**Programming Assignment**

**2021KU0089** 이재형

**구현 목표**

1. OpenGL library의 Basic objects와 Basic, Advanced Transformations를 이용한 3-D텍스트 구현
2. Model Transformation을 사용한 3D텍스트의 심미적이고 다양한 움직임 구현
3. 코드의 수정 용이성과 직관성 그리고 간결함을 위한 코드의 기능중심 함수화
4. Complexity의 최소화를 지향하는 코딩방향성

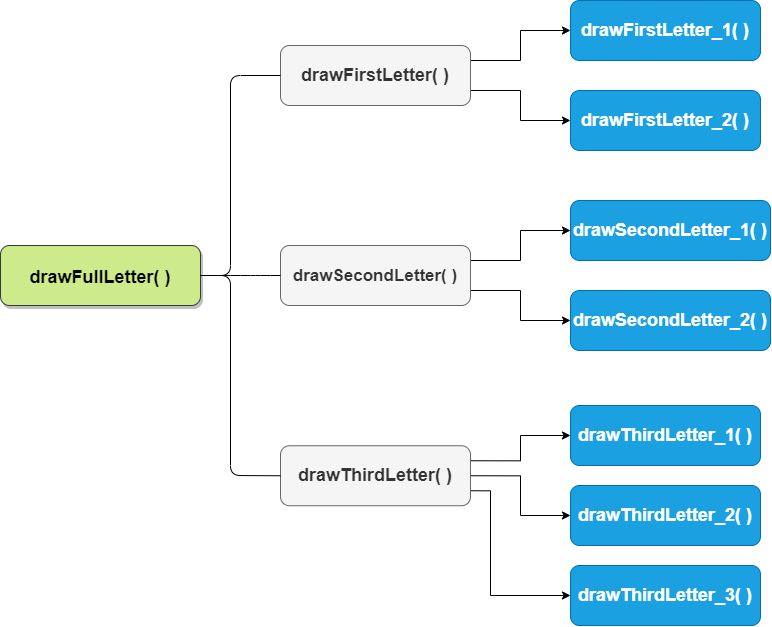
**구현 과정**

1. **주안점**

이번 과제는 변환행렬이 많이 들어가는 작업이었기 때문에 매번 버퍼가 초기화되는 glLoadIdentity()를 이용해 각 변환시 발생하는 간섭을 없애기 보다는 glPushMatrix() 와glPopMatrix() 를 사용하여 각 변환간 원치 않는 간섭은 없애고, 변환중에 모든 objects 들에 적용되기 원하는 것과 국부적인 object(“이재형” 에서 이응)에만 적용되기 원하는 변환을 최대한 구분하여 코딩이 용이하게 하였다.   
 이외 구현목표3 에서 언급하였듯, 단순한 이름의 projection결과만을 목표로 하지 않고, 지속해서 수정이 필요한 변수나 기준이 되는 상수, 반복되어 사용되는 Translation Matrix또한 최대한 함수나 상수 등으로 간결히 분류하여 코드에 대한 직관성을 높였다.

1. **함수 구조**

* **drawFullLetter(**GLfloat x, GLfloat y, GLfloat z, GLfloat scale**)**



위 구조도는 이름 석자를 그리는 함수로, 다음과 같은 포함관계를 가지고 있다. 그리고 2, 3번째 layer(하얀색, 하늘색)는 각 시작과 끝, 그리고 내부 Transformation 함수가 모두 glPushMatrix()와 glPopMatrix()로 감싸져 각 음절을 구현하면서 사용되는 Transformation이 다음 음절에 영향을 미치지 못함과 동시에 외부에서 적용한 Transformation이 한번에 모든 음절들에 적용될 수 있도록 하였다. 처음에는 스택을 초기화하는 glLoadIdentity()를 사용하였지만, 이는 모든 변환 Matrix가 초기화되어 전체적인 Rotation, Trnaslation, Scaling이 매우 불편하고 복잡해 Push와 Pop을 사용하게 되었다.

