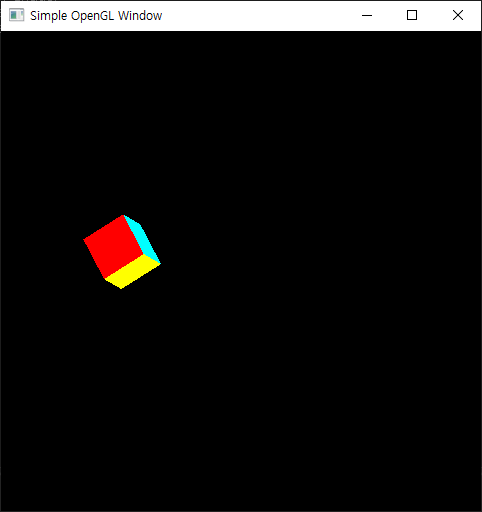
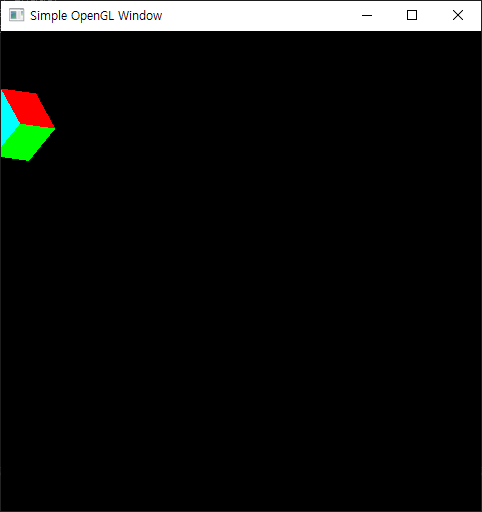
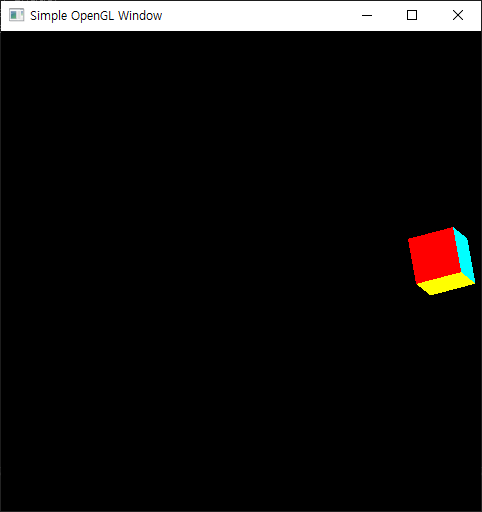
**Homework #8 Animation Interpolation**

