3D게임1 과제 설명 문서

2018182025 이승준

1. 과제에 대한 목표 및 가정

본 과제는 기본적으로 레일을 깔고 그 위에서 자동으로 움직이는 열차 객체를 만드는 과제이다.

이 과제에서 가장 큰 목표로 두고 가장 중요하게 생각한 것은, 레일의 자연스러운 생성과 소멸이다. 이 과제는 긴 레일을 깔아야 하는 과제이기 때문에, 레일을 까는 과정에서 하드 코딩이 동원되기 쉽다. 우리가 생각하는 자연스러운 레일의 모습은 잘 꺾여져 들어가는 여러 곡선이 조화된 모습이고, 이는 소프트 코딩으로 구현하기 난해할 수 있기 때문이다.

나는 이러한 자연스러운 곡선을 구현하기 위해 4차 베지에 곡선을 활용해 보기로 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

*(4차 베지에 곡선 계산식)*

4차 베지에 곡선은 점 네 개를 매개변수로 하여 곡선을 그리는데, 두 번째 점과 세 번째 점을 잇는 구간에서 네 점의 정보를 모두 활용한다. 곡선을 잇는 과정에서 앞 뒤 점의 정보가 모두 활용되기 때문에 두 번째 점과 세 번째 점을 이은 4차 베지에 곡선으로만 이루어진 곡선은 곡선의 연결이 어색하지 않고 자연스럽다.

곡선을 생성하는 과정에서 하드 코딩을 지양하기 위해 최소한의 범위만을 지정하고 범위 내 랜덤한 좌표를 이용하여 곡선을 자동으로 생성되게 하기로 하였다. 이를 통해 Z축이 증가하는 방향으로 곡선이 계속해서 생성되며, 플레이어 객체는 이를 따라 쭉 이동한다.

곡선이 무한히 생성되기 때문에, 이미 지나쳐온 곡선을 삭제하여 메모리 확보가 용이하도록 계획하였다. 즉 컨테이너에 뒤쪽에 곡선이 계속해서 추가되고, 이미 지나쳐온 컨테이너 앞쪽에서는 곡선이 계속해서 삭제되는 구조로 작동하는 것이다. 이러한 동작에 최적화된 컨테이너인 *덱(deque)*을 이용하여 구현해 보기로 하였다.