1. **이외 사용자 정의 함수**

void drawSimpleCube() 크기 1의 정해진 색상을 가지는 Solid + Wire cube를 그린다.  
void SecondTransformation() 키보드 입력 2가 입력되었을 때 사용될 변환을 모았다.  
void display() 버퍼를Claear하고, model을 transform시킨다.  
void Reshape() Window크기 조정이있을 때 무엇을 할지 정의한다.  
void keyboard() Keyboard 입력이 들어왔을 때 동작을 다룬다.

1. **구현 기능**

기본적으로 글자가 화면에 뜨고, 키보드의 특정 키를 누르고 있을 때 반응하도록 Keyboard()함수 내부에 switch 문을 이용하여 ‘x’ ‘y’ ‘z’ ‘r’ ‘q’ ‘1’ ‘2’ ‘3’ 총 8가지 입력에 대해 반응하도록 작성했다. 이 때 전역변수로 (BOOLEAN)token과 (GLfloat)angle을 각 입력마다 각각 지정하여 해당 button이 눌리면 token 을 enable 시켜 display에서 미리 구현한 Transformation이 적용되게 하였고, angle을 계속 증가시켜 회전 animation을 만들 수 있었다.

‘q’ 입력은 기존 skeleton code와 같이,exit(0)을 이용해 종료하는 기능을 담당한다.

‘r’은 변환에 있어서 초기상태로 reset을 해주는 기능을 담당한다.

‘x’,’y’,’z’ 는 각각 x y z축에 대한 회전변환을 매 호출 시 각도를 5도 +하여 회전하는 애니메이션을 만들었다.

‘1’ 은 Arbitrary axis에 대한 각 letter들의 회전변환을 구현하였다. “이재형” 중 ‘이’ 는 <3,2,5>방향벡터를 가지는 직선에 대하여, ‘재’는 <8,4,1> 직선에 대하여, ‘형’은 <1,2,4> 에 대하여Rotatef()를 이용, 각각 회전변환을 하였다.

‘2’ 와 ‘3’은 글자들이 +z방향으로 이동하였다가 최대 위치에 도달하면 다시 -z방향 원위치로 돌아가는 모션을 구상하였다. 이 때 속도 증가량이 일률적이면 너무 딱딱한 애니메이션이 나와, 가속도를 부여하는 느낌으로 아래에서 자세히 설명하겠지만 if문을 사용해 속도 증가량에 차이를 주어 부드러운 모션을 만들었다.  
 이 때 ‘2’ 는 글자가 앞으로 돌면서 점점 가까워졌다가 다시 원위치로 돌아가는 모션을 생각하였다. 이 때 앞 뒤 움직임은Translation 을, 제자리 회전은 Rotation을 사용하여 앞서 말했듯 가속도를 주고 각 글자마다 시작 위치와 각도를 다르게 부여해 모션을 더 Dynamic하게 만들었다.  
 마지막으로 ‘3’ 은 스타워즈의 명장면인 우주에서 별들이 앞으로 다가오는 장면의 느낌을 생각하였다. 많은 글자들을 밋밋하지 않게 다양한 크기로 Scale하고 map의 random location으로 Translation한 뒤 다시 key입력으로 Translation하여 앞뒤로 통통 거리는 듯한 모션을 만들었다.

이 때 x,y,z,1,2,3 변환 모두 서로 독립적인 Transformation이어서, 여러 개를 한번에 작동시켰을 때 재미있는 연출을 할 수 도 있었다.

**핵심 소스코드**

|  |
| --- |
| void drawFirstLetter\_1() {  //Draw 'ㅇ'  glPushMatrix();  glTranslatef(- 9, 0, 0);  glScalef(1, 1.4, 1);  glColor3f(0.3, 1.0, 0.5);  glutWireTorus(2.7, 6, 20, 20);  glColor3f(0.1, 0.8, 0.3);  glutSolidTorus(2.7, 6, 20, 20);  glPopMatrix();  }  void drawFirstLetter\_2() {  //Draw 'ㅣ'  glPushMatrix();  glTranslatef(+ 5, 0, 0);  glScalef(4.5, 30, 5);  drawSimpleCube();  glPopMatrix();  }  void drawFirstLetter() {  glPushMatrix();  glTranslatef(FIRST\_LETTER\_POSITION, 0, 0);  if (token\_1) { glRotatef(angle\_1, 0.3, 0.2, 0.5); }//1  if (token\_2) { SecondTrasnform(50,2); } //2 : back and forth rotation  drawFirstLetter\_1();  drawFirstLetter\_2();  glPopMatrix();  } |

