기초 컴퓨터 그래픽스

HW 1 – GLUT을 이용한 2차원 아핀 변환의 구현

20181625 남현준

1. **프로그램 작성 및 실행환경**

본 HW 1의 작성 및 구동은 Windows 10 64bit, AMD Ryzen 5 5600X, NVIDIA GeForce RTX 3070, Visual Studio Release x64에서 진행되었습니다.

1. **HW 1 요구사항** 
   1. **윈도우 화면의 초기설정**

-구현 여부 : 예

-확인 방법 : Readme와 함께 주어진 cpp파일을 성공적으로 컴파일 했을 경우, 프로그램을 실행하게 되면 윈도우가 화면에 나타나게 된다.

-구현 방법:

텍스트이(가) 표시된 사진

자동 생성된 설명

[그림1] 윈도우 설정에 대한 코드부분

Main 함수 내부에서 glutInitWindowSize()와 glutInitWindowPosition() 함수를 호출하여 실행하게 될 프로그램 윈도우의 초기 가로/세로 길이와 위치를 설정해주었다. 그 후, glutCreateWindow() 함수를 호출하여 설정된 크기 및 위치를 가진 윈도우를 생성하여 화면에 해당 설정을 가진 창을 만든다.

* 1. **선분의 위치 및 형태설정**

**-**구현 여부 :예

**-**확인 방법 : A에서 구현한 윈도우가 성공적으로 화면에 나타나게 되었다면, Salmon 색의 배경색과 함께 x, y 축, 파란 점과 흰색 점을 가지는 선분, 파란색의 중심점과 흰색 꼭짓점, 연두색 선분을 가지는 다각형이 같이 화면에 출력된다.

텍스트이(가) 표시된 사진

자동 생성된 설명**-**구현 방법 :

[그림2] 선분의 초기설정에 대한 코드부분

Main 함수에서 호출되는 initialize\_renderer() 함수 내부에서 선분, xy축, 다각형의 초기 정보가 주어지게 된다. 과제 소스 파일에서 기본적으로 값이 주어진 상태로 주어지는 px, py, qx, qy를 임의의 값으로 assignment를 통해 적당한 값을 할당해주었다.

* 1. **비대칭 다각형의 위치, 크기 및 형태 설정**

-구현 여부 : 예

-확인 방법 : A에서 구현한 윈도우가 성공적으로 화면에 나타나게 되었다면, Salmon 색의 배경색과 함께 x, y 축, 파란 점과 흰색 점을 가지는 선분, 파란색의 중심점과 흰색 꼭짓점을 가지는 다각형이 같이 화면에 출력된다.

-구현 방법 :

텍스트이(가) 표시된 사진

자동 생성된 설명

[그림3] 다각형의 초기설정에 대한 코드부분

B에서 구현한 선분과 마찬가지로, main 함수에서 호출되는 initialize\_renderer() 함수 내부에 다각형과 관련된 초기 정보가 주어지게 된다. 소스 코드에서 주어진 값들을 임의의 값으로 assignment를 통해 적당한 값으로 바꿔주었다.

* 1. **마우스 휠의 움직임을 통한 선분의 회전 구현**

-구현 여부 : 예

-확인 방법 : 마우스의 휠을 위로 돌리면 그에 맞춰서 선분의 흰색 꼭짓점이 파란색 점을 기준으로 반시계방향으로 회전하게 된다. 마우스의 휠을 아래로 돌리면 선분의 흰색 꼭짓점이 파란색 점을 기준으로 시계방향으로 회전하게 된다.

-구현 방법 :

텍스트이(가) 표시된 사진

자동 생성된 설명

[그림4] 마우스 휠의 움직임에 따른 선분의 회전 코드부분

2차원 아핀 변환을 통해 해당 기능을 구현하기 위해 수업에서 제공되는 강의자료의 composition of 2D transformations 부분을 참조하였다.

흰색 꼭짓점을 파란색 점을 기준으로 회전시키기 위해서는 우선 주어진 선분의 회전 기준점을 원점을 중심으로 하게 할 필요가 있었다. 그를 구현하기 위해서 2차원 아핀 변환 중 하나인 translation의 행렬을 이용할 필요가 있었고 이를 위해 사용자 선언 함수인 create\_Matrix\_T() 함수를 선언하여 해당 행렬을 구할 수 있게 하였다. 이 기능에서 선분은 파란색 점을 기준으로 회전하게 되므로 파란색 점과 흰색 점에서 파란색 점의 x, y 좌표만큼의 값을 빼주어서 선분이 원점을 중심으로 하도록 translation을 시켜야한다. 해당 부분을 x좌표와 y좌표의 값을 가질 수 있는 구조체 point를 선언한 다음, 파란색 점의 정보를 구조체를 선언하여 저장했다. 그 후, 해당 값을 이용하여 translation 행렬을 생성하게 된다.

