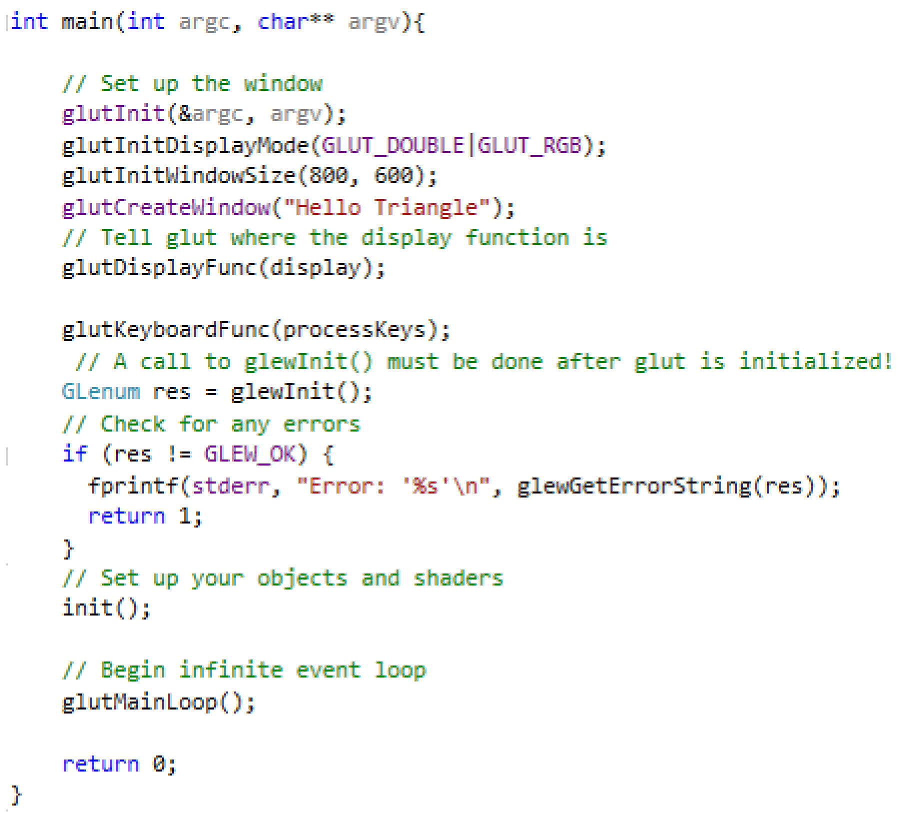
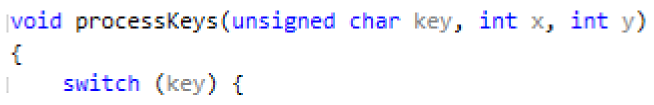
Keyboard Controls