글자를 모델링하는 부분의 소스코드이다. 해당 부분은 ‘이’를 모델링 하였다. glut라이브러리의 Cube와 Torus를 사용하였고, 이들을 Scale하고 적절히 Translation 하였다. 그리고 Push() Pop() 을 이용해 display()에서 들어오는 Trnasformation을 object들이 모두 받을 수 있게 하였고, 각자 크기, 위치, 회전을 조절하는 변환들끼리 간섭이 없도록 하였다.   
 drawFirstLetter()에 있는 Translation은 ‘이’를 제일 왼쪽으로 옮긴 과정이고, FIRST\_LETTER\_POSITION는 위치가 상수로 지정된 부분이다. If문 두개는 각각 ‘1’,’2’ 입력이 들어왔을 때 글자를 조절하는 부분이고, 글자를 각각 다르게 조절하기위해 매 글자에 다르게 Trnasformation을 주려 한 부분이다. SecondTransform()은 glTrnaslatef(), glRotatef()로 구성되어있다.

나머지 글자들도 같은 과정으로 적절히 만들었다.

|  |
| --- |
| void drawSimpleCube() {  glColor3f(0.3, 1.0, 0.5);  glutWireCube(1);  glColor3f(0.1, 0.8, 0.3);  glutSolidCube(1);  } |

크기1의 외각선을 가지는 Cube를 그리는 함수로 사소하지만 SolidCube 위에 비슷한 색의 WireCube를 겹쳐 눈에 들어오도록 신경을 써 볼 수 있었다. 뒤에서 글씨를 디자인할 때 계속해서 사용되어 따로 함수로 정의하였다.

|  |
| --- |
| void drawFullLetter(GLfloat x, GLfloat y, GLfloat z,GLfloat scale) {  glPushMatrix();  glTranslatef(x, y, z);  glScalef(scale, scale, scale);  drawFirstLetter();  drawSecondLetter();  drawThirdLetter();  glPopMatrix();  } |

(x,y,z)위치에 scale크기의 이름 “이재형” 을 호출하는 함수이다. 이 또한 간섭은 피하고 전역적인 변환이 가능하도록 Push Pop을 이용하였다.

|  |
| --- |
| void SecondTransformation(int n, int m) {  glTranslatef(0, 0, (angle\_2 - 100) / m);//m : make differences in velocity  glRotatef(angle\_2 + n, 0, 1, 0); //n : gives different rotating visual  } |

다음은 맨처음 코드설명에서 언급하였던 ‘2’ 호출 시 에서 호출하는 함수로, n만큼 Rotation angle을 더하고 m만큼 회전 속도를 늦추는 함수이다.