마우스 휠의 작동의 경우 검색을 통해 기본적으로 주어진 함수들 외에 glutMouseWheelFunc() 함수와 void mousewheel()이라는 함수가 필요하다는 것을 알았다. Void mousewheel()을 선언해주고 glutMouseWheelFunc() 함수를 register\_callbacks() 함수 내부에 선언하여 프로그램의 다른 기능들과 마찬가지로 프로그램이 작동되는 동안 user input을 통해 대응되는 output를 반환할 수 있도록 해주었다.

그 후, 휠의 움직임에 따라 선분이 원점을 기준으로 반시계 혹은 시계방향으로 회전하게 된다. 이 기능을 위해 rotation 행렬을 구하기 위한 사용자 선언 함수 create\_Matrix\_R()을 구현해주었다. 휠의 움직임은 위와 아래로 주어질 수 있고, 이와 같은 경우들에 대응할 수 있도록 if문을 통해 휠의 움직임이 주어졌을 때 의도한 것과 같이 시계/반시계 방향으로 회전할 수 있도록 구분해주었다.

회전이 성공적으로 진행되었다면 이전의 기준점, 즉 주어졌던 선분의 파란색 점의 초기 위치를 중심으로 하도록 다시 파란점의 자료를 이용해 translation 행렬을 만든 다음 행렬의 곱셈을 통해 기준점을 옮겨 주는 작동을 하도록 하였다. 해당 부분은 행렬의 곱셈을 진행시키는 함수를 따로 만들어 처리를 했으며 translation, roation, 다시 translation을 통해 얻을 결과값을 qx, qy, 즉 흰색 점의 값에 assign하여 값을 갱신하는 것으로 마무리된다. 이후, glutPostRedisplay()를 통해 갱신된 값을 이용하여 화면 상에 출력하게 된다.

* 1. **선분의 파란색 꼭짓점의 이동 구현**

-구현 여부 : 예

-확인 방법 : 구동된 프로그램의 화면 상에 그려진 선분의 꼭짓점 중 파란색 점을 왼쪽 마우스 버튼으로 클릭했을 때, shift키가 눌러져 있다면 파란색 점을 마우스의 커서의 위치에 따라 이동시킨다.

-구현 방법 :

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 실내, 스크린샷, 은이(가) 표시된 사진

자동 생성된 설명 [그림5] 마우스 좌클릭에 대한 코드부분

[그림6] 좌클릭, shift키에 대한 코드부분

마우스의 클릭을 담당하는 mousepress() 함수에서 우선 마우스 좌클릭이 진행되었을 때, 파란색 점의 위치를 클릭했는지를 판별한다. 이를 위해 Boolean type의 변수 picked\_blue를 선언해주어 파란점의 클릭 여부를 알 수 있게 하였다. 파란색 점을 클릭했는지의 판단 기준은 사용자 입력을 통해 클릭된 위치의 근사값이 x, y좌표를 기준으로 임의의 숫자로 사용한 0.005f의 절대값보다 작다면 클릭을 제대로 했다는 의미에서 picked\_blue 변수의 값을 true로 할당해주었다. 이외의 경우에는 picked\_blue가 false로 반환이 되며 해당 기능은 작동하지 않는다. 좌클릭의 경우에도 mousepress() 함수 내부에서 좌클릭 입력이 들어오게 되었을 때, leftbuttonpressed 변수의 값을 1로 해주었고, 그 외의 경우는 leftbuttonpressed 변수의 값이 0이 되어 후술할 함수에서 기능들이 작동하지 않게 된다.

