컴퓨터애니메이션실습(CAL)

HW7_Spherical Linear Interpolation

Self-scoring table

	P1	P2	P3	E1	Total
Score	**1**	**1**	**1**	**1**	**4**

2018707068 김경환

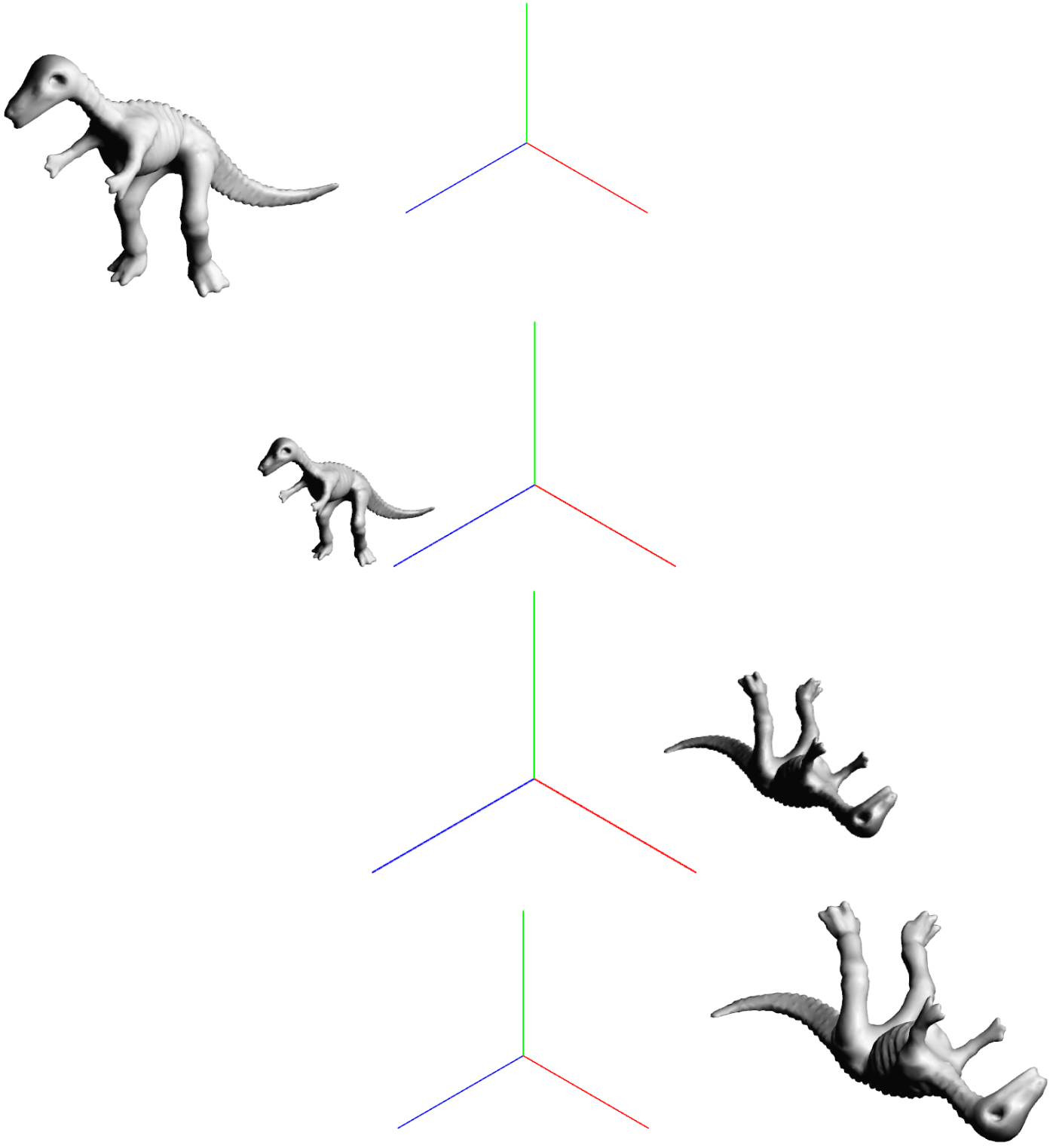
KwangWoon University

Code Analysis:

먼저 Init()에서 출력되는 내용들을 살펴보면 "q1 = 1, 0 0 0"으로 Vector3f(1, 1, 1).normalize()를 axis로 하고 angle을 0 rad로 주었기 때문에 실수부에서만 1이 나오고 나머지 허수부는 전부 0인 것을 볼 수 있다. 그리고 q2는 q1과 동일한 axis를 주고 angle은 181도를 주었고 "q2 = -0.00872665, 0.577328, 0.577328, 0.577328"이 출력된 것을 볼 수 있다.
한편 AngleAxisf aa(q1.inverse() * q2);을 수행했을 때 q1의 경우는 angle이 0도로 회전이 없으므로 q2에 대한 AngleAxisf 만들어질탠데 출력문은 angle이 181도가 아닌 179도, axis가 (0.577328, 0.577328, 0.577328)가 아닌 (-0.577328, -0.577328, -0.577328)로 출력된 것을 확인할 수 있다.
이는 AngleAxis가 자동으로 antipodal equivalence를 적용해준 것인데, 그렇기 때문에 angle은 180도 이내로 나오고 axis는 반대로 뒤집힌 것을 확인할 수 있다.
마지막으로 q3 = -q2로 설정하여 "q3 = 0.00872665, -0.577328, -0.577328, -0.577328"이 출력되었고 이러한 q3로 AngleAxisf aa(q1.inverse() * q3);를 수행했을 때 출력문이 "Angle = 179 degree, Axis = -0.577328, -0.577328, -0.577328"로 AngleAxisf aa(q1.inverse() * q2);를 수행한 출력문과 동일한 것을 확인할 수 있다. 즉, Eigen에서 제공하는 AngleAxis는 Antipodal equivalence를 감안하여 계산하는 것을 확인할 수 있다.

실습을 위해서 ocillator를 만들었는데 이는 elapsed time이 증가함에 따라 currTime을 누적하고 이를 3초의 interval로 나눠 n-iteration 변수 n을 만든다. 이러한 n으로 (currTime - n * interval) / interval을 수행해 3초마다 0에서 1까지 증가하는 변수 s를 만든다. 마지막으로 (s < 0.5) ? 2 * s : 2 * (1 - s)을 통해 s가 0~0.5일 때는 0에서 1로 증가하고 0.5~1일 때는 1에서 0으로 감소하는 oscillator, 변수 t를 만든다.
그리고 Transformation matrix에서 Translation과 rotation을 수행하는 부분을 이러한 t를 이용하여 Linear interpolation을 수행한다.
"Linear interpolation of position", "Linear interpolation of rotation matrices", "Linear interpolation of unit quaternions with normalization", "Spherical linear interpolation using Eigen's slerp"는 크게 설명할 부분은 없으므로 "Spherical linear interpolation using Eigen's AngleAxis (Log)", "Spherical linear interpolation using Eigen's AngleAxis variation"에 대해 설명하겠다.
먼저 "Spherical linear interpolation using Eigen's AngleAxis (Log)"을 수행하기 위해 q1.inverse()를 대신해서 q1.conjugate()를 사용했는데 이는 q1이 unit Quaternion이므로 inverse는 conjugate/norm의 표현이 conjugate와 같아짐을 알 수 있다. 그리고 AngleAxisf aa(q1.conjugate() * q2);를 수행함으로써 AnglaAxis를 얻어내는 것을 볼 수 있는데 이는 logarithmic map을 수행한다고 볼 수 있다. 또한 이러한 AngleAxis의 angle에 t를 곱함으로써 Sphererical interpolation을 수행하였고 이를 Quaternion q1에 곱했는데 이것이 가능한 이유는 Eigen에서 AnglaAxis를 Quaternion으로 승격시켜 계산하기 때문이다. 즉, 이는 exp()을 수행한다고 볼 수 있다. 마지막으로 Quaternion을 matrix로 바꿔 Transformation matrix의 rotation 부분에 대입해준 것을 볼 수 있다.
"Spherical linear interpolation using Eigen's AngleAxis variation"은 위 과정을 동일하게 수행하지만 마지막에 q1에 AngleAxis를 곱할 때 AngleAxis를 Quaternionf()를 통해 Quaternion으로 바꿔 수행한다.