**1. 사용법**

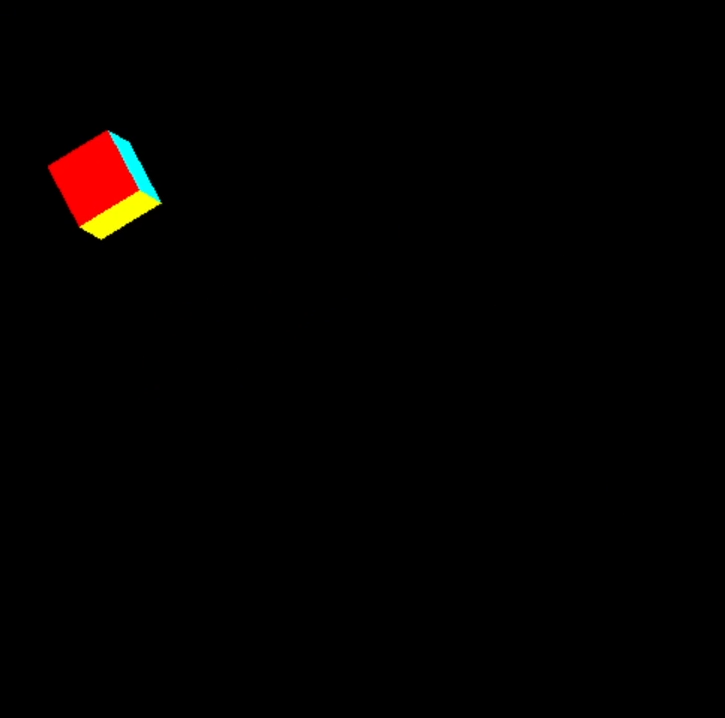
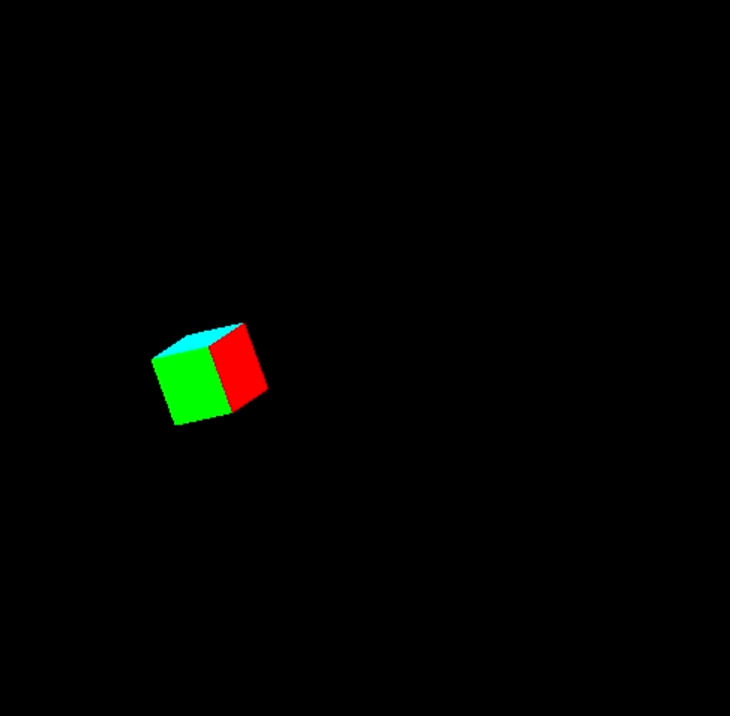
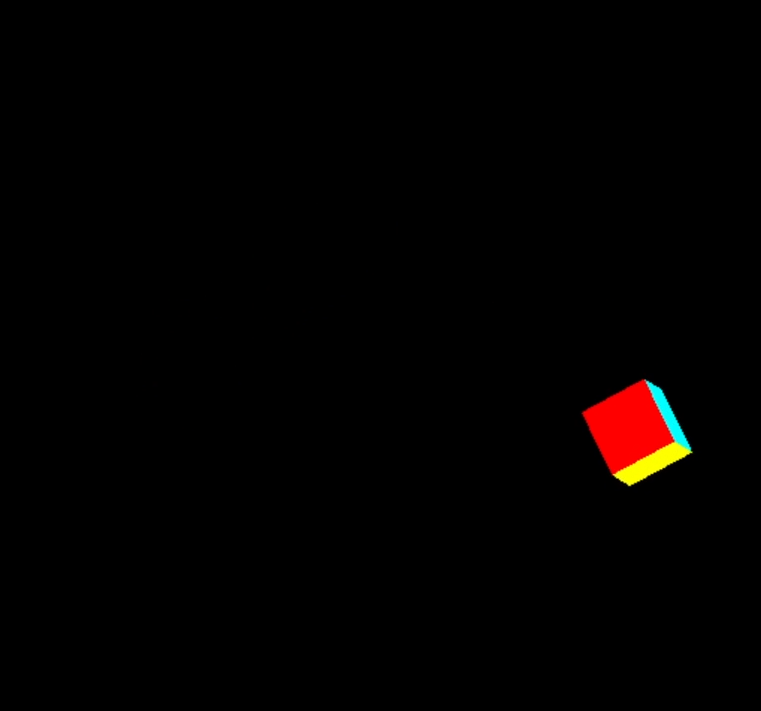
Control Point는 총 5개. 1~5번을 누른 뒤 마우스를 클릭하면 Control Point의 X,Y좌표가 바뀐다.

M을 누르면 Draw Mode가 바뀌고, Interpolation이 Bezier와 Linear Interpolation이 서로 바뀐다.