|  |
| --- |
| BOOLEAN reset = false;  BOOLEAN token\_x = false;  BOOLEAN token\_y = false;  BOOLEAN token\_z = false;  BOOLEAN token\_1 = false;  BOOLEAN token\_2 = false;  BOOLEAN token\_2\_spd = false;  BOOLEAN token\_3 = false;  BOOLEAN token\_3\_spd = false;  float angle\_x = 0.0f;  float angle\_y = 0.0f;  float angle\_z = 0.0f;  float angle\_1 = 0.0f;  float angle\_2 = 0.0f;  float speed\_3 = 0.0f; |
| void Keyboard(unsigned char key, int x, int y)  {  switch (key)  {  case 'q'://press"q" for exit program  exit(0);  break;  case 'r'://press"r" for reset  reset = true;  angle\_x = 0;  angle\_y = 0;  angle\_z = 0;  angle\_1 = 0;  angle\_2 = 0;  speed\_3 = 0;  token\_x = false;  token\_y = false;  token\_z = false;  token\_1 = false;  token\_2 = false;  token\_3 = false;  token\_2\_spd = false;  token\_3\_spd = false;  glutPostRedisplay();  break;  case 'x': //press"x" for x-axis rotation  token\_x = true;  angle\_x += 5;  glutPostRedisplay();  break;  case 'y': //press"x" for x-axis rotation  token\_y = true;  angle\_y += 5;  glutPostRedisplay();  break;  case 'z': //press"x" for x-axis rotation  token\_z = true;  angle\_z += 5;  glutPostRedisplay();  break;  case '1': //press"1" for Arbitrary axis rotation  token\_1 = true;  angle\_1 += 5;  glutPostRedisplay();  break;  case '2':  token\_2 = true;  //soft angle speed control  if (!token\_2\_spd) {  if (angle\_2 < 20) { angle\_2 += 1; }  if (angle\_2 >= 20 && angle\_2 < 50) { angle\_2 += 3; }  if (angle\_2 >= 50 && angle\_2 < 180) { angle\_2 += 6; }  if (angle\_2 >= 180 && angle\_2 < 280) { angle\_2 += 5; }  if (angle\_2 >= 280 && angle\_2 < 330) { angle\_2 += 3; }  if (angle\_2 >= 330 && angle\_2 <= 360) { angle\_2 += 1.5; }  if (angle\_2 > 360) { token\_2\_spd = !token\_2\_spd; }  }  //reverse the speed incresment  if (token\_2\_spd) {  if (angle\_2 < 0) { token\_2\_spd = !token\_2\_spd; }  if (angle\_2 < 20) { angle\_2 -= 1; }  if (angle\_2 >= 20 && angle\_2 < 50) { angle\_2 -= 3; }  if (angle\_2 >= 50 && angle\_2 < 180) { angle\_2 -= 6; }  if (angle\_2 >= 180 && angle\_2 < 280) { angle\_2 -= 5; }  if (angle\_2 >= 280 && angle\_2 < 330) { angle\_2 -= 3; }  if (angle\_2 >= 330) { angle\_2 -= 1.5; }  }  glutPostRedisplay();  break;  case '3':  token\_3 = true;  //dynamic speed control  if (!token\_3\_spd) {  if (speed\_3 < 20) { speed\_3 += 2; }  if (speed\_3 >= 20 && speed\_3 < 50) { speed\_3 += 4; }  if (speed\_3 >= 50 && speed\_3 < 80) { speed\_3 += 6; }  if (speed\_3 >= 80 && speed\_3 < 120) { speed\_3 += 7; }  if (speed\_3 > 120) { token\_3\_spd = true; }  }  //inverse process  if (token\_3\_spd) {  if (speed\_3 < 0) { token\_3\_spd = false; }  if (speed\_3 >= 0 && speed\_3 < 20) { speed\_3 -= 2; }  if (speed\_3 >= 20 && speed\_3 < 50) { speed\_3 -= 4; }  if (speed\_3 >= 50 && speed\_3 < 80) { speed\_3 -= 6; }  if (speed\_3 >= 80) { speed\_3 -= 7; }  }  glutPostRedisplay();  break;  }  } |

전역변수 선언부와 keyboard()이다. 전역변수들은 키보드 입력에 의해 조정되는 값들을 선언하였다. Keyboard()는 키보드 입력에 따른 token 과 angle 값 수정이다. Skeleton code2 처럼 switch문을 사용하였고, ‘2’ ‘3’ 입력에 대해서는 가속도를 주었다고 언급했었는데, 다음과 같이 angle\_2 혹은 speed\_3 를 0~360사이로 증감하되, 초반과 후반은 조금씩 증가시키고 중간은 빠르게 증가시켜 구현하였다.

‘r’의 reset은 다음처럼 모든 token을 false로 초기화시키고 angle과 speed또한 0으로 초기화 시켜 처음 값으로 돌렸다.