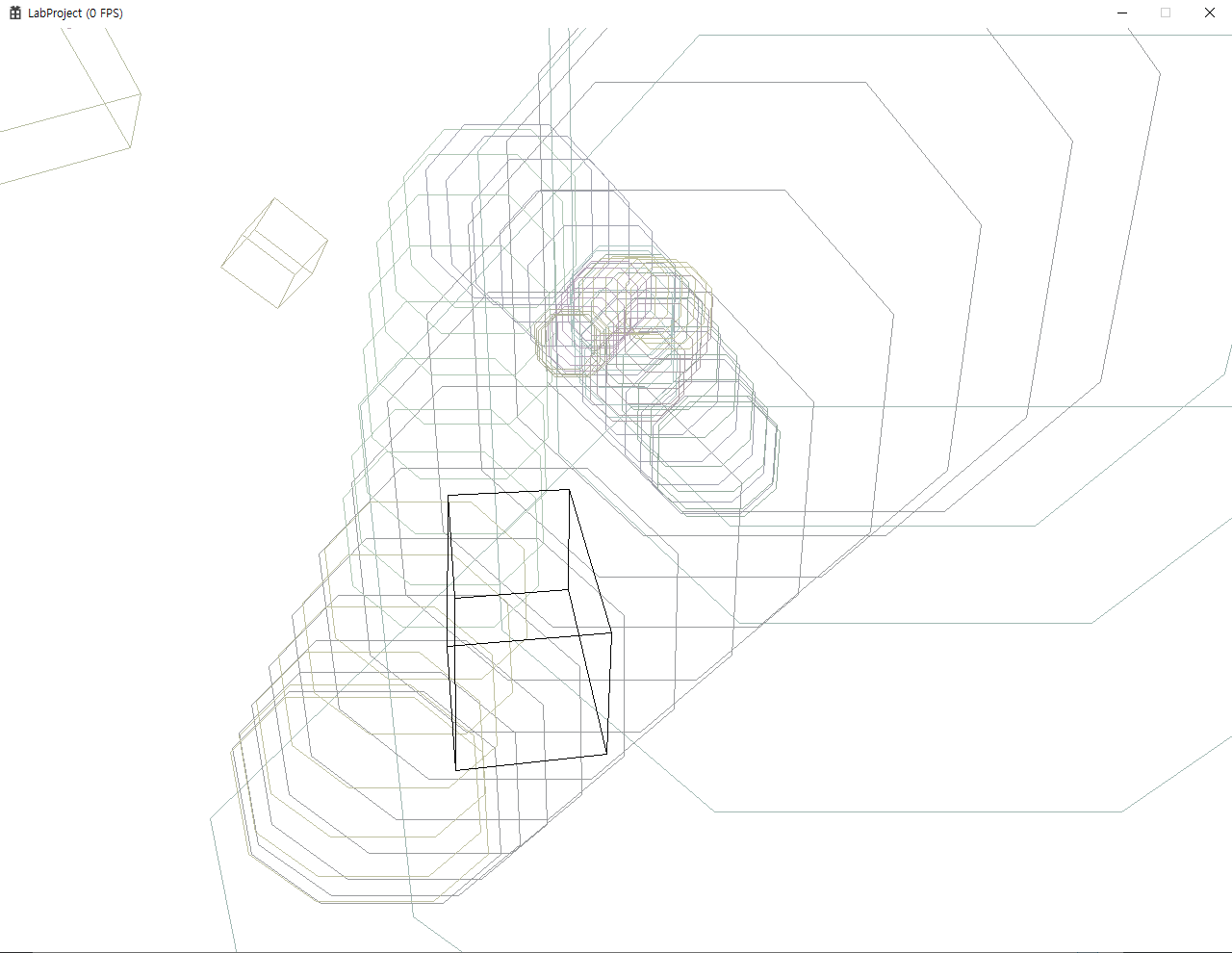
반면 적의 생성과 삭제의 경우, 나중에 생성된 객체가 먼저 제거될 수 있기 때문에 이러한 동작에 최적화된 컨테이너인 *리스트(list)*를 이용하여 관리하였다.

이와 같이 각 객체의 특징에 따라 가장 적절한 STL 컨테이너를 사용하여 객체를 관리하고자 하였다.

수업 시간에 다룬 *Software Renderer*의 예제 프로그램의 경우 마우스 클릭 및 드래그를 통해 카메라 시점이 따라 변하는 구조로, 매우 자연스럽다. 그래서 이 부분은 그대로 가져다 쓰기로 하였다. 다만 *Software Renderer*에서는 마우스가 움직이면 플레이어 객체(전투기)가 바라보는 방향이 같이 움직이기 때문에 플레이어 객체(열차)가 한 곳만 바라보며 항상 레일을 따라 이동해야 하는 이번 과제와는 맞지 않아 이 부분은 수정해야 하겠다.