Leftbuttonpressed 변수의 값이 1, 즉 true일 때 mousemove() 함수 내부에서 glutGetModifiers() 함수를 통해 SHIFT키가 눌려졌는지 확인을 한다. 만약 SHIFT키가 눌러졌다면 다음 동작을 수행하게 된다. 해당 부분은 glutGetModifers()를 통해 ctrl, shift, alt키의 입력여부를 알 수 있는 함수를 통해 값을 받은 뒤, if문을 통해 GLUT\_ACTIVE\_ALT가 true라면, 다음과 같은 작동을 하게 된다. 현재 마우스 커서의 위치를 x와 y에 저장하게 되고 prevx와 prevy에 커서가 움직이기 전의 x y좌표의 값을 저장한다. X와 y에서 prevx와 prevy 만큼의 값을 빼주어서 dx, dy, 즉 x방향과 y방향으로 움직인 거리를 구한다. Dy의 경우, freeGLUT에서 사용하는 좌표계에서 y좌표의 양의 방향은 real world와 반대가 되므로 -1의 값을 곱해주었다. 그 후, 해당 값들을 이용하여 사용자 선언 함수 create\_Matrix\_T() 함수를 통해 translation 행렬을 만들어주었다. 만약, 위에서 정의된 picked\_blue 변수가 true라면, 파란색 점의 x,y 좌표에 아까 구해준 translation 행렬을 곱해 새로운 좌표의 위치를 구한 뒤, px, py, 프로그램 상에서 사용되는 파란색 점의 x, y좌표 변수에 새로 구하게 된 값을 assignment를 통해 할당해주는 것을 통해 파란색 점의 클릭에 따른 이동을 구현하였다. 이후 glutPostRedisplay()를 통해 갱신된 값을 화면 상에 출력한다.

* 1. **비대칭 다각형의 이동 구현**

**-**구현 여부 : 예

**-**확인 방법 : 마우스 우클릭이 사용자 입력으로 들어오게 되었을 때, alt키 또한 눌러져 있는 상태라면 프로그램의 화면 상에 그려져 있는 연두색 선분을 가진 다각형이 마우스의 움직임에 따라 이동하게 된다.

**-**구현 방법 :

텍스트이(가) 표시된 사진

자동 생성된 설명

[그림7] 우클릭 구현에 대한 코드부분

텍스트이(가) 표시된 사진

자동 생성된 설명 [그림8] 우클릭, alt에 대한 코드부분

마우스의 클릭을 담당하는 mousepress() 함수에서 마우스 우클릭이 입력으로 들어왔을 경우, rightbuttonpressed 변수의 값을 1, 즉 true로 해주었다. 이외의 경우에는 rightbuttonpressed 변수의 값은 0이 된다.

Mousepress() 함수에서 반환된 rightbuttonpressed 변수의 값이 1이라면 다음과 같은 작동을 하게 된다. glutGetModifiers() 함수를 통해 alt키가 눌려진 상태인지 확인하고 분기문을 통해 다음과 같은 작동을 수행한다. alt키가 눌러진 상태라면 현재 마우스 커서의 위치를 x와 y에 저장하게 되고 prevx와 prevy에 커서가 움직이기 전의 x y좌표의 값을 저장한다 (prevx, prevy의 초기값은 0이다). X와 y에서 prevx와 prevy 만큼의 값을 빼주어서 dx, dy, 즉 x방향과 y방향으로 움직인 거리를 구한다. Dy의 경우, freeGLUT에서 y좌표의 양의 방향은 real world와 반대가 되므로 -1의 값을 곱해주었다.

그 후, 해당 값들을 이용하여 사용자 선언 함수 create\_Matrix\_T() 함수를 통해 translation 행렬을 만들어주었다. 이후, 다각형의 x,y좌표의 정보가 저장되어 있는 object 배열을 for 반복문을 통해 접근하여 각 정보에 translation 행렬을 곱한 뒤 해당 값들로 object 배열의 정보를 갱신해주었다. 중심점의 경우에도, 다각형의 점들이 새로운 값으로 갱신됨에 따라 같이 갱신되어야 하기 때문에 dx, dy의 값으로 구한 translation 행렬을 마찬가지로 곱해준 다음 값을 생긴하여 마우스의 움직임에 따라 이동될 수 있도록 하였다. 이후, glutPostRedisplay()를 이용해 갱신된 값을 화면 상에 출력하게 된다.

* 1. **비대칭 다각형의 크기변환 구현**

**-**구현 여부 : 예

**-**확인 방법 : 마우스 우클릭이 사용자 입력으로 들어오게 되었을 때, control 키 또한 눌러져 있는 상태라면 프로그램의 화면 상에 그려져 있는 다각형이 마우스의 움직임에 따라 크기가 커지거나 줄어들게 된다.

**-**구현 방법 :

텍스트, 모니터, 스크린샷, 은이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

[그림9] 다각형의 scaling 구현에 대한 코드부분

마우스의 클릭을 담당하는 mousepress() 함수에서 마우스 우클릭이 입력으로 들어왔을 경우, rightbuttonpressed 변수의 값을 1, 즉 true로 해주었다. 이외의 경우에는 rightbuttonpressed 변수의 값은 0이 된다.

