시뮬레이션 기초 및 실습

HW 2

시뮬레이션 기초 및 실습

김지범 교수님

컴퓨터 공학부

202201529

정상혁

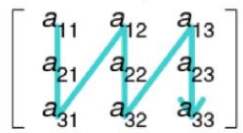
목차

I. 과제 2_1

a)

(a) OpenGL에서는 차원을 하나 높인 동차좌표를 사용하고 변환시에도 4x4 행렬을 사용한다. 4x4행렬을 정의할 수 있는 glm::mat4와 4x1벡터를 정의할 수 있는 glm::vec4를 아래 출력 결과와 같이 정의해보자. 4x4 행렬을 M, 4x1 벡터를 v라고 하자. 아래와과 같이 M과 v를 곱한 결과 Mv를 출력해보자. 단, glm에서 행렬을 정의시에 아래와 같이 column-major 순서로 정의되고 행렬 M과 벡터 v, 벡터 Mv를 모두 출력하고 결과를 리포트에 넣자.

Column-major order



$$M=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & -10\\ 0 & 0 & 0 & 1\end{bmatrix}, v=\begin{bmatrix}1\\2\\3\\1\end{bmatrix}$$

```
glm::mat4 matA = glm::mat4(1.0f); //4*4 단위행렬
matA[3][2] = -10.0;               //-10을 넣어줌
cout << "matrix A : ₩n";
for (int i = 0;i < 4;i++) {
    for (int j = 0;j < 4;j++) {
        cout << matA[j][i] << " ";
    }
    cout << endl;
}
```

가장 먼저 행렬 A를 선언해주었는데 문제에 나온 행렬의 특징은 단위행렬에 3행 4열에 -10이 들어가있는 행렬이다. 따라서 이 행렬을 만들기 위해 4*4 단위행렬을 만들어 준 다음

Glm의 경우 행부터 선언하기 때문에 우리가 평소에 사용하는 행렬과는 반대로 넣어주었다.

그리고 출력 역시 반대로 출력해주었다.

```
glm::vec4 vecB = glm::vec4(1.0f, 2.0f, 3.0f, 1.0f);
cout << "vector B : ₩n";
for (int i = 0;i < 4;i++) {
    cout << vecB[i] << endl;
}
cout << endl;
```

그리고 다음으로 4*1 벡터를 선언해주었는데 glm을 이용해 선언해주었고, 벡터 역시 for 문을 이용해 출력해주었다.

다음으로 matrix A와 vector B를 곱해서 나온 vector C를 계산해주었다.

```
glm::vec4 vecC = matA * vecB;
std::cout << "Matrix A * vector B:₩n";
for (int i = 0; i < 4; i++) {
    std::cout << vecC[i] << std::endl;
}
```

matrixA와 vectorB의 곱을 계산한 값을 vector C에 대입해주었고, 출력하는 코드를 짜주었다.

오른쪽 사진이 코드 실행 결과이다.

예상한 대로 출력결과가 잘 나온 것을 알 수 있었다.

```
matrix A :
1 0 0 0
0 1 0 0
0 0 1 -10
0 0 0 1

vector B :
1
2
3
1

Matrix A * vector B:
1
2
-7
1
```

I. 과제2_1

b)

(b) 선형대수에서 4x4 크기의 단위 행렬 (I)과 4x1 크기의 어떤 벡터(v)를 곱하면 자기 자신이 나온다. 이를 glm 라이브러리를 사용하고 v는 (a)에서의 v를 사용하자. 행렬 I와 벡터 v, 벡터 Iv를 모두 출력하고 결과를 리포트에 넣자.

```
glm::mat4 matI = glm::mat4(1.0f); //4*4 단위행렬
cout << "matrix I : ₩n";
for (int i = 0;i < 4;i++) {
    for (int j = 0;j < 4;j++) {
        cout << matI[j][i] << " ";
    }
    cout << endl;
}
```

단위 행렬을 선언해주고 출력해주었고, vector의 경우는 아까 a번에서 사용한 것을 사용했다.