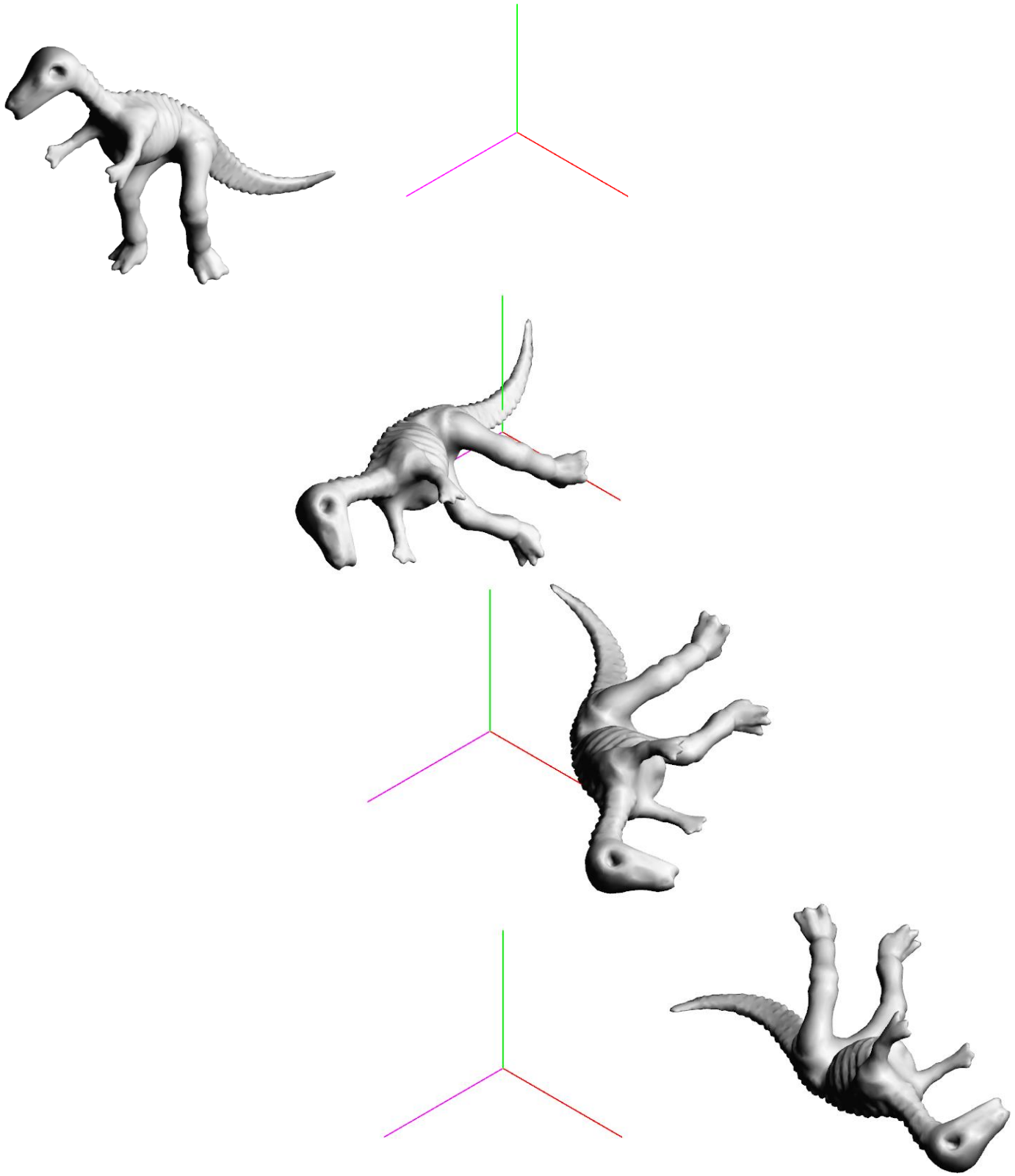
Practice 01. Linear interpolation of rotation matrices:



Rotation matrix의 Linear interpolation에서 t가 0에서 1로 변하는 모습이다. 대략 맨위부터 (0, 0.45, 0.6, 1)
위와 같이 dinosaur가 좌우로 이동하는 이유는 p1 = Vector3f(-1, 0.5, 2), p2 = Vector3f(2, 0.5, -1)을 Linear interpolation해 Transformation matrix의 translation 부분에 넣었기 때문이다.
Size가 줄어드는 이유는 Linear interpolation으로 만들어진 matrix가 normalize되지 않았기 때문이다.
회전에 대해서는 정확히는 모르겠지만 Quternion q2를 (s, **v**)라고 하면 Quaternion에서 rotation matrx을 계산할 때 필요한 값은 s = cos(θ/2), **v** = axis * sin(θ/2)으로 표현했을 때 s와 **v**의 계수이다. Antipodal equivalence를 만족하는지 보기 위해 q2를 rotation matrix로 바꾼 것과 -q2를 rotation matrix로 바꾼 걸 비교해보면 -cos(θ)는 cos함수가 우함수이므로 cos(θ) = -cos(θ)이고 -sin(θ)는 sin함수가 기함수이자 주기

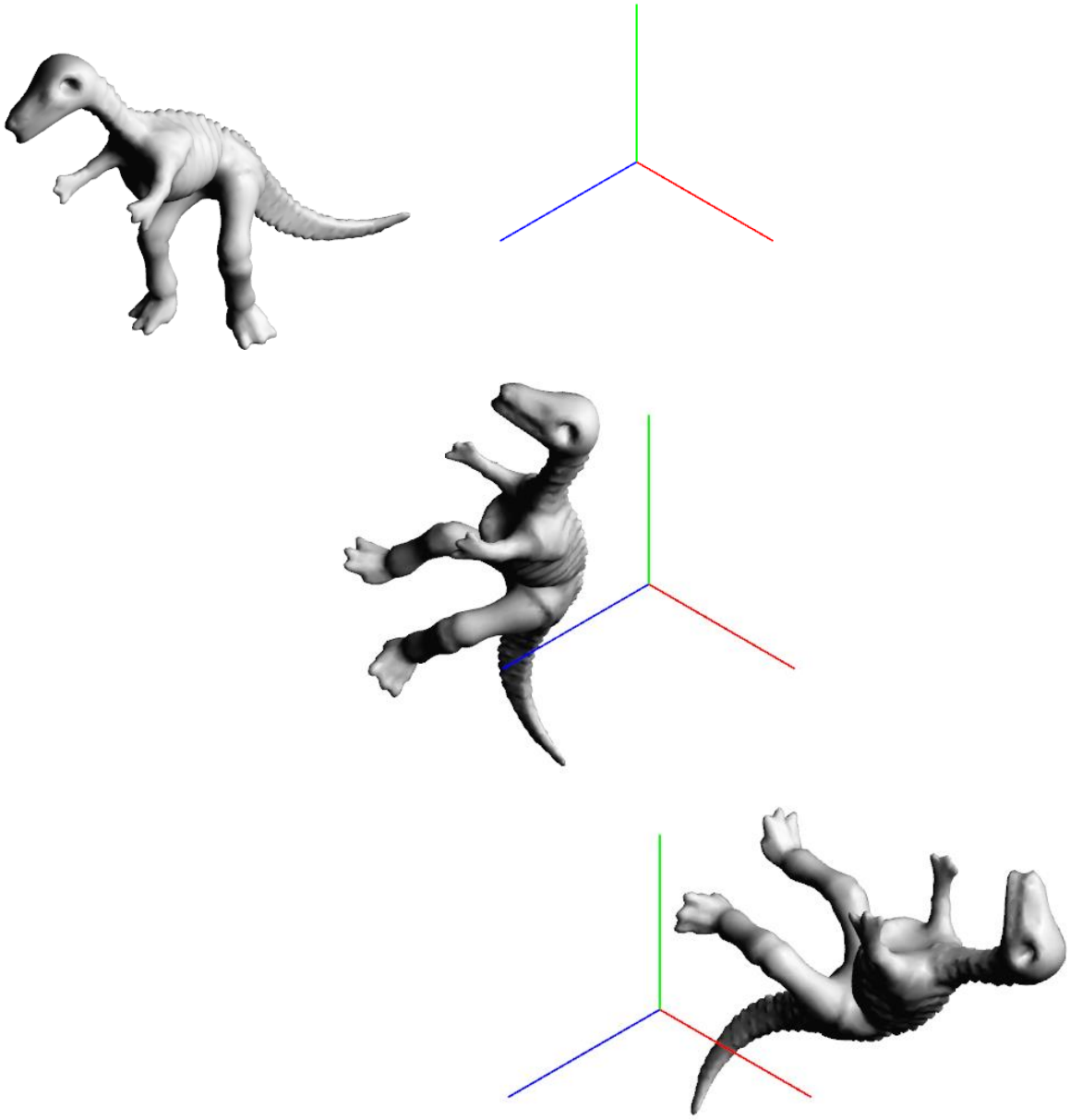
함수이므로 -sin(θ) = sin(2π-θ)가 되어 이때 angle은 2π-θ가 된다. 따라서 이는 Quaternion q2로 표현한 rotation matrix는 antipodal equivalence를 만족한다는 것과 같다.
하지만 rotation matrix의 Linear interpolation은 q2의 angle이 π에 근처일수록 q2의 axis는 q1과 반대가 되므로 181도 rotation하는 q2의 경우에는 t가 0.5 근처일 때는 q1과 q2을 합한 matrix의 determinant가 거의 0까지 줄어드는데, 이로인해 rotation이 영향이 크게 작용하여 해당 t가 0.5인 근처에서 모든 rotation이 이루어지기 때문인 것 같다.