Mousepress() 함수에서 반환된 rightbuttonpressed 변수의 값이 1이라면 다음과 같은 작동을 하게 된다. glutGetModifiers() 함수를 통해 control키가 눌려진 상태인지 확인하고 분기문을 통해 다음과 같은 작동을 수행한다. control키가 눌러진 상태라면 마우스가 x축을 기준으로 했을 때, 양의 방향으로 움직였는지 음의 방향으로 움직였는지 체크하기 위해 dx의 값을 구해준다.

Dx의 값은 현재 마우스 커서의 좌표 x에서 이전의 위치를 저장하고 있는 prevx의 값을 빼주어서 움직인 거리를 구하게 된다. Dx의 값을 구했다면 우선 scaling 아핀 변환을 진행하기 전에 다각형이 원점을 중심으로 하게 할 필요가 있다. 이를 위해, 다각형의 모든 점을 다각형의 중심의 x,y표만큼 이동시켜줄 필요가 있다. 해당 작동을 위해 주어져 있는 다각형의 중심의 x,y좌표인 object\_center\_y와 object\_center\_x를 이용해 translation 행렬을 만들어주었다.

이후, 다각형의 중심점을 포함한 저장되어 있는 다각형의 x,y 좌표값들에 대해 object\_center\_y와 object\_center\_x만큼씩 이동시켜 주었다. 이 때, object\_center\_y와 object\_center\_x의 거리만큼 원점을 향해 이동시켜야 하므로 다각형의 모든 점에서 해당 수치만큼씩 빼준 것이 결과값이 된다. 다각형의 모든 점에 앞서 구한 translation 행렬을 곱해주어 원점을 중심으로 하는 다각형이 되도록 했다.

이 부분 이후에 작동되는 부분은 한 가지만 빼면 동일한 방식으로 진행된다. 앞서 구한 dx 값을 활용하여 scaling을 진행하게 되는데, dx의 값에 따라 scaling 행렬을 생성하는 사용자 선언 함수 create\_matrix\_S()에 인자로 주어지는 dx값이 달라지게 된다. 양의 방향으로 움직였을 경우에는, 기존의 x, y좌표에서 x의 변화량에 x, y값을 곱해 기존의 x, y값에 더해주었다.

이후에 해당 값들에서 x, y의 수치로 나눈 값이 scaling factor가 된다. 이런 계산이 된 이유는 바뀐 x,y 좌표의 값을 기존 x,y의 수치로 나눠줘야 수치가 몇 배 증가했는지 알 수 있기 때문이다. 음의 방향으로 움직였을 경우에도 동일한 로직으로 scaling 행렬을 구하게 되는데 이 때, x와 y좌표의 scaling factor는 양의 방향으로 움직였을 때의 역수가 된다.

Scaling 행렬을 구했다면, for() 반복문을 통해 다각형의 모든 좌표의 정보에 대해 해당 행렬의 값을 곱해준 다음 값을 갱신해주었다. 중심점의 경우, 단순히 scaling만 진행되기 때문에 갱신해 줄 필요가 없다. 원점을 중심으로 하는 다각형에 scaling 행렬을 곱한 다음, 다시 기존에 중심으로 하던 점을 기준으로 만들어야 하기 때문에 중심점의 좌표를 인자로 가지는 translation 행렬을 생성하여 해당 값을 scaling 계산 후에 나오는 값에 곱해준다. 이후, glutPostRedisplay()를 통해 갱신된 값을 가지는 다각형을 화면 상에 출력한다.

* 1. **그 외 추가 구현들에 대한 설명**

**-**구현 여부 : 예

**-**확인 방법 : 프로그램 상에서 마우스 휠에 해당하는 부분을 누른 다음 마우스를 움직이면 위의 항목들에서 사용된 선분과 다각형이 아닌, 프로그램 상에 그려져 있는 나머지 다각형(코드 부분에서 polygon이라는 이름으로 정의하였음)이 회전하면서 크기가 커지거나 작아지게 된다. 이 때, 마우스가 양의 y축 방향으로 움직였는지, 음의 y축 방향으로 움직였는지에 따라 결정되게 된다. 마우스 좌클릭이 입력으로 주어졌을 때, alt키가 눌러져 있는 상태라면, 마우스 커서의 위치가 변함에 따라 해당되는 다각형이 해당 x,y 방향으로 잡아 당겨지게 된다 (shear function).

