20140559 조형준, HW2

1. Affine Transform Factorization

1.1 linear Fact

```
glm::mat4 linearFact(glm::mat4 A)
{
	glm::mat4 L = A;
	L[0][3] = 0;
	L[1][3] = 0;
	L[2][3] = 0;
	return L;
}
```

1.2 trans Fact

```
glm::mat4 transFact(glm::mat4 A)
{
	glm::mat4 T = glm::mat4(1.0f);
	T[0][3] = A[0][3];
	T[1][3] = A[1][3];
	T[2][3] = A[2][3];
	return T;
}
```

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b & c & 0 \\ e & f & g & 0 \\ i & j & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

를 그대로 구현하였다

2. Object Manipulation with Auxiliary Frame

우선 mouse right button (translation)을 구현할 때는 A = $(O)_T(E)_R$를 적용하였다. 이를 위해 get_aFrame이라는 함수를 정의해서 translation을 할 때마다 aFrame을 계산해주었다.

```
static void get_aFrame(void)
{
	glm::mat4 O_t, E_r;
	O_t = transFact(arcballRBT_i);
	E_r = linearFact(eyeRBT);
	aFrame = O_t * E_r;
}
```

이후

```
skyRBT = aFrame * glm::translate(glm::mat4(1.0f), arcBallScale * glm::vec3(xpos - mouse_xpos, -(ypos - mouse_ypos), 0.0)) * glm::inverse(aFrame) * skyRBT
```

를 각각의 case에 맞게 skyRBT, g_objectRbt[0], g_objectRbt[1]로 바꿔주면서 translation을 처리해주었다. (O <- $AMA^{-1}O$ 를 적용하였다.)

mouse left button (rotaion)을 구현할 때는

```
aFrame = glm::translate(glm::mat4(1), glm::vec3(arcballRBT_i[3][0], arcballRBT_i[3][1],
arcballRBT_i[3][2]));
```

를 적용해주었고 skyRBT를 변환할 때는 aFrame을 적용할 필요가 없었기에

```
skyRBT = rotation * skyRBT;
```

를 적용하였고, g_objectRbt[0], g_objectRbt[1]를 변환할 때는

```
g_objectRbt[0] = aFrame * rotation * glm::inverse(aFrame) * g_objectRbt[0];
g_objectRbt[1] = aFrame * rotation * glm::inverse(aFrame) * g_objectRbt[1];
```

를 적용하였다.

3. Arcball Interface

```
init_sphere(arcBall);
arcBall.initialize(DRAW_TYPE::INDEX, "VertexShader.glsl", "FragmentShader.glsl");
arcBall.set_projection(&Projection);
arcBall.set_eye(&eyeRBT);
arcBall.set_model(&arcballRBT);
```

를 이용해 arcBall을 initialize해주었다.

draw part에서는 arcBall이 GL_LINE으로 그려져야 하므로

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE); // draw wireframe
arcBall.draw();
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // draw filled models again
```

위 코드처럼 처리해주었다.

Interaction with mouse를 구현하기 위해 mouse_mode라는 변수를 선언했고 mouse_button_callback에서 이를 변화시켜주었다.

이후 cursor_pos_callback에서 mouse_mode값을 switch-case문에 적용하였다.

rotation의 경우 우선적으로 z값을 구하기 위해 compute_z라는 함수를 정의하였는데 여기서 eye_to_screen함수를 이용해 ball의 center (스크린상) 좌표를 구하였다. 이후 $z^2 = r^2 - x^2 - y^2$를 이용해 z값을 구해주었다. 이후 start, dest vector를 구하여 normalize하였고 이를 이용해 start-center-dest각도를 구해주었고, quat를 위한 k도 glm::cross(start,dest)를 이용해 구해주었다. 이 quat를 mat4_cast함수를 이용해 rotation matrix로 변환하였고, 2번 항목에서 설명한 것과 같은 방식으로 rotation을 적용하였다.

translation의 경우는 따로 처리해줄 필요없이 2번 항목에서 설명한 것과 같은 방식으로 aFrame을 구하고 O <- AMA_1O 를 적용해주었다.

arcBall의 scale 조정을 위해서 arcballRBT이외에 arcballRBT_i라는 변수를 선언해주었다. 이를 선언한 이유는 do-while문에서 arcballRBT라는 변수만으로 scale 조정을 실행하게되면 callback함수가 불러질때마다 re-scale되어 크기가 무한정 작아지는 문제가 발생하기 때문으로, scale이전에는 모든 part에서 arcballRBT_i를 변화시키고, 마지막으로

arcballRBT = arcballRBT_i * glm::scale(glm::vec3(arcBallScreenRadius * arcBallScale))

를 적용하여 이 문제를 해결하였다.

위 식에 있는 arcBallScale을 구하기 위해

```
if (mouse_mode == 0)
{
        arcBallScale = compute_screen_eye_scale((glm::inverse(eyeRBT) * (arcballRBT_i[3]))[2],
fovy, int(windowHeight));
        if (select_object == 0)
                arcBallScale = compute_screen_eye_scale((glm::inverse(eyeRBT) *
worldRBT[3])[2], fovy, int(windowHeight));
}
```

를 코드에 넣어주었다. mouse_mode == 0이라는 조건은 마우스가 클릭 되어있지 않은 상황을 나타낸 것이다. compute_screen_eye_scale의 첫 변수로 `(glm::inverse(eyeRBT) * (arcballRBT_i[3]))[2]`과 `(glm::inverse(eyeRBT) * worldRBT[3])[2]`를 넣어준 이유는 현재 eyeRBT를 기준으로 arcball의 z좌표를 구해 scale을 진행하기 위함이다.

4. Correct Implementation for Mouse and Keyboard Callbacks

document에 주어진 대로 적용하였으나 rotation부분에서 document에 헷갈리는 점이 있어 rotation 방향이 solution과 다를 수 있다고 생각한다.

6. Creativity

cube의 scaling을 넣어주었다. 기존에는 mouse middle button이 click되어있을 때 y좌표의 변화를 계산해 arcball의 z좌표를 변화시켜주었는데, x좌표의 변화를 이용해 cube의 scale을 변화시키는 함수를 추가하였다. cube의 scale을 알아보기위해 해당 cube에 scale에 비례하여 arcball의 scale도 변화시켜주었다 (object를 변환할 때 cube의 scale을 인식해 cube의 scale에 맞는 크기로 자동 변환한다.) cube의 scaling은 windowWidth에 기반을 두고 처리하였는데 x좌표를 계속해서 -방향으로 이동시키면 – scaling이 일어난다. 원래는 이를 방지하려고 했으나 – scaling이 된 cube의 모습이 흥미로워서 건드리지 않았다.