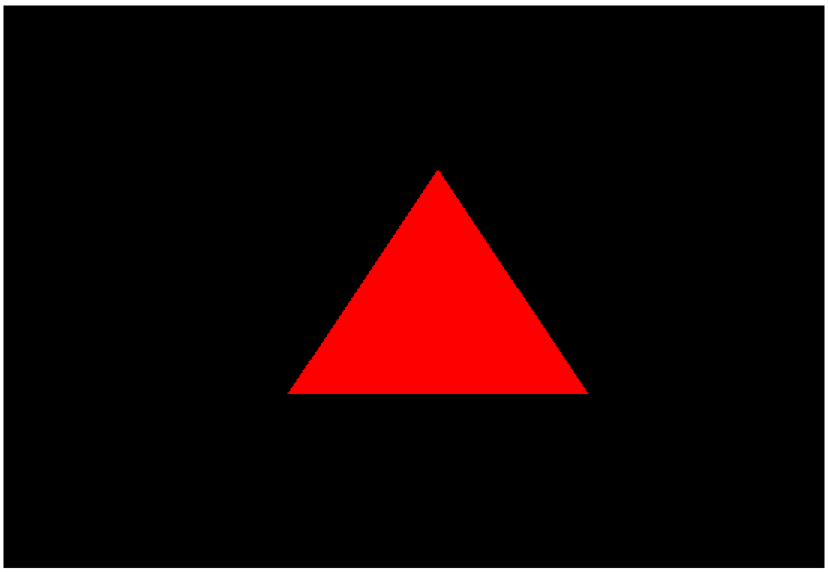
Keyboard controls were accomplished by adding a glutKeyboardFunc() callback which would be invoked upon keypress. The callback function takes in another function called processKeys where the transformations take place and the keys transformations are used with different switch cases.



```
int main(int argc, char** argv){

    // Set up the window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Hello Triangle");
    // Tell glut where the display function is
    glutDisplayFunc(display);

    glutKeyboardFunc(processKeys);
     // A call to glewInit() must be done after glut is initialized!
    GLenum res = glewInit();
    // Check for any errors
    if (res != GLEW_OK) {
      fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res));
      return 1;
    }
    // Set up your objects and shaders
    init();

    // Begin infinite event loop
    glutMainLoop();

    return 0;
}
```



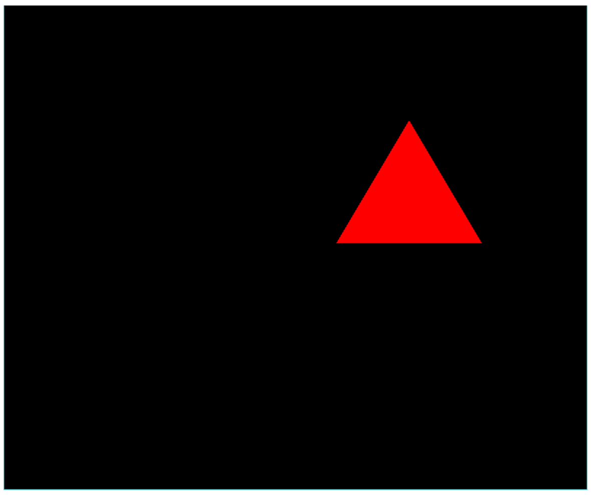
```
void processKeys(unsigned char key, int x, int y)
{
    switch (key) {
```