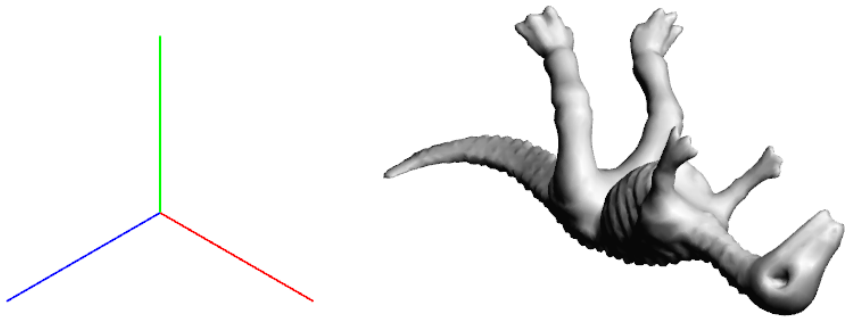
Practice 02. Linear interpolation of unit quaternions with normalization:



Quaternion의 Linear interpolation에서 t가 0에서 1로 변하는 모습이다. 대략 맨위부터 (0, 0.45, 0.6, 1)
위와 같이 dinosaur가 좌우로 이동하는 이유는 p1 = Vector3f(-1, 0.5, 2), p2 = Vector3f(2, 0.5, -1)을 Linear interpolation해 Transformation matrix의 translation 부분에 넣었기 때문이다.
Rotation matrix의 Linear interpolation과는 다르게 Size가 줄어들지 않는 이유는 Linear interpolation으로 만들어진 Quaternion을 normalize했기 때문이다.
여기서도 회전이 어떻게 이루어지는지 정확히는 모르겠지만 Linear interpolation에 사용된 Quaternion q1, q2가 rotation이 아닌 orientation이기 때문에 이를 Linear interpolation하면 orientation이 만들어지게 되고, 이를 matrix3f()를 통해 rotation matrix로 바꿔 사용하기 때문이 이런 모습이 보여지는 것 같다. 따라서 각 지점에서의 orientation의 rotation matrix를 구해 rotation하므로 antipodal equivalence를 만족하지 않고 Spherical을 고려하지 않았기 때문에 artifact을 갖는 것 같다.
위에서 말한 artifact는 rotation의 angle이 균일하지 않고 t가 0.5인 근처에서 다른 곳에 비해 많은 rotation이 이루어진다는 것을 통해 확인할 수 있고 antipodal equivalence를 만족하지 않는 모습은 angle의 값을 181에서 더욱 키우면 0~π까지의 rotation이 아닌 이를 넘어서는 rotation이 발생하는 것을 볼 수 있다.