그리고 나머지 코드는 a번의 코드와 같이 작성해주고 실행해보면

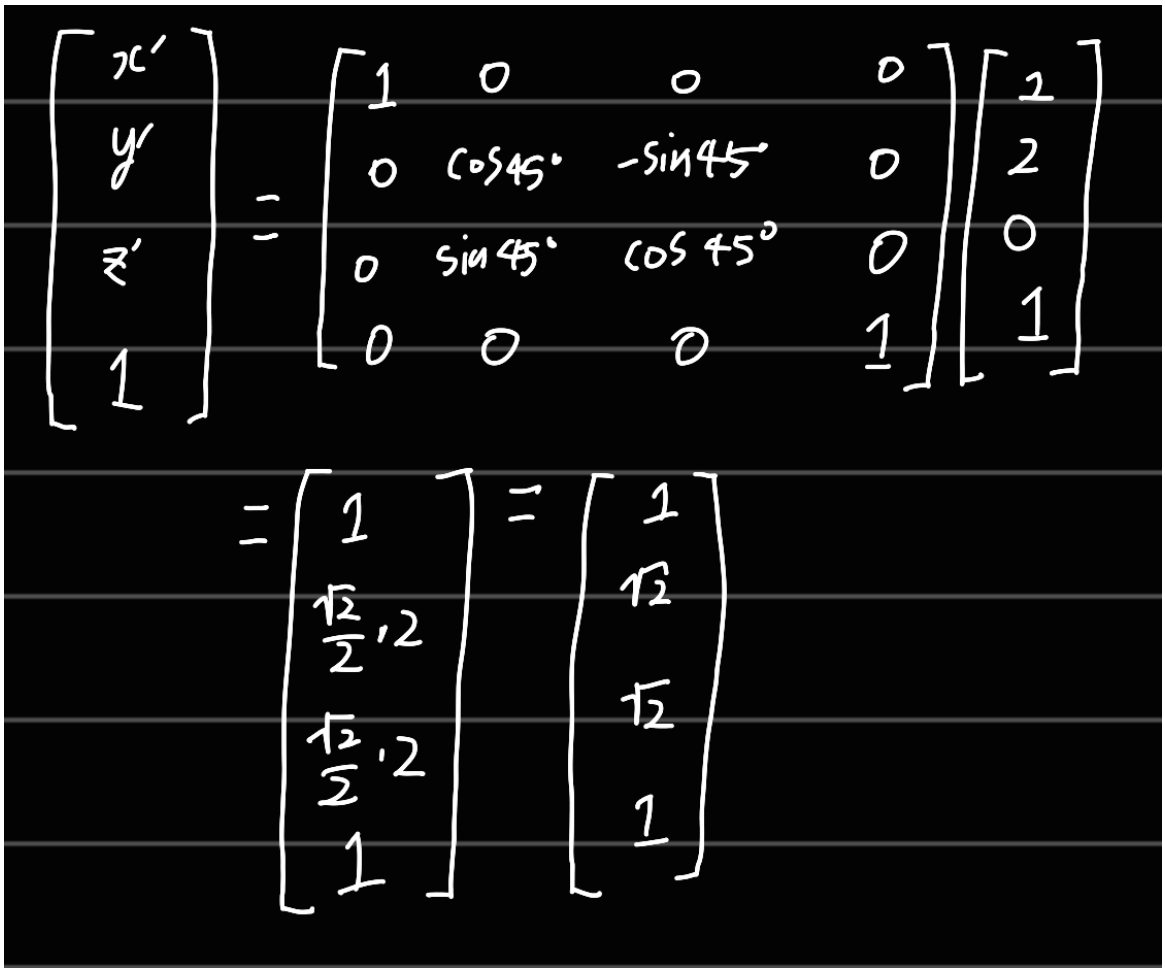
예상한 대로 vector와 단위행렬을 곱하게 되면 vector의 값이 변하지 않는다는 사실을 알 수 있었다.

```
matrix I :
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

vector B :
1
2
3
1

Matrix I * vector B:
1
2
3
1
```

II. 과제2_2

a)

(a) 점 (1, 2, 0)이 x축 기준으로 CCW (반 시계 방향)으로 45도 회전했을 때 4x4 변환행렬을 손으로 구해보자.



X축 기준으로 변환행렬을 계산하기 위하여 행렬 곱을 계산해주었다.

그러자 위와 같은 결과가 나왔다.

b)

(b) (a)와 동일한 결과를 glm library를 써서 구현해 보고자 한다. Lecture 6의 glm의 rotation 예제 코드 (glm::rotate)를 사용하여 1)의 결과를 콘솔에 출력해보자. 1)과 결과가 동일한가? rotation 코드를 작성시에 아래 헤더를 추가하자.

```
// 4x4 단위 행렬 생성
glm::mat4 trans = glm::mat4(1.0f);
// x축 기준으로 반시계 방향 45도 회전 변환 행렬
trans = glm::rotate(trans, glm::radians(45.0f), glm::vec3(1.0f, 0.0f, 0.0f));
// (1, 2, 0) 좌표를 회전
glm::vec4 point(1.0f, 2.0f, 0.0f, 1.0f);
glm::vec4 result = trans * point;
// 결과 출력
printf("Rotated Point: %f, %f, %f ₩n", result.x, result.y, result.z);
```

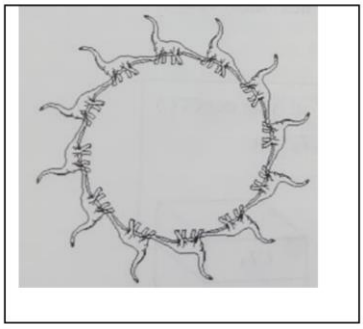
단위행렬을 선언해주고 glm라이브러리에 있는 함수를 이용하여 코드를 작성해주었다. 그리고 출력해주게 된다면 아래와 같은 출력 결과가 나오게 된다.

```
Rotated Point: 1.000000, 1.414214, 1.414214
```

손으로 계산한 값이 루트 2이고, 루트 2는 1.414214로 근사될 수 있으므로 이 둘의 값은 같게 나왔다고 볼 수 있다.