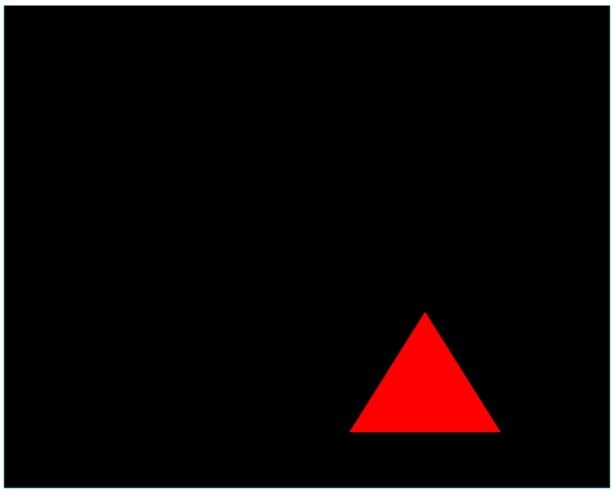


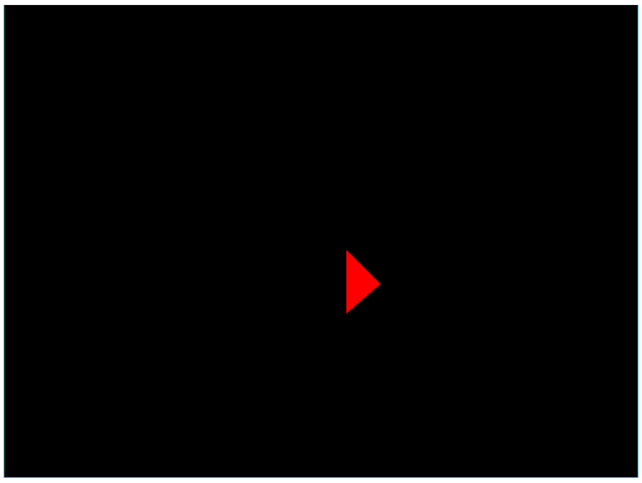
Translations

On keypress the red triangle has a translation operation performed on its transformation matrix using the glm math library glm::translate() function, the triangle is translated by 0.01 on each keypress.

```
case 'q':
    tran_x += 0.01f;
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    translatex = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    scaling = glm::scale(transform, glm::vec3(scale));
    transform = translatex*rotation_x* rotation_y* rotation_z*scaling;
    break;
case 'w':
    tran_y += 0.01f;
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    translatey = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    scaling = glm::scale(transform, glm::vec3(scale));
    transform = translatey*rotation_x* rotation_y* rotation_z*scaling;
    break;
case 'e':
    tran_z += 0.1f;
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    translatez = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    scaling = glm::scale(transform, glm::vec3(scale));
    transform = translatez*rotation_x* rotation_y* rotation_z*scaling;
    break;
```


x-direction translation

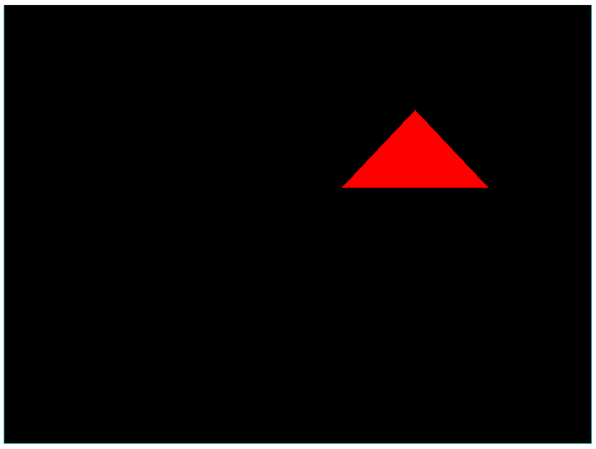

y-direction translation

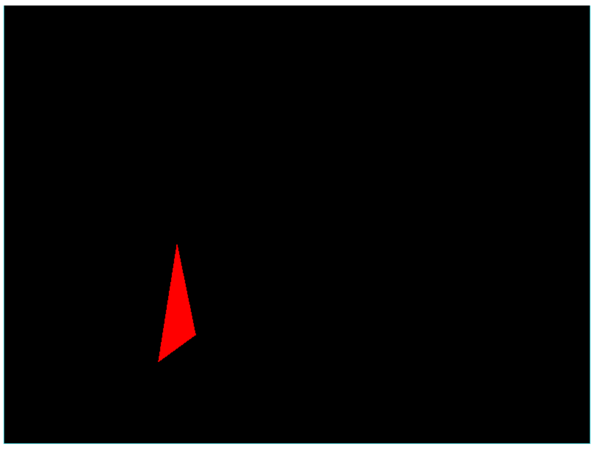

z-direction translation (with rotation)

