한다.

저는 이번 과제에서 새롭게 추가된 따라하기의 픽킹과 이전 따라하기의 인스턴싱을 최대한 활용하는 것을 목표로 하였습니다.

1. 프로그램에 대한 가정

먼저, 저는 지난번과 동일하게 벽에 부딪히는 경우 단순히 멈추거나 위치를 초기화 하는게 아니라 벽 쪽으로 가는 속도만을 제거하여 자연스럽게 미끄러져 지나갈 수 있도록 하였으며, 이 과정에서 기존의 GetVelocity가 마찰력으로 인해 Velocity를 update 이후 0이되는 것을 발견하여 velocity를 저장할 새로운 변수를 만들어 사용하였으며 이로 자연스러운 이동이 가능하게 되었습니다.

또한, 벽과의 충돌처리를 할 때, 기존의 방법이 제대로 작동하지 않았기에 픽킹을 활용하여 벽의 위치와 플레이어 위치를 이용하여 월드공간에서 광선을 만들고 이 광선을 모델 좌표계로 옮겨 삼각형과 광선의 충돌검사를 하여 충돌하는 면을 알아내고 그 면의 법선 벡터를 리턴하여 충돌 처리에 활용하였습니다.

적 AI는 단순히 월드공간의 특정 축을 기준으로 회전하는 게 아니라 플레이어의 위치를 바탕으로 플레이어 쪽으로 굴러오는 모습을 나타내기 위하여 플레이어의 위치와 현재 적의 위치를 이용하여 플레이어 쪽으로의 방향 벡터를 구하고 이 방향벡터와 UP벡터인(0, 1,)을 외적 하여 나온 벡터를 회전 축으로 삼았습니다. 그러나 이 과정에서 월드 상의 회전이 이루어져 제대로 된 회전이 되지 않아 이 벡터를 월드변환의 역변환을 이용하여 로컬 좌표계로 표현하여 목표한 회전을 만들어 낼 수 있었습니다.

총알의 발사는 평소에 이루어지지 않으며 적 객체를 픽킹하였을 경우 총알을 발사하며 벽을 눌렀을 때 총알이 나가지 않도록 하기 위하여 벽을 포함하여 픽킹 범위를 정했으며 그 결과로 나온 오브젝트가 적 오브젝트가 아닌 경우 픽킹 결과를 버리는 방법으로 총알 발사를 제한하였습니다.

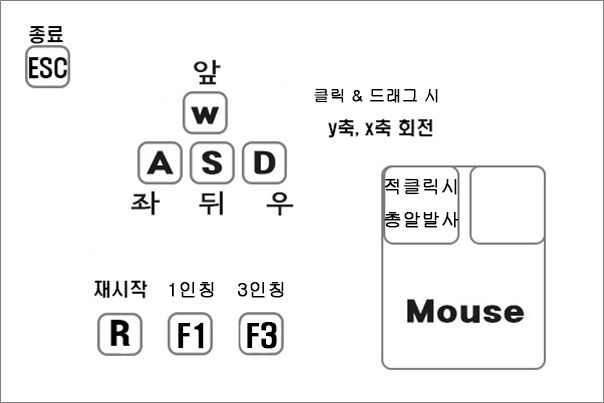
적 오브젝트를 성공적으로 선택한 경우엔 플레이어가 적 오브젝트 방향으로 돌아보고 플레이어의 방향으로 총알을 발사하게 됩니다. 플레이어가 적 오브젝트 방향으로 돌아보게 하기 위하여 현재 플레이어의 방향 벡터와 적과 플레이어의 위치를 이용하여 만든 방향 벡터의 내적의 결과를 acos에 넣어 각도를 구하고 스칼라 삼중적을 이용하여 어느 방향으로 회전할지를 결정하여 회전을 하였습니다.