III. 과제 2_3

3. 수업시간에 'dino.dat' 파일을 사용하여 dinosaur를 그린 적이 있다. 이를 기본으로 수업시간에 배운 CTM과 복합 변환을 이용하여 아래 그림 왼쪽과 같은 모양을 만들어 보자. 각각의 dinosaur는 회전된 형태로 각각의 발은 원점을 향하는 모양이다. 리포트에 어떻게 CTM을 이용하여 그렸는지 자세히 설명하자.



```
int numDinos = 12; // 공룡의 개수
float angleStep = 360.0f / numDinos; // 공룡 간 각도 차이
float distance = 100.0f; // 공룡 발 위치에서의 거리

for (int i = 0; i < numDinos; i++) {
    float angle = i * angleStep; // 각 공룡의 각도 계산
    drawTransformedDino("dino.dat", angle, distance);
}
```

가장 먼저 공룡의 개수를 정하여 변수에 넣어주었고, 공룡의 개수만큼 360도에서 나누어서 한 마리당 각도를 정해주었고, 함수 인자를 이용해 넘겨주었다.

```
// 기본 단위 행렬 생성
glm::mat4 transform = glm::mat4(1.0f);
```

가장 먼저 기본 단위 행렬을 생성해주었다.

넘겨받은 인자들을 사용하여, glm라이브러리를 이용해 변환해주었다.

```
transform = glm::scale(transform, glm::vec3(0.1f, 0.1f, 1.0f)); // 크기 조정
```

glm라이브러리는 아래서부터 계산하기 때문에, 아래서부터 서술하자면, 가장 먼저 공룡의 크기를 작게 변환해주었고, 위치를 이동해주었다.

```
transform = glm::translate(transform, glm::vec3(-20.0f, distance, 0.0f)); // 공룡 위치 이동
```

여기서 x방향으로 -20만큼 이동한 이유는 만약 x축을 이동하지 않는다면, 공룡의 왼쪽아래를 기준으로 공룡이 위치하기 때문에 조금 어긋나 보일 수 있다. 그래서 x축 이동을 이용해 조금 보정해주었다.

```
transform = glm::rotate(transform, toRadians(angle), glm::vec3(0.0f, 0.0f, 1.0f)); // 회전
```

그리고 z축을 중심으로 입력 받은 각만큼 회전을 시켜주었다.

```
transform = glm::translate(transform, glm::vec3(400.0f, 400.0f, 0.0f)); // 화면 중심 이동
```

그리고 마지막으로 중심점으로 이동시켜 주었다.

```
// 변환 적용
transform = glm::translate(transform, glm::vec3(400.0f, 400.0f, 0.0f)); // 화면 중심 이동
transform = glm::rotate(transform, toRadians(angle), glm::vec3(0.0f, 0.0f, 1.0f)); // 회전
transform = glm::translate(transform, glm::vec3(-20.0f, distance, 0.0f)); // 공룡 위치 이동
transform = glm::scale(transform, glm::vec3(0.1f, 0.1f, 1.0f)); // 크기 조정
```

실제 적용 순서는 이렇게 되지만 실제 코드 작성 순서는 역순임을 주의 해야 한다.