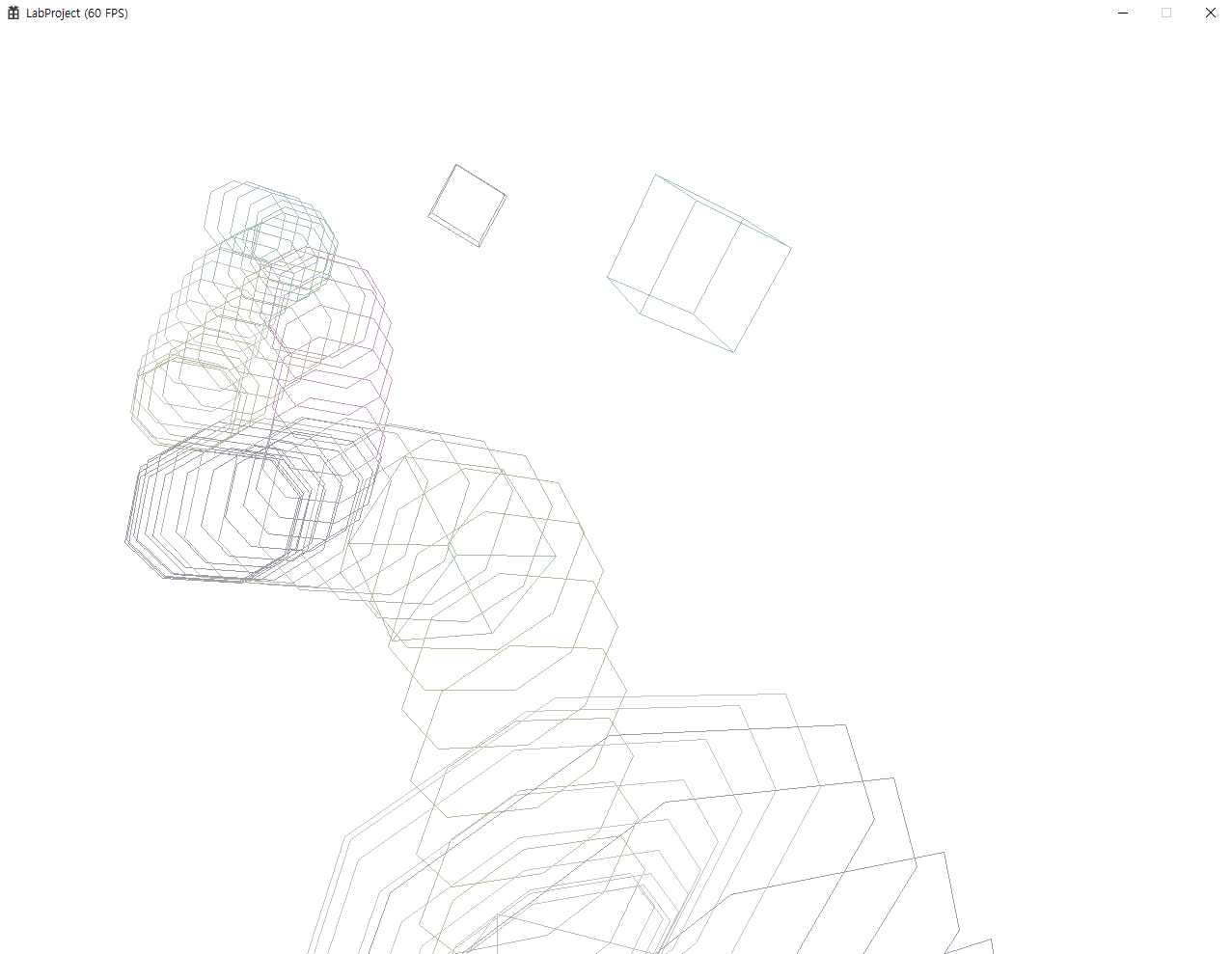
곡선 레일을 이동하는 객체가 계속해서 한 방향을 바라보며 이동하면 어색할 것이다. 객체가 움직이는 곳으로 방향을 틀어 자연스러운 움직임을 구현해 보고자 하였다.