P를 누른 경우 입구로 이동하며, 이 경우에도 플레이어가 입구 방향을 바라보도록 설정하기 위하여 위와 동일하게 처리하였습니다. 방향벡터와 목표한 방향 벡터가 동일하여 내적 결과가 nan인 경우 isnan함수를 통해 오류가 발생하지 않도록 처리하였습니다.

현재, 카메라와 벽의 충돌을 검사하지 않아 카메라가 벽 안으로 들어가는 경우가 발생하는데 이 경우에 적을 눌렀을 때에도 앞서 설명한 이유로 인해 픽킹이 되지 않기에 총알이 발사되지 않습니다. 이 경우 1인칭으로 진행하시면 됩니다.

시점의 경우 1인칭과 3인칭으로 구별하였으며, 사람이라고 가정되기 때문에 스페이스 쉽 시점은 선택할 수 없도록 하였습니다.

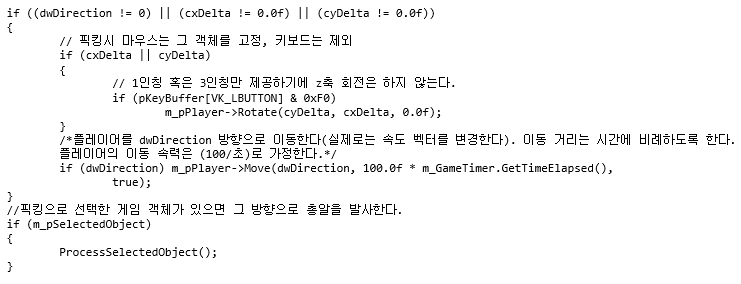
1. 조작 방법



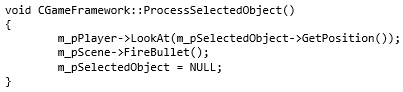
1. 코드 설명
2. GameFramework

OnProcessingKeyboardMessage()의 case VK\_F2를 삭제하여 스페이스 쉽 모드를 선택하지 못하게 하였습니다.

ProcessInput()에서 픽킹이 진행될 때 플레이어 조작이 안되던 것을 해결하기 위하여 픽킹 처리와 플레이어의 조작을 분리하였습니다.



ProcessSelectedObject()에서는 플레이어의 방향을 선택한 오브젝트를 바라보도록 변경하고 총알을 발사하도록 하였습니다. FireBullet()은 내부적으로 BulletShader의 SetBullet을 호출합니다.



1. GameObject

각각 적, 총알, 파티클을 나타내기 위한 CEnemyObject, CBulletObject, CParticleObject를 추가하였습니다.

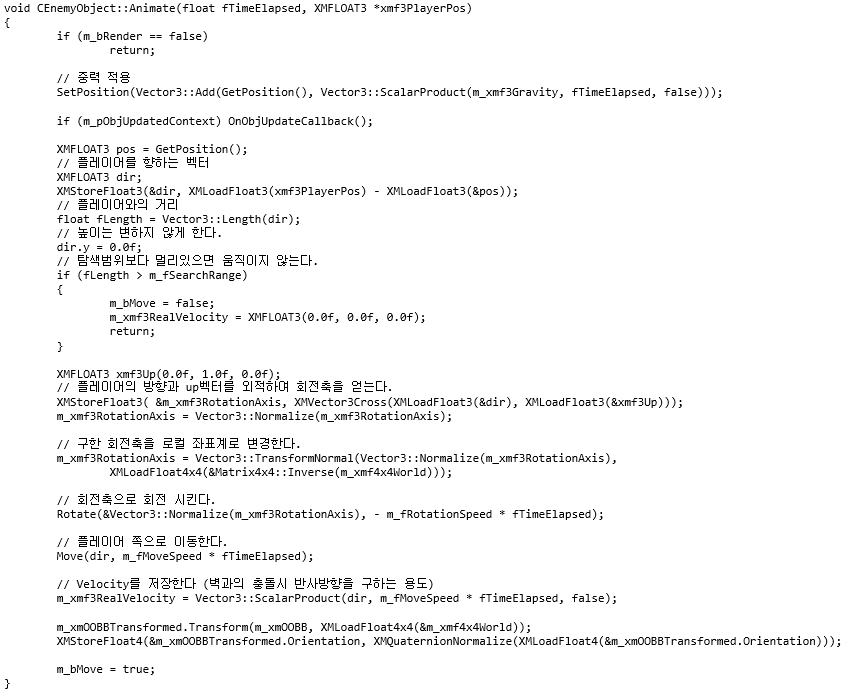
이중 ParticleObject와 BulletObject는 기존에 사용한 것과 동일하여 설명을 하지 않겠습니다.