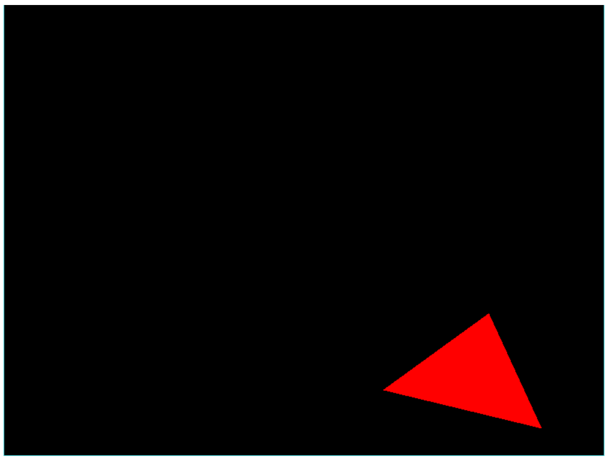
Rotations

On keypress the red triangle has a rotation operation performed on its transformation matrix using the glm math library glm::rotate() function, the angle is increased by 15 degrees upon each keypress.

```
case 'r':
    angle_x += 15;
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    translatex = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    scaling = glm::scale(transform, glm::vec3(scale));
    transform = translatex*rotation_x* rotation_y* rotation_z*scaling;
    break;
case 't':
    angle_y += 15;
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    translatey = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    scaling = glm::scale(transform, glm::vec3(scale));
    transform = translatey*rotation_x* rotation_y* rotation_z*scaling;
    break;
case 'y':
    angle_z += 15;
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    translatez = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    scaling = glm::scale(transform, glm::vec3(scale));
    transform = translatez*rotation_x* rotation_y* rotation_z*scaling;
    break;
```

