# 컴퓨터애니메이션실습(CAL)

## HW6_Rotation


**Self-scoring table**

| | P1 | P2 | P3 | P4 | E1 | Total |
|---|---|---|---|---|---|---|
| **Score** | **1** | **1** | **1** | **1** | **1** | **5** |

2018707068 김경환

KwangWoon University

## Code Analysis:

Basic rotation을 수행하기 위해 rotation matrix from angle-axis rotation, Quaternion, Transformation을 사용하였다. Angle-axis rotation과 Quaternion은 모든 original vertex와 normal에 R(rotation matrix) 혹은 Q(quaternion), $Q^{-1}$(inverse quaternion)을 곱해 새로운 vertex와 normal을 구하고 이를 통해 drawing한다. 하지만 Transformation matrix를 사용하는 경우에는 vertex와 normal을 original 그대로 가져다가 사용하고, Transformation matrix는 opengl의 pipeline에서 연산되기 때문에 앞에 두 방법보다 매우 빠르다.

Incremental Rotation을 수행하기 위해 Rotation matrix without normalization, Rotation matrix with normalization, Quaternion with normalization을 사용하였다. Rotation matrix without normalization은 normalization이 되지 않았기 때문에 Determinant가 증가하며 Scaling되는 모습을 보이고 x, y, z의 dot product의 결과가 1이 아닌 값을 보인다. Rotation matrix with normalization은 simple normalization을 사용하여 Deterterminant가 증가하지는 않아 Scaling되지는 않았지만, 앞선 예시와 같이 x, y, z의 dot product의 결과가 1이 아닌 값을 보인다. Quaternion with normalization은 간단한 normalization을 통해 unitary와 orthogonal constraints를 만족하는 것을 볼 수 있다.

그외 특이사항으로는 drawMesh()를 수행하기전에 glPushMatrix()을 통해 기존 transformation matrix를 저장하고, glMultiMatrixf()으로 rotation matrix를 곱해 변환된 transformation matrix로 mesh를 그리는 것을 볼 수 있다. 그 이후에는 glPopMatrix()를 통해 기존 transformation matrix를 원래대로 돌려놓는다.

## Practice 01. Basic rotation using angle-axis rotation:

## Practice 02. Basic rotation using quaternion:

angle-axis rotation

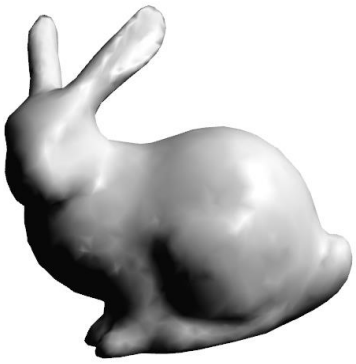
quaternion rotation

transformation rotation

세 rotation 모두 같은 모습을 보인다. 하지만 angle-axis와 quaternion은 transformation에 비해 위에서 설명한 것처럼 모든 original vertex와 normal을 바꿔 그리기 때문에 연산량이 매우 많고 느리다. 또한 quaternion는 vertex와 normal의 rotation을 위해 purely imaginary quaternion을 만들고, 여기에 앞 뒤로 quaternion과 inverse quaternion을 곱한 뒤 허수값을 뽑아낸다. 이러한 과정으로 인해 quaternion이 angle-axis 보다 더 연산량이 많고 느리다. 따라서 속도순으로 transformation > angle-axis > quaternion이다.

Practice 03. Incremental rotation using rotation matrix:

Practice 04. Incremental rotation using unit quaternion:


rotation matrix without normalization  rotation rotation with nomalization  quaternion

rotation matrix without normalization을 통해 Incremental rotation을 처음 수행했을 때 출력되는 값이다.

```
Unitary:        1, 1, 1
Orthogonal:     5.47152e-09, 3.59668e-09, -2.59897e-08
Determinant:    1
```

Rotation matrix의 모든 column의 크기(norm)는 모두 1로 Unitary하다.

세 column의 dot product의 값이 $10^{-9}$으로 0이라고 볼 수 있고 따라서 Orthogonal하다.

Rotation matrix의 determinant도 1로 출력된 것을 볼 수 있다.

Incremantal rotation을 수행하고 어느정도의 시간이 지난 뒤에 출력되는 값이다.

```
Unitary:        1.00191, 1.00149, 1.00177
Orthogonal:     -0.000248263, 0.000191942, 0.000335175
Determinant:    1.00518
```

Rotation matrix의 모든 column의 크기(norm)는 모두 1보다 크므로 Unitary하지 않다.