EnemyObject는 기본적으로 지형을 이동할 수 있도록 중력을 적용하고 플레이어와 동일하게 지형 데이터를 저장해 두고 Animate가 호출될 때 마다 높이 정보를 업데이트 합니다.

또, Animate에서 플레이어의 정보를 받아와 플레이어의 위치와 현재 오브젝트와의 위치를 이용하여 플레이어를 향하는 벡터를 만들고 그 벡터의 크기를 플레이어와의 거리로 보고 플레이어가 탐색거리보다 멀리 있는 경우 움직이지 않도록 하였습니다. 또, 움직이지 않는 것을 m\_nbMove에 저장하여 이 bool 값을 바탕으로 벽과의 충돌검사에서 제외하기도 하였습니다.

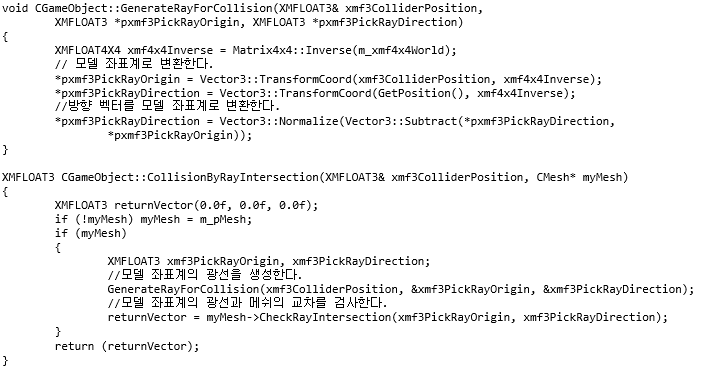
또, 위에서 쓴 것과 마찬가지로 플레이어를 바라보는 방향벡터와 up벡터를 외적하고 결과 벡터를 월드변환의 역변환으로 로컬 좌표계로 변경하여 회전하였습니다.

이때, 회전 각도는 지정된 각도로 회전하며 이 회전각을 바탕으로 회전 둘레를 계산하여 이동속도도 결정해 두었습니다.



충돌 검사 및 충돌 처리를 위한 법선 벡터를 구하기 위해 GenerateRayForCollision과 CollisionByRayIntersection을 기존의 Picking 함수를 수정하여 만들었습니다.

CollisionByRayIntersection은 화면 좌표계와 화면에서 클릭한 위치 대신에 월드 좌표계의 충돌한 오브젝트의 원점과 인스턴싱을 이용한 경우 메시를 받아 검사를 합니다. 이때 광선은 충돌한 오브젝트의 원점과 현재 오브젝트의 원점을 모델 좌표계로 바꾼 뒤 벡터로 만들어 생성합니다.

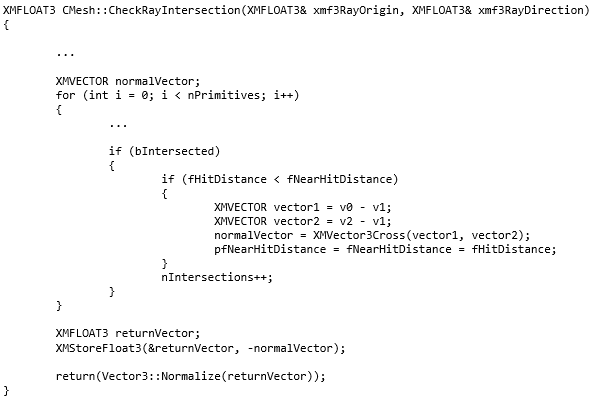


1. Mesh

지붕을 그리기 위한 CRoofMeshDiffused와 총알과 파티클을 처리하기 위한 CFragmentMeshDiffused를 추가하였으며 이때, 인덱스를 이용하여 생성하였습니다.