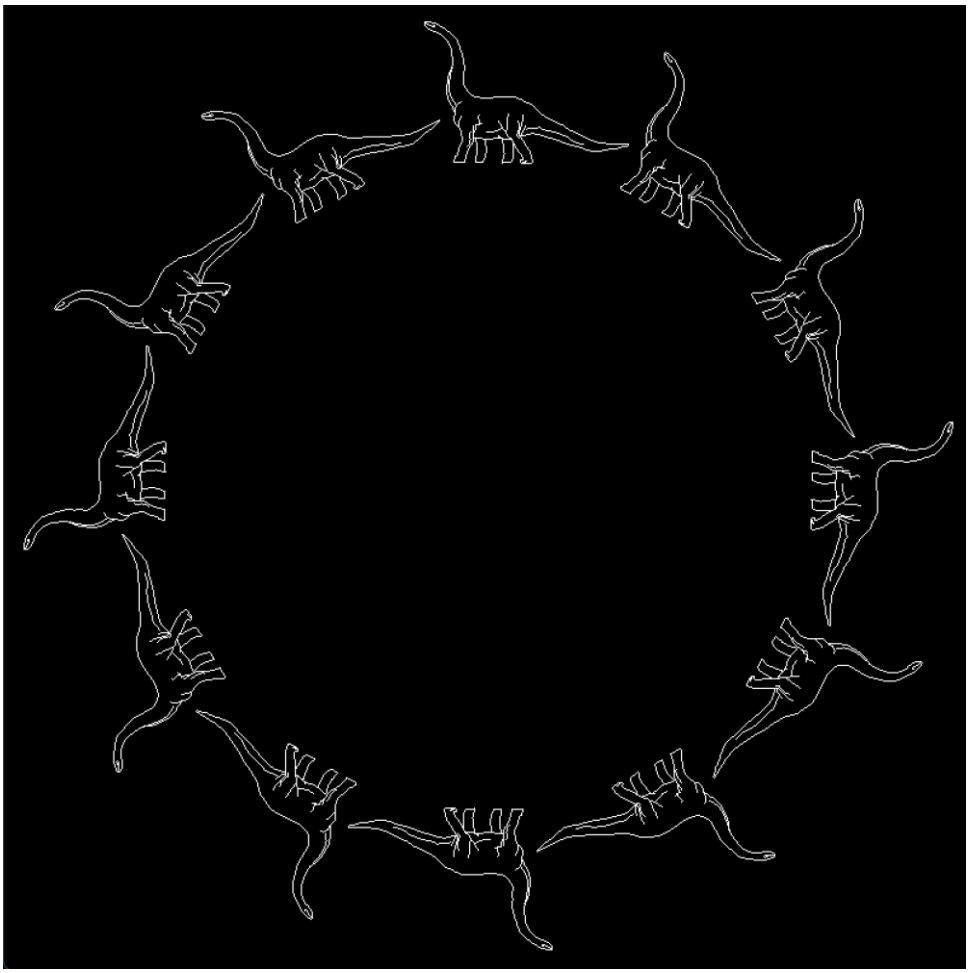
```
// 변환된 행렬을 OpenGL에 전달
glPushMatrix();
glMultMatrixf(glm::value_ptr(transform));
```

그리고 이렇게 변환된 행렬을 openGl에 전달해주었다.

그리고 공룡을 그려주었다.

그리고 아래는 실행한 뒤 출력결과이다.

예측하고, 설계한 대로 출력이 잘 그려진 것을 확인 할 수 있다.



IV. 과제 2_4

a)

4. 아래와 같이 6개의 color로 나눠진 과녁판 같은 모양을 transformation을 사용하여 만들고자 한다.

(a) GL_TRIANGLE_FAN을 사용하여 아래 그림에서 빨간색 부분을 그려보자. 빨간색 부분을 여러 삼각형으로 triangulation 하면 그런 모양이 가능하다. 출력 결과를 보여주고 리포트에 어떻게 구현했는지 설명해보자.

```
#define M_PI 3.14159265358979323846
#define NUM_SEGMENTS 100  // 원을 근사할 세그먼트 수
```

가장 먼저 몇 개의 삼각형으로 근사할 것인지를 정해주었다.

```
glBegin(GL_TRIANGLE_FAN);
glColor3f(1.0f, 0.0f, 0.0f); // 선 색상: 빨간색
glVertex2f(0.0f, 0.0f); // 중심점
for (int i = 0; i <= NUM_SEGMENTS; i++) {
    float angle = (M_PI / 3) * i / NUM_SEGMENTS; // 각도 계산
    float x = cos(angle); // x좌표
    float y = sin(angle); // y좌표
    glVertex2f(x, y);     // 정점 추가
}
```

그리고 각도를 계산하여 60도까지 100개의 각도로 나눈 뒤에 100개의 삼각형을 그려준다면, 다음과 같은 모양이 나오게 된다.

b)

(b) (a)의 결과를 6번 반시계 방향으로 rotate하면서 아래 모양을 완성해보자. glRotatef 함수를 사용하자. 출력 결과를 보여주고 리포트에 어떻게 구현했는지 설명해보자.

```
for (int j = 0; j < 6; j++) { // 6번 회전하여 도형 그리기
    glPushMatrix(); // 현재 변환 행렬 저장
    glRotatef(j * 60.0f, 0.0f, 0.0f, 1.0f); // z축을 중심으로 회전

    // 회전된 위치에 도형을 그릴 위치 설정
    drawShape(j); // 색상을 바꿔가며 도형 그리기

    glPopMatrix(); // 변환 행렬 복원
}

glFlush();
```

drawShape()함수가 바로 위의 도형을 그리는 함수이다.

For문을 이용해, 60도씩 6번 회전시키며 색깔도 바꾸어 가며 그려주었다.