1. 실행 결과와 조작법



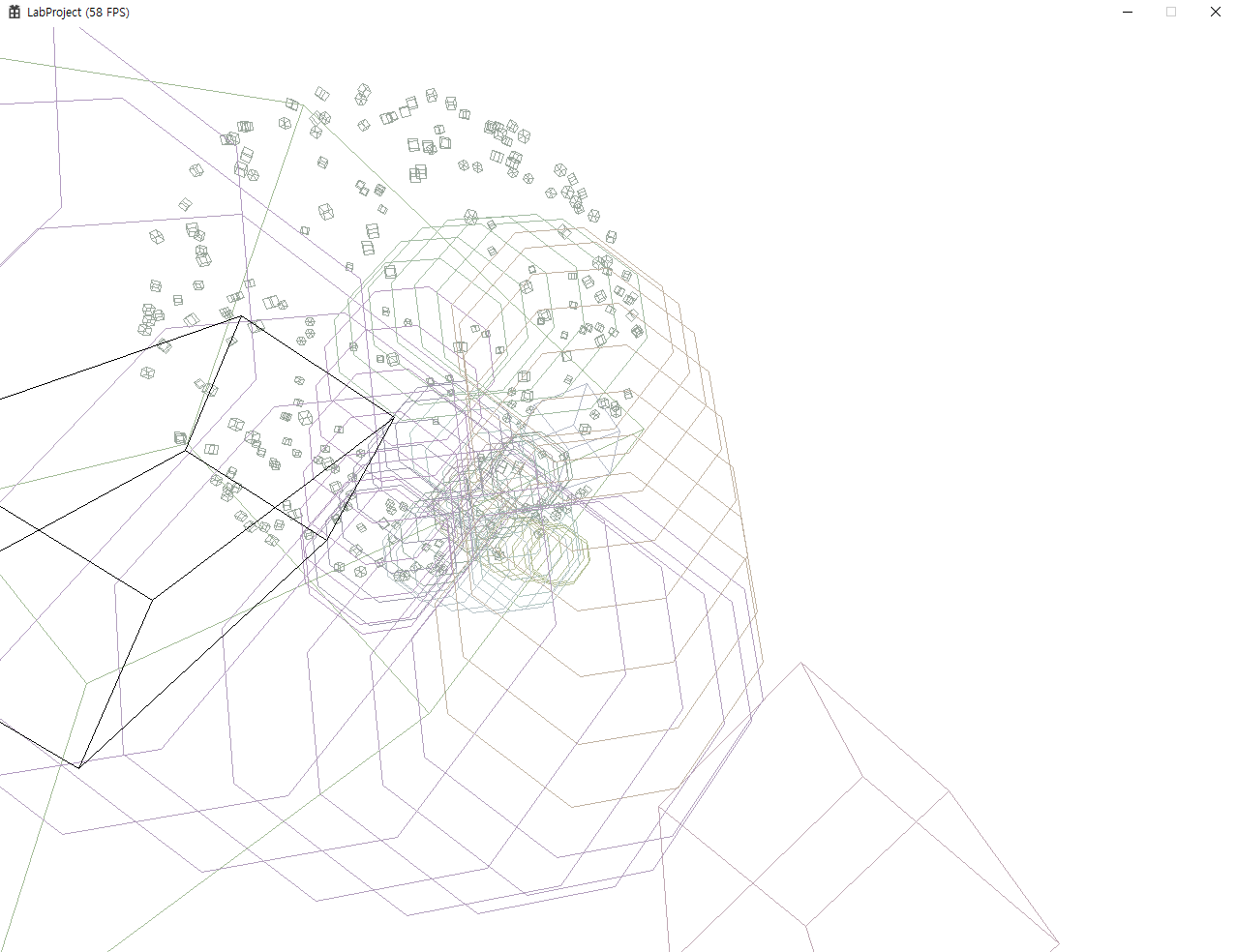
*(과제 실행 화면)*

프로그램을 실행하면 Z축이 증가하는 방향으로 원통형의 곡선 레일이 생성된다. 플레이어 객체는 자동으로 이 곡선을 따라 이동하며, 마우스 클릭 및 드래그를 통해 카메라를 회전시킬 수 있다. 물론 카메라를 회전시켜도 객체의 방향은 레일이 향하는 방향으로 고정된다.



*(적 접근)*

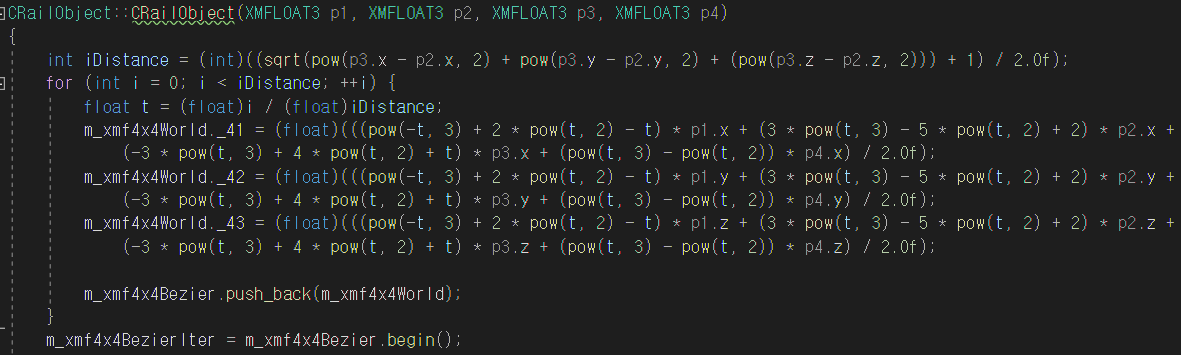
플레이어 객체의 주변에서는 적이 플레이어를 향해 점점 다가오는데, 이들의 속도는 시간이 지날수록 점차 빨라진다. 또한 일정한 시간 간격을 두고 계속해서 생성되기 때문에 총알을 맞춰 빠르게 제거해줘야 한다.