특히, CFragmentMeshDiffused는 이전의 CBulletMesh를 인덱스를 이용한 방식으로 수정하면서 파티클도 이 메시를 이용하기에 이름을 변경하였습니다.

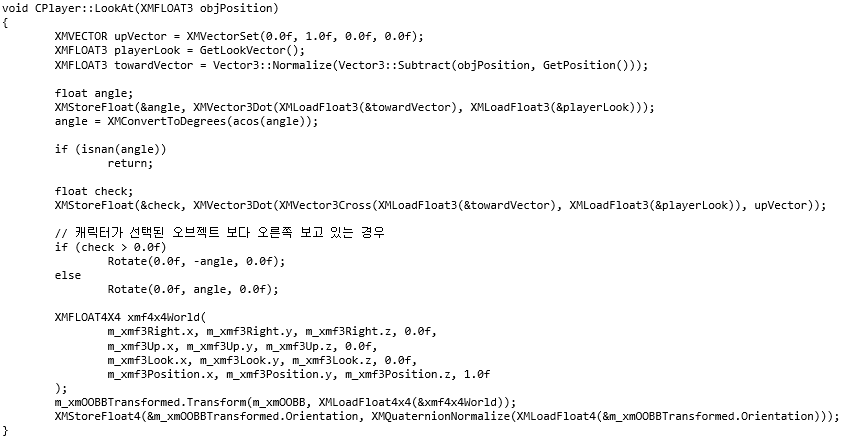
벽과의 충돌 처리를 위해 그 충돌한 면의 법선 벡터를 얻어 오기 위해 CheckRayIntersection을 수정하였습니다. 먼저, pNearHitDistance는 리턴하지 않기에 파라미터에서 제거하였습니다. fNearHitDistance가 갱신되는 경우 검사한 삼각형을 이루는 정점을 이용하여 법선 벡터를 얻어 저장하였으며, 최종적으로 적용된 벡터를 리턴 값으로 주었습니다.



1. Player

플레이어는 기본적으로 TerrainPlayer를 그대로 사용하였습니다. 그 중 플레이어의 방향을 설정하기 위해 추가한 LookAt과 MoveToEntrance가 있습니다. 이 중 MoveToEntrance는 입구의 위치를 저장해 두고 그 위치로 플레이어의 위치를 변경한 뒤 LookAt을 호출하여 방향을 설정합니다.

LookAt은 바라봐야 할 위치를 설정해 주면 그 방향으로 회전할 수 있도록 위에 서술한 대로 플레이어의 위치와 주어진 위치로 방향벡터를 만들고 이 방향벡터와 플레이어의 현재 방향 벡터의 내적으로 각도를 구하고 이 두 벡터와 업벡터의 스칼라 삼중적을 이용하여 어느 방향으로 회전할 지 결정하여 회전합니다.



1. Scene

씬의 경우 각각 다른 쉐이더를 사용하기에 기존에 포인터에서 배열로 변경하였으며 현재 5개의 쉐이더를 사용하고 있습니다. 또, 처음 프로그램 실행 시 4개의 맵 데이터 중 하나를 선택하여 열게 하기 위하여 fileNumber를 랜덤으로 생성하여 쉐이더들에 넘겨주어 동일한 파일을 열게 하였습니다. 또, 집을 만들기 위하여 충분한 평지를 얻기 위해 맵의 크기를 기존의 8배에서 20배로 수정하였습니다.