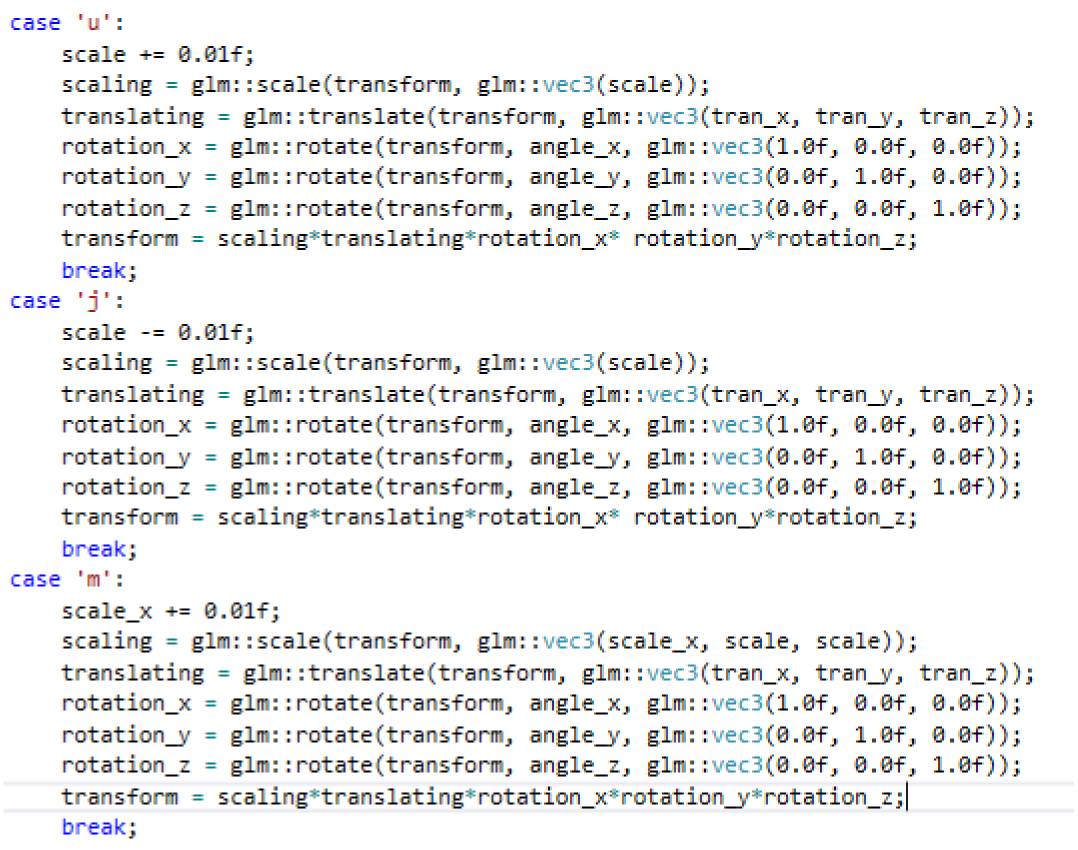

x-axis rotation


y-axis rotation

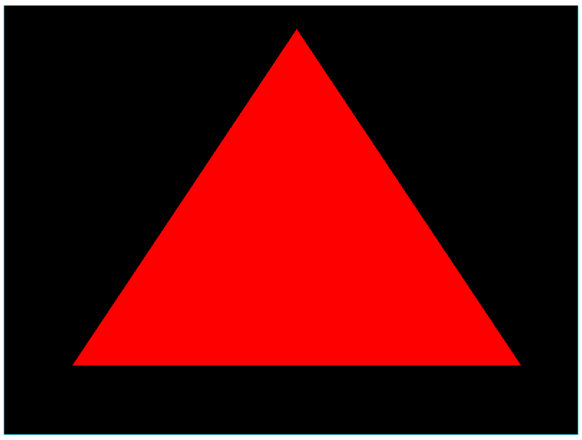

z-axis rotation

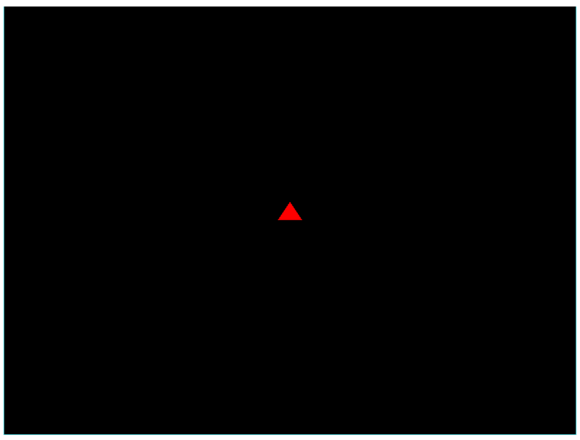
Scaling

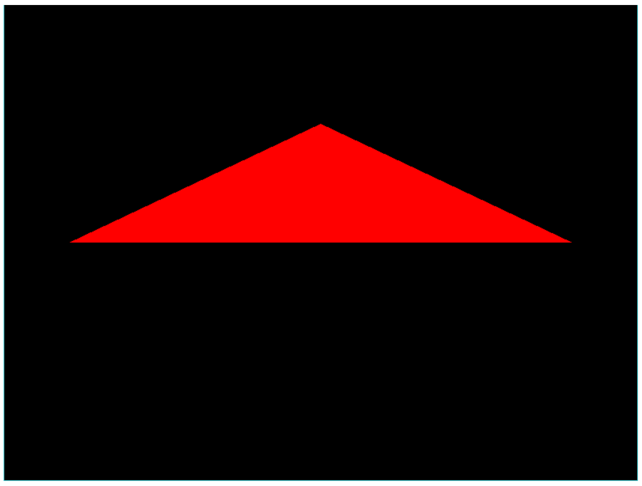
On keypress the red triangle has a scaling operation performed on its transformation matrix using the glm math library glm::scale() function, the triangle is scaled by 0.01 on each keypress.



```
case 'u':
    scale += 0.01f;
    scaling = glm::scale(transform, glm::vec3(scale));
    translating = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    transform = scaling*translating*rotation_x* rotation_y*rotation_z;
    break;
case 'j':
    scale -= 0.01f;
    scaling = glm::scale(transform, glm::vec3(scale));
    translating = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    transform = scaling*translating*rotation_x* rotation_y*rotation_z;
    break;
case 'm':
    scale_x += 0.01f;
    scaling = glm::scale(transform, glm::vec3(scale_x, scale, scale));
    translating = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    transform = scaling*translating*rotation_x*rotation_y*rotation_z;
    break;
```


uniform scaling (increase)


uniform scaling (decrease)