*(적 격추)*

총알은 *Ctrl* 키를 통해 발사할 수 있다. *Software Renderer*에서는 총알이 객체의 정면에서 발사되었으나, 이 과제는 객체가 한곳만을 바라보고 있기 때문에 카메라가 바라보는 방향으로 총알이 발사되도록 하였다.

1. 구현 내용



*(베지에 곡선 좌표 생성)*

랜덤으로 생성된 좌표 4개를 넘겨주면 이 정보를 바탕으로 베지에 곡선을 이루는 좌표를 생성하고, 이를 벡터에 저장한다. 이 좌표를 통해 레일을 그린다.

텍스트이(가) 표시된 사진

자동 생성된 설명

*(레일 객체 생성 및 삭제)*

*덱(deque)*을 통해 레일 객체를 관리하며, 이미 지나쳐 쓸모 없어진 레일은 *pop\_front*를 통해 삭제해 주고, 앞으로 지나칠 구역에 *push\_back*을 통해 미리 레일을 생성해 주었다.

레일의 z축은 일정하게 증가하지만, x축과 y축은 랜덤하게 지정된다.

적 객체의 경우는 리스트(list)를 통해 관리되며, 피격을 통해 제거된 적 객체는 폭파 애니메이션 출력 후 리스트에서 제거된다.

적은 일정한 시간 간격을 두고 최대 개수(25)가 될 때까지 계속해서 생성된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

(Player와 Object의 구분)

Player 객체인 m\_pPlayer는 열차와 관련된 모든 연산을 정의하나 실제로 Render하지 않는다. 대신 열차 객체의 Render만을 위해 정의된 m\_pTrainObject가 Player 객체의 위치를 받아 객체를 그린다.

이를 통해 객체가 회전하지 않고도 Player 객체 내부에서 정의되는 카메라 객체의 회전을 구현할 수 있다. 실제로 Player 객체는 회전하지만 m\_pTrainObject는 회전하지 않기 때문이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

*(플레이어의 방향 설정)*

다음 Rail Mesh의 위치를 받아와 이전 Rail Mesh의 위치와의 노말 벡터를 구하여 플레이어가 향할 방향을 계산한다.

이렇게 구한 XMFLOAT3 변수는 Player의 회전 방향 결정에도 사용되는데, 물론 m\_pPlayer가 아닌 m\_pTrainObject를 회전시키는데 사용된다.

1. 결론

가장 중요하게 생각한 부분인 베지에 곡선의 자연스러운 구현에 성공했다. 카메라 회전 및 총알 발사, 적 생성 및 제거 역시 부드럽게 진행되어 처음 계획했던 소기의 목적은 어느 정도 달성했다고 생각한다.

다만 아쉬운 부분도 몇 가지 있었는데,

1. 레일이 순환하지 않는다.

다만 이 부분은 무한한 레일 생성을 구현하기 위해 어쩔 수 없는 부분이었다고도 생각된다.

1. 객체지향 원칙에 어긋난 부분이 많다.

대표적으로 플레이어 움직임에 관련된 상당 부분을 RailObject에서 처리한다.

m\_pTrainObject 객체를 따로 만든 것도 엄밀히 말하면 적절하지는 않다고 생각한다.

1. 플레이어의 움직임이 조금씩 끊어진다.

RailObject에서 다음 RailObject로 넘어가면서 발생하는 현상으로 추정된다.

다음 과제부터는 개발 초기 단계부터 이러한 문제의 해결 방안에 대해 철저하게 구상하여 같은 문제가 발생하지 않도록 하고 싶다.