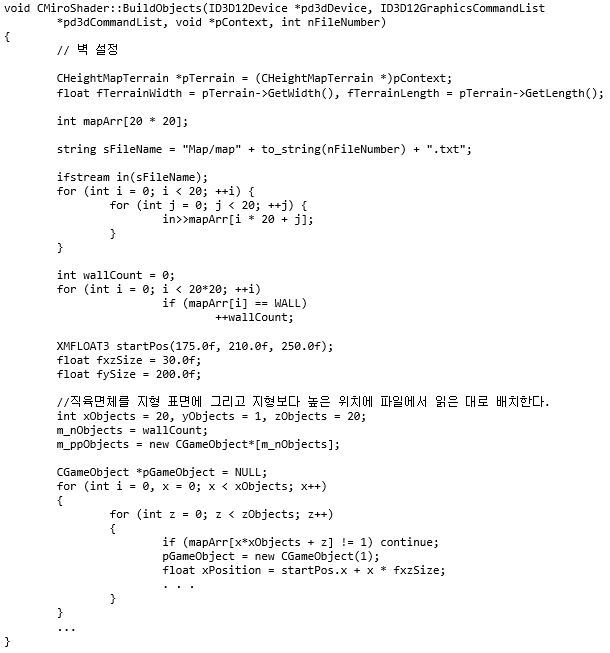


또, 파티클이 생성될 주 쉐이더에 파티클 쉐이더의 포인터를 넘겨 파티클을 생성할 수 있도록 처리하였습니다.

1. Shader

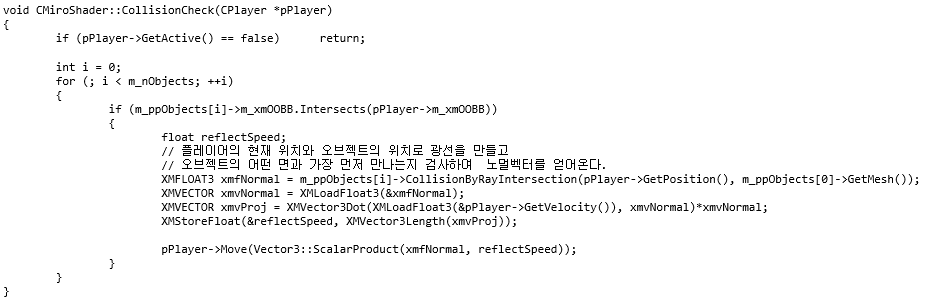
기본의 CInstancingShader의 이름을 MiroShader로 변경하여 미로를 만들기 위한 쉐이더로 사용하였습니다.

BuildObjects에서 Scene에서 넘겨온 파일 번호로 파일을 열고 그 파일에서 벽을 나타내는 숫자(1)의 개수를 파악하여 오브젝트의 개수로 지정하였으며, 각 벽의 크기를 동일하게 지정하였기에 벽이 아닌 오브젝트를 건너뛰어 의도한 대로 배치할 수 있었습니다.

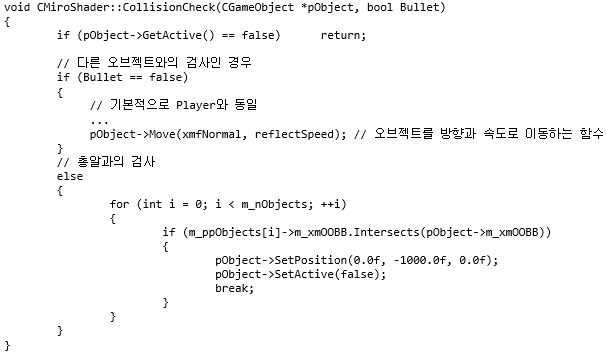


충돌 검사의 경우 플레이어의 위치를 CollisionByRayIntersection에 넘겨주어 오브젝트의 위치와 플레이어의 위치를 이용하여 월드공간상의 광선을 만들 수 있도록 하였으며, 인스턴싱을 이용하였기에 각 오브젝트가 메시는 가지고 있는 것이 아니기에 광선과의 충돌검사를 위해 메시를 넘겨주어 사용할 수 있도록 하였습니다.