-구현 방법 :

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

[그림10] 휠 부분을 눌렀을 때, polygon의 회전 및 scaling에 대한 코드부분

텍스트이(가) 표시된 사진

자동 생성된 설명

[그림11] polygon의 shear 아핀 변환에 관한 코드부분

회전하면서 크기가 변동시키는 작동의 경우, 우선, 해당 다각형을 원점을 중심으로 하게 할 필요가 있다. 따라서, 다각형의 중심점을 구한 뒤, 해당 x, y좌표의 거리만큼 원점을 향해 이동시켜 줄 필요성이 있다. 이를 위해서, 해당 부분에 해당되는 center\_of\_polygonx와 center\_of\_polygony의 거리만큼 이동해야 하기에 -center\_of\_polygonx, -center\_of\_polygon를 사용자 선언 함수 create\_Matrix\_T()의 인자로 넘겨주어 해당 동작에 맞게 translation 행렬을 정의하였다.

그 후, polygon의 모든 점에 대해 translation 행렬을 곱해주어 원점을 중심으로 하는 polygon이 되게 하였다. 회전의 경우, 마우스 커서가 움직인 거리, 즉 dy의 값을 기준으로 회전과 scaling이 진행된다. 따라서, 해당 작동 부분에 y - prevy를 정의해주어 dy의 값을 미리 구했다.

기본적으로 polygon은 반시계방향으로 회전하게 되며 dy의 값이 양수라면, 반시계방향으로 회전하면서 다각형의 크기가 커지고, 음의 값이라면 반시계방향으로 회전하면서 polygon의 크기가 작아지도록 구현했다.

회전에 해당되는 행렬을 사용자 선언 함수 create\_Matrix\_R()로 생성할 때, dy의 값에 상관없이 반시계방향으로 회전하므로 해당 상황에 맞게 rotation\_of\_angle \* TO\_RADIAN의 값을 이용해 rotation 행렬을 만들었다. 그 후, 원점으로 이동된 다각형의 모든 점에 상황에 맞는 회전 행렬을 곱해주어 원점을 기준으로 의도한 각도와 방향으로 다각형을 회전시키게 된다.

이 작동에서, 중심점을 원점을 기준으로 계속해서 회전하게 하였기 때문에, 중심점 또한 사용자 선언 함수 create\_Matrix\_R()을 이용해서 구한 rotation 행렬과 곱해주어 새로운 중심점의 위치를 구한다. 그 후, 갱신된 중심점의 x, y좌표를 이용하여 translation 행렬을 만들어 polygon의 모든 점들을 갱신된 중심점의 x,y좌표만큼 이동시켜 주었다.

Shear function의 경우, 마우스 좌클릭과 alt키가 동시에 입력으로 주어졌는지 확인을 해야한다. 이를 위해 mousepress() 함수에서 위의 기능들에서 말했던 것과 같이 좌클릭이 입력으로 주어졌을 때, leftbuttonpressed의 값을 1, 즉 TRUE로 해주었고, 더 이상 클릭을 하고 있지 않다면 leftbuttonpressed의 값을 1, 즉 FALSE로 해주었다. 해당 값에 따라 leftbuttonpressed의 값이 TRUE일 경우에만, mousemove() 함수 내부에서 다음 단계의 작동을 실행하게 된다.

우선 glutGetModifiers() 함수를 통해 alt키가 눌려진 상태인지 확인한다. 만약 해당 값이 TRUE의 값을 가지고 있고, 동시에 leftbuttonpressed의 값이 TRUE일 경우에만, 분기문에 따라 다음과 같은 작동을 하게 된다. Dx와 dy, 즉 마우스 커서가 움직인 x방향과, y방향만큼 움직인 거리를 구해야 한다. 이를 위해 현재의 값인 x와 y에서 이전 위치의 좌표를 저장하고 있는 prevx, prevy의 값을 빼주었다.

그 후, dx와 dy의 값을 사용자 선언 함수 shearmatrixx()와 shearmatrixy()로 넘겨주어 x,y방향으로의 shear matrix를 계산한다. 이후, polygon의 모든 점들에 대해 shearx와 sheary 행렬을 곱해주어 해당 방향으로 점들이 당겨지게 하였다. 중심점에도 똑같은 계산을 해주어서 마우스의 움직임에 맞게 shear 아핀 변환을 통해 중심점이 이동될 수 있도록 하였다. 이후, 갱신된 값들을 glutPostRedisplay()를 통해 화면 상에 재출력하면서 함수가 마무리된다.