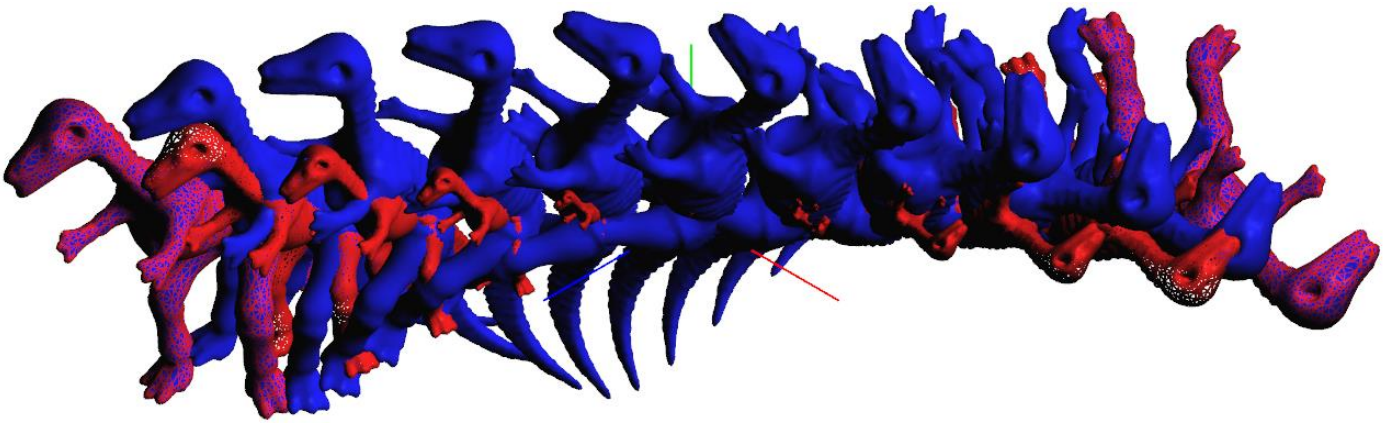
Practice 03. Spherical linear interpolation using Eigen's slerp:





Quaternion의 Linear interpolation에서 t가 0에서 1로 변하는 모습이다. 대략 맨위부터 (0, 0.45, 0.6, 1)
위와 같이 dinosaur가 좌우로 이동하는 이유는 p1 = Vector3f(-1, 0.5, 2), p2 = Vector3f(2, 0.5, -1)을 Linear interpolation해 Transformation matrix의 translation 부분에 넣었기 때문이다.
eigen에서 지원하는 slerp() 함수 자체에서 quaternion의 normalize를 진행하기 때문에 Size가 줄어들지 않고 일정한 것을 볼 수 있다.
Linear interpolation을 Spherical하게 고려하였으므로 rotation의 angle이 일정하며 Quaternion의 Linear interpolation과는 다르게 artiface가 보이지 않는다.

Exercise 01. Visual comparison between LERP of rotation matrices and SLERP of quaternions:



파란색의 solid fill이 quaternion의 SLERP를 t가 [0, 1, 0.1]이 되도록 총 11개를 그린 모습이다.
빨간색의 wireframe이 rotation matrix의 LERP를 t가 [0, 1, 0.1]이 되도록 총 11개를 그린 모습이다.
두 결과를 비교해보면 먼저 두 case 모두 antipodal equivalence를 만족하기 때문에 t가 0, 1일 때는 똑같은 모습을 보인다. 하지만 t가 0, 1일 때를 제외하면 rotation matrix의 LERP이 normalize되지 않아 size가 균일하지 않고 rotation의 angle이 일정하지 않은 artifact를 보이는 것을 확인할 수 있다.

이러한 mesh를 그리는 방법으로는 pracitce의 update() 함수를 수정하여 flag (0, 1)에 따라 rlerp 또는 slerp를 수행하도록 하였고 Linear interpolation에 사용되는 t를 [0, 1, 0.1]이 되도록 총 11개를 전달하였다.
또 interpolation의 결과를 ModelView matrix에 glMultMatrixf()를 통해 곱하고 practice의 drawmesh() 함수를 수정해 위에서 전달한 flag와 동일한 flag (0, 1)을 전달하여 soild fill, wireframe을 정해주었다.
그리고 위와 같은 과정이 한 번의 render() 함수 호출에 그려져야 하는데 ModelView matrix에 translation matrix가 incremental하게 곱해지므로 위 과정 직전과 직후에 glPushMatrix(), glPopMatrix()를 수행한다.
Soild fill 11개, wirefram 11개 총 22개가 그려져야 함으로 11번 반복하는 for문을 2개 작성한다.