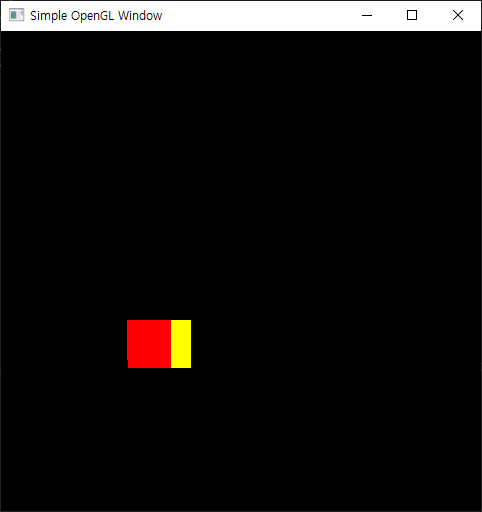
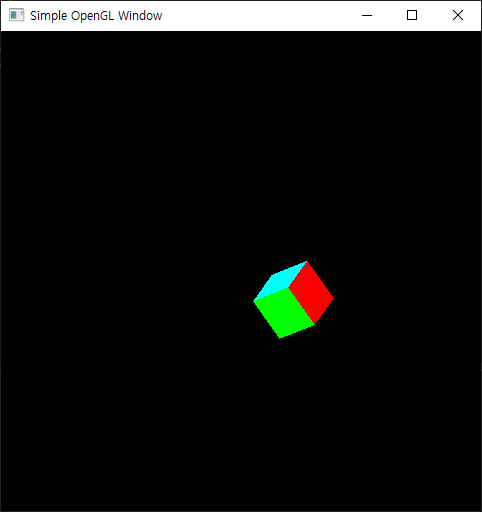
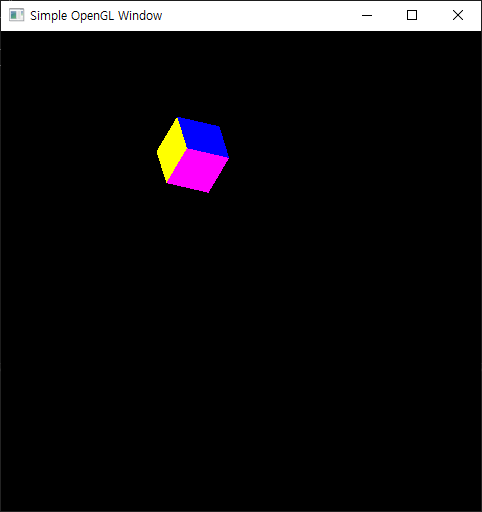
Linear Interpolation

Bezier Interpolation

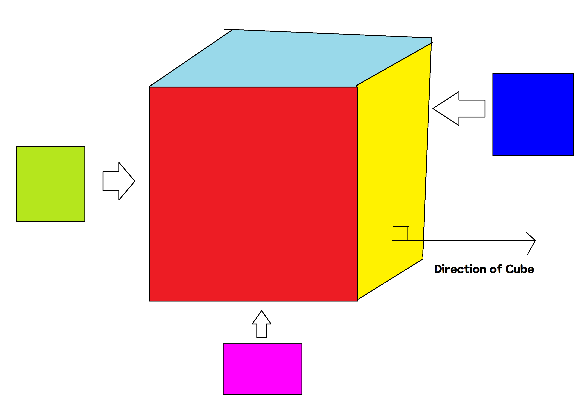
Bezier Control Point Change

기존의 Curve에서 원형모양으로 이동하도록 경로를 바꾸었다.

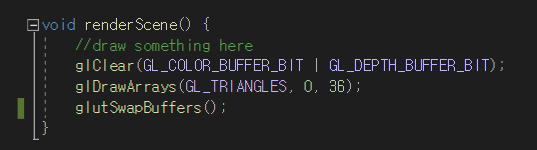
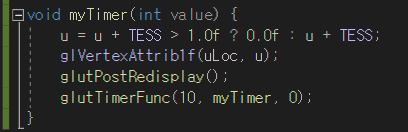
각각의 Control Point의 Z좌표가 다르기 때문에 원형으로 돌면서 큐브도 한바퀴를 돌게 된다.

Cube는 아래와 같은 모양을 가지고, 큐브가 바라보는 방향은 항상 노란색 면이다. 이동 경로 간에 Quaternion을 이용해서 언제나 진행방향을 바라보도록 프로그램을 만들었다.