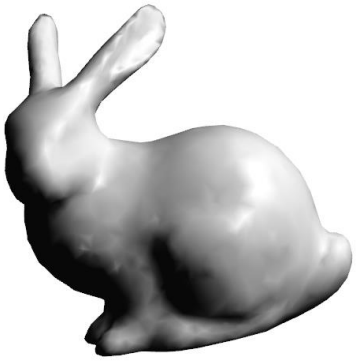
세 column의 dot product의 값이 $10^{-3}$정도로 0이 아니라고 볼 수 있고 따라서 Orthogonal하지 않다.

Rotation matrix의 determinant도 1보다 커진 것을 볼 수 있다.

이처럼 rotation matrix without normalization을 통해 Incremental rotation을 수행하면 오차가 점점 누적되어 matrix의 determinant가 증가하고 Unitary와 Orthogonal Contraint 모두 깨진 것을 볼 수 있다.
이를 방지하기 위해서는 rotation matrix에 normalization을 수행해야한다. 하지만 여기서 rotation matrix의 orthonormal을 지키려면 Gram-Schimidt Orthogonalization을 해야하고 이는 연산량이 많고 복잡하다.

그렇지만 Quaternin으로 Incremental rotation을 수행하면 renormalization을 위해 quaternion의 크기만을 구하면 되기 때문에 rotation matrix에 비해 normalization이 매우 쉽고 간단하다.

Exercise 01. Orthonormality of the rotation matrix with a simple normalization:


rotation matrix without normalization  rotation rotation with nomalization  quaternion

Incremental rotation을 처음 수행했을 때 출력되는 값이다.

```
Rotation matrix without normalization
Unitary:        1, 1, 1
Orthogonal:     5.47152e-09, 3.59668e-09, -2.59897e-08
Determinant:    1

Rotation matrix with simple normalization
Unitary:        1, 1, 1
Orthogonal:     5.47152e-09, 3.59668e-09, -2.59897e-08
Determinant:    1
```

Normalization을 수행한 rotation matrix와 수행하지 않은 rotation matrix 모두 Unitary와 Orthogonal constraint를 만족한다. 그리고 matrix의 determinant도 1이 나온 것을 확인할 수 있다.



rotation matrix without normalization  rotation rotation with nomalization  quaternion

Incremantal rotation을 수행하고 어느정도의 시간이 지난 뒤에 출력되는 값이다.

```
Rotation matrix without normalization
Unitary:        1.32954, 1.32933, 1.32137
Orthogonal:     0.00430583, 0.00521646, -0.00230594
Determinant:    2.33538

Rotation matrix with simple normalization
Unitary:        1.00078, 1.00045, 0.998779
Orthogonal:     0.00211289, 0.00294975, -0.00144458
Determinant:    1
```

Normalization을 수행하지 않은 rotation matrix은 Unitary와 Orthogonal constraint를 만족하지 않고

matrix의 determinant도 약 2.34로 1보다 큰 값이 나왔다.

Normalization을 수행한 rotation matrix는 Unitary constraint를 만족하고 matrix의 determinant도 1로 출력되었지만 Orthogonal constraint를 만족하지 않는 것을 볼 수 있다.

이는 여기서 rotation matrix를 normalization하기 위해 수행된 과정이 determinant만을 1로 맞추기 위해 rotation matrix를 determinant의 -1/3을 곱했기 때문이다.

Rotation matrix를 orthonormal을 지키면서 renormalization을 하기 위해서는 Gram-Schimidt orthonormalization을 수행해야 하지만 이는 연산량이 많고 복잡하여 해당 과제에서는 simple-normalization을 하였다.

결과로 나온 사진 중에 왼쪽을 통해 Unitary constraint가 깨지고 determiant가 1보다 커지면 Derterminant 만큼 부피가 Scaling되는 것을 볼 수 있다. 또 가운데와 오른쪽 사진을 비교해보면 Orthogonal을 만족하지 않을 때 rotation의 결과가 다른 것을 볼 수 있다. 이는 귀쪽을 잘보면 quaternion이 조금 더 큰 것을 확인할 수 있다.

결과적으로 **Basic**한 rotation을 수행할 때는 **transformation** matrix를 사용하여 opengl pipeline에서 적은 연산량으로 빠르게 돌리는 것이 좋을 것이고, **Incremental** rotation을 수행할 때는 renormalization이 간단한 **Quaternion**을 사용하는 것이 좋을 것이다.

이를 위해서 Eigen에서는 Quaternionf q(aa); Matrix3f R_from_q(q); Quaternionf q_from_R(R); AngleAxisf aa_from_q(q); AngleAxisf aa_from_R(R); 과 같은 Quaternion – Rotation matrix – Angle Axis rotation 간의 변환을 지원해준다.