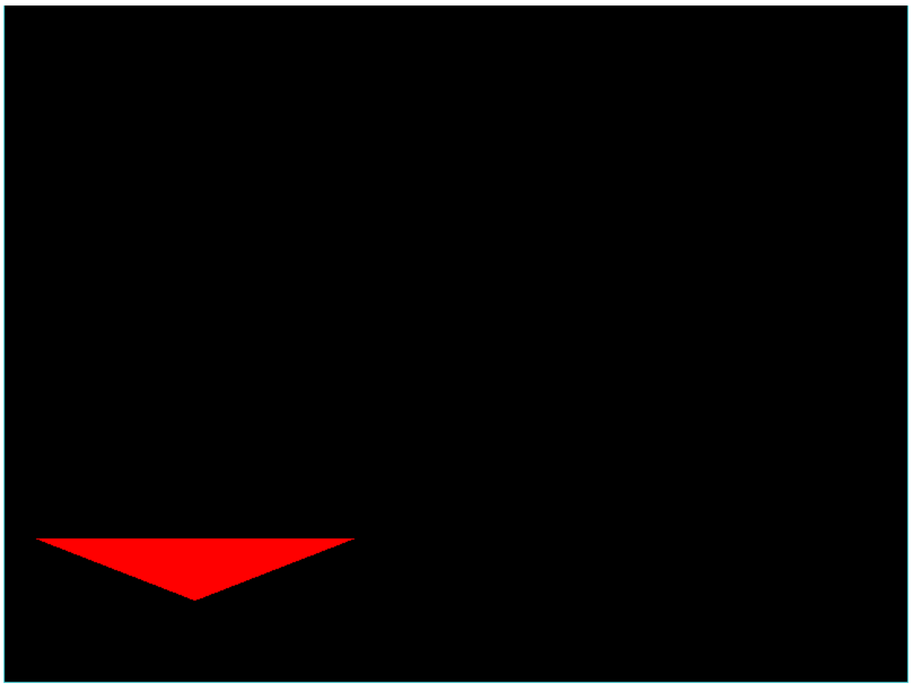


non-uniform scaling (x-direction)

Combination

On keypress the red triangle has scaling, translating (in the x and y direction) and rotation operations performed on its transformation matrix using the glm math library transformation functions.

```
case 'i':
    scale += 0.01f;
    angle_x -= 15;
    tran_x -= 0.01f;
    tran_y -= 0.01f;
    scaling = glm::scale(transform, glm::vec3(scale));
    translating = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    rotation_x = glm::rotate(transform, angle_x, glm::vec3(1.0f, 0.0f, 0.0f));
    rotation_y = glm::rotate(transform, angle_y, glm::vec3(0.0f, 1.0f, 0.0f));
    rotation_z = glm::rotate(transform, angle_z, glm::vec3(0.0f, 0.0f, 1.0f));
    transform = scaling*translating*rotation_x*rotation_y*rotation_z;
    break;
```



Multiple Triangles

On keypress the Boolean flag is set to true and a translation operation is performed on two triangles in the opposite x-directions, using two different transformation matrix. Otherwise transformations are only performed on a single red triangle.

```
if (flag == true)
{
    tran_x += 0.01f;
    transform = glm::translate(transform, glm::vec3(tran_x, tran_y, tran_z));
    glUniformMatrix4fv(transformationID, 1, GL_FALSE, glm::value_ptr(transform));
    glDrawArrays(GL_TRIANGLES, 0, 3);

    glm::mat4 transform2;

    tran_gx -= 0.01f;
    transform2 = glm::translate(transform2, glm::vec3(tran_gx, tran_y, tran_z));
    glUniformMatrix4fv(transformationID, 1, GL_FALSE, glm::value_ptr(transform2));
    glDrawArrays(GL_TRIANGLES, 3, 3);

    flag = false;
}
else if (flag == false)
    glDrawArrays(GL_TRIANGLES, 0, 3);
```