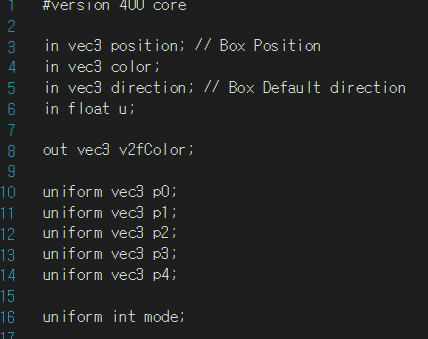
**2. 주요 구현 부분 및 Shader Source**

대부분의 구현은 최대한 GPU를 활용하고자 하는 마음에 Vertex Shader 안에서 계산을 진행하였다.

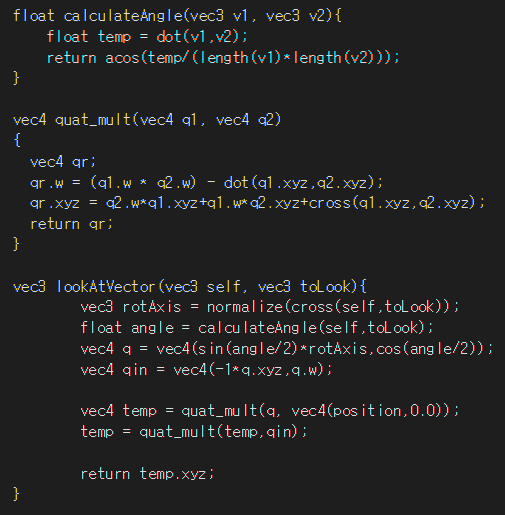
 

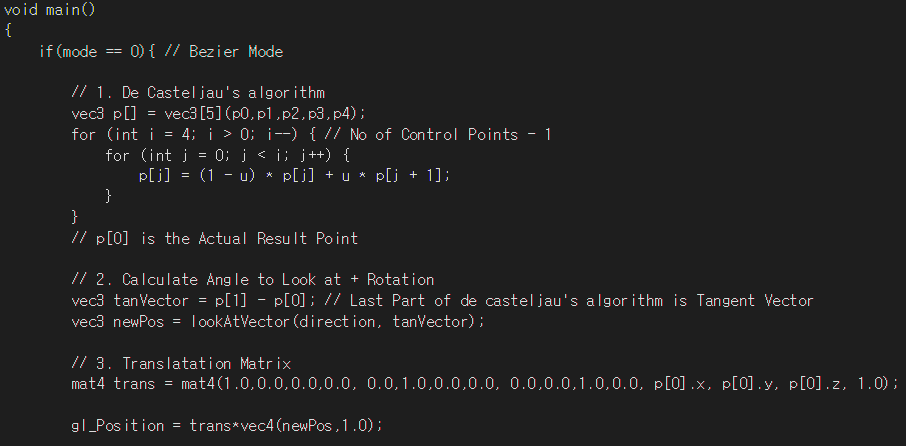
renderScene에서는 단순히 큐브만 그려주었고, u값의 증가는 glutTimerFunc을 이용해서 지속적으로 증가시켜주었습니다. 나머지 곡선의 계산은 전부 shader에서 진행하였습니다.

- Vertex Shader

Box의 경우 Local Coordinate 상에서 원점을 중심으로 만들었다. 또한 위에서 적었듯이 Direction은 (1,0,0)으로 기본적으로 오른쪽을 바라보게 설정하였다.

이외에는 5개의 Control Point를 조절하기 위해서 p0~4를 줬고, Interpolation Mode를 구분하기위해서 mode 변수도 줬다.

구글링을 해보면서, GLSL에서 지원해주지 않는 함수 두 개를 구현하였다. 벡터 두개의 사잇각 함수와, 쿼터니언 곱셈을 강의자료를 활용하여서 구현하였다. 이때 강의자료에 쿼터니언은 [s (x y z)]의 방식이였지만, GLSL의 vec4는 요소이름이 x,y,z,w순이라서 혼동요소를 줄이기 위해서 s와 v의 위치를 바꾸어서 구현하였다.

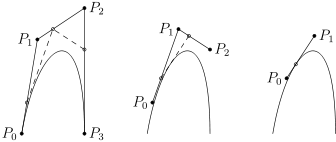
세번째 함수는 특정 벡터를 바라보게 하는 함수인데, 자신이 바라보는 곳과 바라볼 벡터의 외적을 하여 나온 법선 벡터를 회전축으로 두 벡터의 사잇각만큼 돌린다는 것을 이용해서, 여기서 쿼터니언을 활용하면 쉽게 구현이 될 것 같았다.