이 방법은 벽과 오브젝트와의 충돌검사도 동일하게 적용하였습니다.



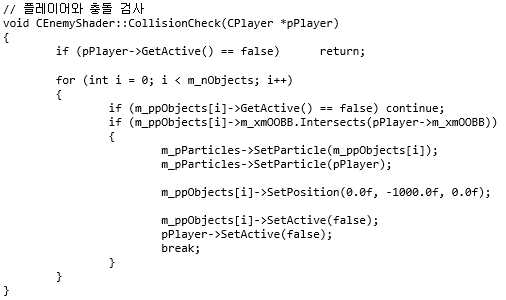
총알과 벽의 충돌은 총알이 충돌 후 사라지게 하였기에 단순하게 OOBB의 Intersects 검 사를 사용하여 충돌을 판단하고 충돌 후 사라진 것 처럼 보이도록 위치를 변경하고 더 이상 Animate를 하지 않도록 m\_bRender를 false로 설정하였습니다.



EnemyShader의 경우 빌드 시 미로를 빌드할 때와 마찬가지로 파일을 읽어 사용하며, 파 일에서 읽는 숫자가 (2)라는 것과 기본설정 외에는 동일하게 적용되어 있습니다.

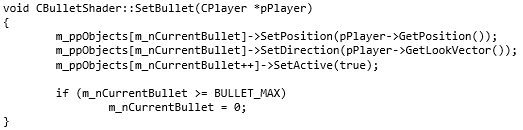
또, Enemy의 메시는 Sphere 메시는 사용합니다.

적과 플레이어, 적과 총알의 충돌 검사는 벽과 총알의 충돌검사와 마찬가지로 OOBB의 Intersect로 검사하며, 검사되어 충돌이 된 경우 위치를 바꾸고, 파티클 쉐이더에 파티클을 만들 오브젝트(적 또는 플레이어)를 넘겨서 그 정보를 이용하여 파티클을 생성하도록 합니다.



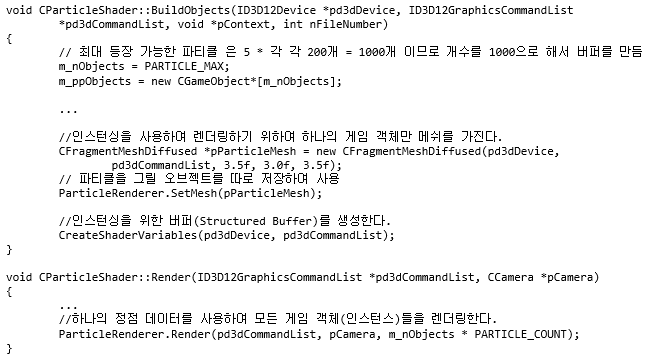
BulletShader의 경우 최대 동시에 화면상에 존재할 수 있는 총알의 수를 제한하여(BULLET\_MAX) 버퍼를 생성하였으며, 이 오브젝트 배열을 재활용하여 무한히 총알을 발사할 수 있는 것 처럼 보이게 하였습니다. 기본적으로 제 게임의 설정이 적을 클릭해야 총알이 나가는 구조이기 때문에 많은 총알이 필요하지 않아 이렇게 지정하였습니다.

또, 총알의 설정은 플레이어가 적을 클릭한 경우 SetBullet 함수에 플레이어 데이터를 넘겨 플레이어의 위치와 방향으로 지정합니다.