매 case 마다 glutPostRedisplay()를 넣어 호출마다 화면을 Refresh하여 변경값을 반영시킬 수 있었다.

|  |
| --- |
| void display()  {  **glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); //scene buffer clear**  **glMatrixMode(GL\_MODELVIEW);**  **glLoadIdentity(); //Transformation matrix initialize**  **glTranslatef(0, 0, -100); //gives camera depth**    if (reset) { reset = false; } //reset toggle  if (token\_x) { glRotatef(angle\_x, 1, 0, 0); } //x : x-axis rotation  if (token\_y) { glRotatef(angle\_y, 0, 1, 0); } //y : y-axis rotation  if (token\_z) { glRotatef(angle\_z, 0, 0, 1); } //z : z-axis rotation  if (token\_1) { glRotatef(angle\_1, 1, 0, 0); } //1 : arbitrary rotation + x-axis rotation  if (token\_3) { glTranslatef(0, 0, -speed\_3); } //3 : moving letters    //default letter  **drawFirstLetter();**  **drawSecondLetter();**  **drawThirdLetter();**    if (token\_3) //3 : making number of letters  {  glTranslatef(0, 0, 70);  drawFullLetter(60, 20, -200, 0.6); drawFullLetter(-74, 0, -100, 0.3);  drawFullLetter(-57, 15, -160, 0.4); drawFullLetter(73, -5, -50, 0.5);  drawFullLetter(18, 10, -210, 0.9); drawFullLetter(-72, -15, 0, 0.8);  drawFullLetter(30, 25, -50, 0.8); drawFullLetter(-81, -20, -150, 1.1);  drawFullLetter(22, 20, -10, 1.6); drawFullLetter(-49, 20, -20, 0.8);  drawFullLetter(-27, 15, 0, 0.1); drawFullLetter(38, -10, -100, 1.2);  drawFullLetter(-52, 10, -20, 0.5); drawFullLetter(-67, 29, 0, 0.2);  drawFullLetter(-19, 10, -100, 0.6); drawFullLetter(14, -30, -190, 0.8);  }  **glutSwapBuffers();**  } |

Display()에선 굵은 글씨 부분인 기본적인 이름 modeling에다 keyboard입력에 의한 transformation 부분인 if문들을 구성하였다. glLoadIdentity()로 호출 시 항상 버퍼를 먼저 초기화시켰고, 아래 glTrnaslatef(0,0,-100)은 카메라를 z=100으로 이동시키는 대신 글자들을 모두 z = -100 으로 이동시켜 적절한 위치에 놓은 부분이다. 처음에는 glutLookAt()을 사용하려 하였지만 기존 코드들과 자꾸 엉키는 등 오류가 발생해 Translate로 간단히 해결할 수 있었다.

|  |
| --- |
| void Reshape(int w, int h)  {  glViewport(0, 0, w, h); // w: is width, h: is height  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(45, (float)w / h, 0.1, 500); // define view frustum  } |
| int main(int argc, char\* argv[]) {  //Window generation  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE);//double buffer -> faster  glutInitWindowPosition(100, 100); //Window position  glutInitWindowSize(1400, 800); //Window size  glutCreateWindow("NAME"); //Window name  //Initialize function  Initialize();  //callback function  glutDisplayFunc(display);  glutReshapeFunc(Reshape);  glutKeyboardFunc(Keyboard);  glutSwapBuffers();  glutMainLoop();  return 0;  } |

다음은 Reshape()와 main()으로 기존 skeleton code2의 것을 차용하였다.  
Reshape()를 통해 창 크기를 동적으로 조정할 수 있게 되었고, main()에선 기본적인 DisplayMode설정과 Window호출, Initialize()에서 배경을 설정하는 등 기본 설정 함수들을 활성화 하였다.

**구현 결과화면**

텍스트이(가) 표시된 사진

자동 생성된 설명

기본 실행 OR ‘r’ pressed (reset)

텍스트이(가) 표시된 사진

자동 생성된 설명

‘x’ pressed : X축 회전

텍스트이(가) 표시된 사진

자동 생성된 설명

‘y’ pressed : Y축 회전

텍스트이(가) 표시된 사진

자동 생성된 설명

‘z’ pressed : Z축 회전

화살이(가) 표시된 사진

자동 생성된 설명

‘1’ pressed : Arbitrary Axis Rotation

|  |
| --- |
|  |
|  |
|  |

‘2’ pressed : velocity controlled Translation & y-axis Rotation

|  |
| --- |
|  |
|  |

3 pressed : Multiple-letter velocity controled Translation

|  |
| --- |
|  |
|  |

1, 2, 3 Mixed