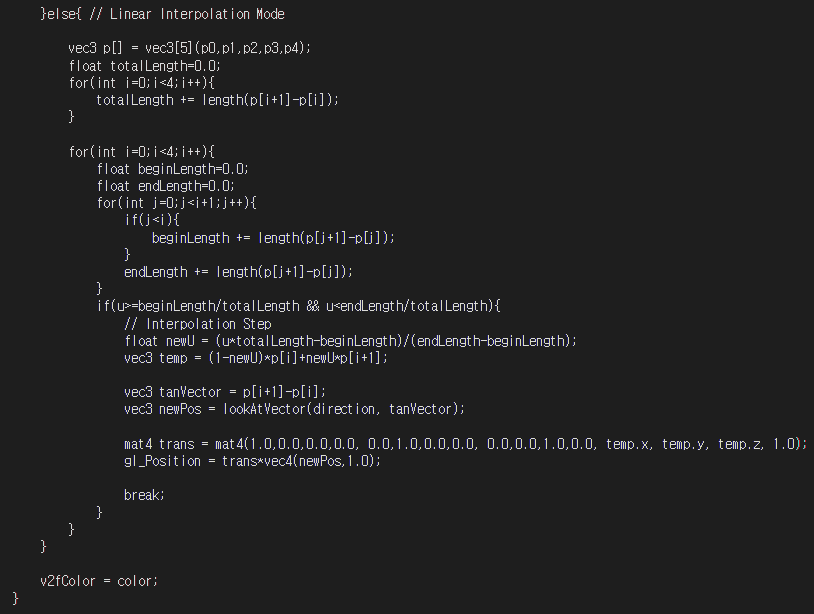
Control Point 5개의 경우에는 4차 베지에 커브가 만들어지면서 공식이 매우 복잡해진다. 따라서 이를 De Casteljau’s Algorithm을 이용하면 쉽게 구현이 가능하다. 재귀의 구현은 불가능하지만, 이중for문으로 쉽게 구현이 가능하다.

흔히들 하는 좌측의 별 찍기 문제의 방식으로 구현하고, 값을 재사용하지 않는 방향으로 잘 덮어써주면, 쉽게 구현이 가능했다.

해당 방식으로 일단 좌표는 잘 구했고, 그 좌표를 Translation Matrix를 만들어주면, 원점을 기준으로 만들어진 큐브를 u를 이용해서 만들어진 Curve의 위치를 기준으로 큐브를 생성하게 할 수 있다.

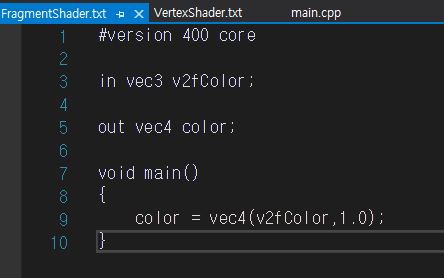
또 De Casteljau’s Algorithm를 보면 마지막으로 만들어진 P0와 P1을 보면, P1-P0는 곡선의 접선 방향이다. 이를 이용하면, 특별한 공식없이 점을 만드는 과정에서 바로 접선벡터를 만들어낼 수 있고, 이 벡터와 큐브의 방향벡터를 이용해서 위에서 만든 함수에 넣으면 Curve의 진행 방향을 바라보게 만들 수 있다. 이렇게 Bezier Interpolation의 위치와 방향을 구할 수 있다.





Linear Interpolation은 간단하게 각 Control Point간의 길이의 합을 구해서, u가 현재 상대적으로 어떤 위치에 있는지를 계산해서, 선형 보간을 하는 방식이다. 특별히 복잡한 식은 없고, 이 경우에는 바라보는 방향을 특정 구간(예: p[2]~p[3] 구간)에서 해당 Control Point두개를 빼면, 방향벡터가 바로 나온다. 이후 Bezier와 같은 방식으로 큐브가 바라보게 만들었다.

- FragmentShader

특별히 Fragment 단위로 할 것이 없어서 단순히 색상만 그대로 출력하는 것을 진행하였다.

**3. 결론**

수업에서 알려준 여러가지 방식은 사용하지 못했지만, 나름대로 이해한 방식으로 열심히 만들었습니다. 지난 학기에 쿼터니언에 대한 이해가 많이 부족했는데, 이번 과제를 통해서 조금이라도 이해하는 시간이 되었던 것 같습니다.