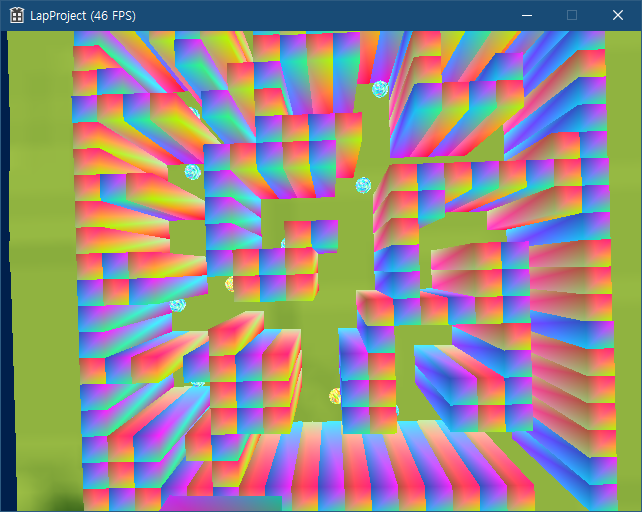


파티클의 경우 지난 과제에서는 동적으로 설정하였으나, 그 동적인 설정 과정이 프레임 레이트를 깎는다고 생각이 되어 최대 생성될 수 있는 파티클 수를 기준으로 버퍼를 생성하였습니다. 제 게임은 클릭 시 총알이 발사되며 최대로 많은 적이 모인 경우와 플레이어가 터지는 경우를 합쳐도 최대 5번으로 커버가 가능하며, 각 횟수마다 200개의 파티클이 생성되므로 m\_nObjects 를 1,000으로 설정하였습니다.

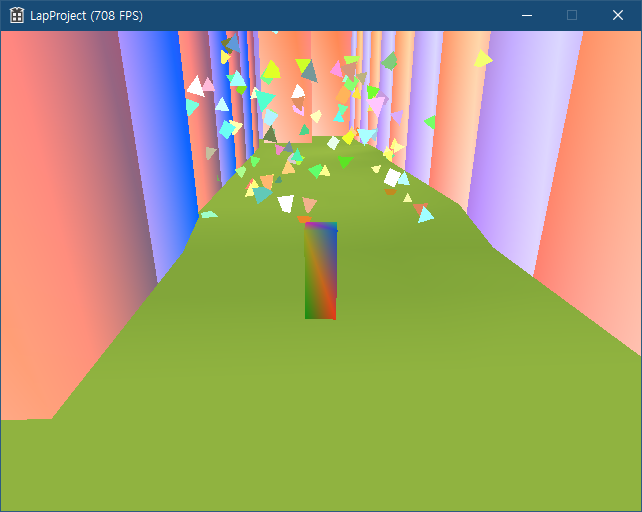
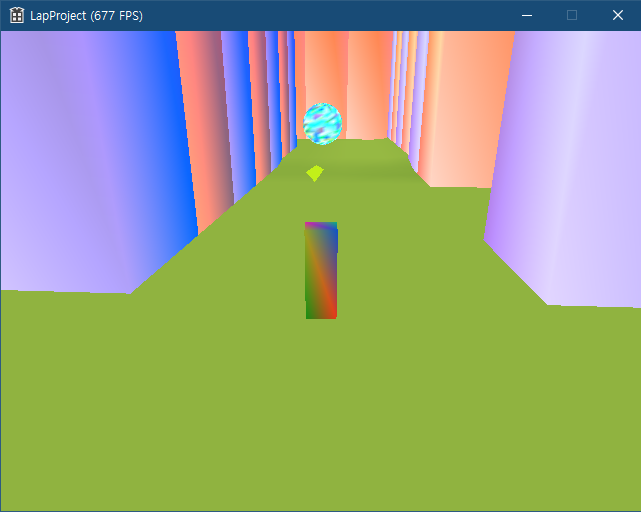
또, 인스턴싱을 하였는데 쉐이더가 가진 오브젝트를 그리는 게 아니라 그 오브젝트가 가진 파티클들을 그려야 하기 때문에 그 파티클을 그릴 Mesh를 저장할 오브젝트를 하나 따로 저장하여 사용하였습니다.



1. 프로그램 실행 화면



미로 구성



총알 생성 및 파티클 생성