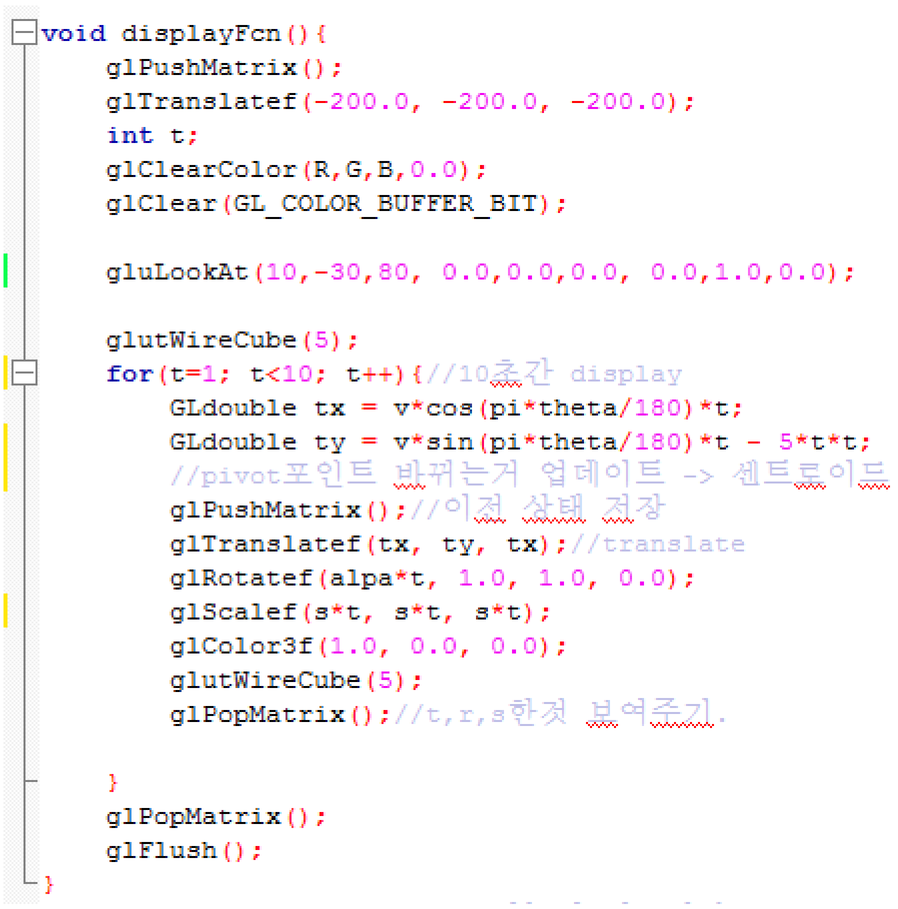
# Computer Graphics HW03

소프트웨어학부
소프트웨어학과
2017012197
여채린

문영식 교수님

1. **Source code for the main part of the program**

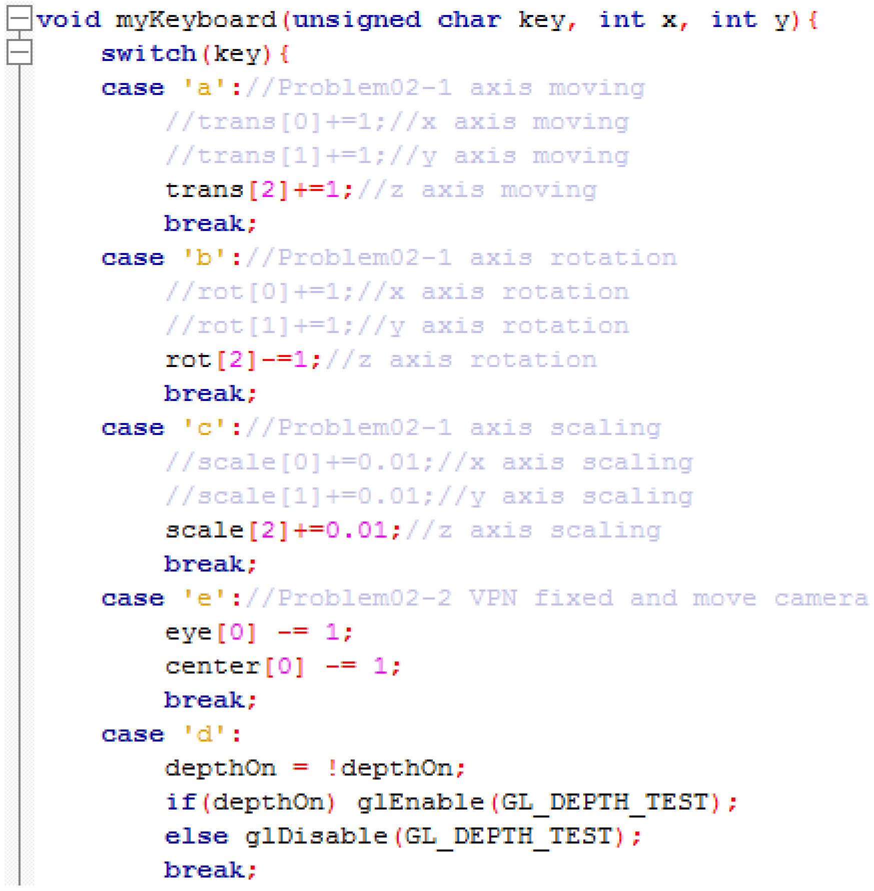
- Problem01



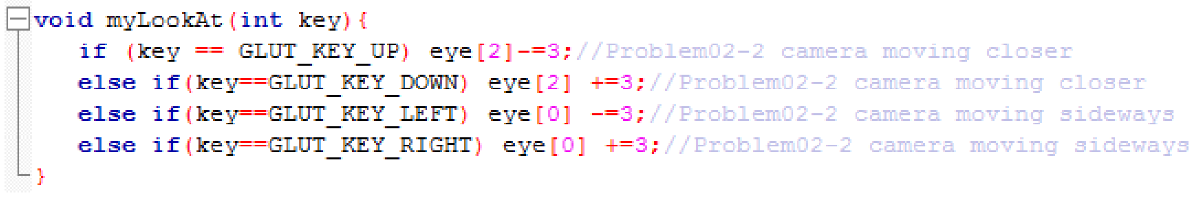
```
void displayFcn(){
    glPushMatrix();
    glTranslatef(-200.0, -200.0, -200.0);
    int t;
    glClearColor(R,G,B,0.0);
    glClear(GL_COLOR_BUFFER_BIT);

    gluLookAt(10,-30,80, 0.0,0.0,0.0, 0.0,1.0,0.0);

    glutWireCube(5);
    for(t=1; t<10; t++){//10초간 display
        GLdouble tx = v*cos(pi*theta/180)*t;
        GLdouble ty = v*sin(pi*theta/180)*t - 5*t*t;
        //pivot포인트 바뀌는거 업데이트 -> 센트로이드
        glPushMatrix();//이전 상태 저장
        glTranslatef(tx, ty, tx);//translate
        glRotatef(alpa*t, 1.0, 1.0, 0.0);
        glScalef(s*t, s*t, s*t);
        glColor3f(1.0, 0.0, 0.0);
        glutWireCube(5);
        glPopMatrix();//t,r,s한것 보여주기.

    }
    glPopMatrix();
    glFlush();
}
```

- Problem02

```
void myDisplay()
{
    glClearColor(R,G,B,0.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    if(!projection)//eye는 관찰자의 눈, center은 카메라가 바라보는 방향.
        gluLookAt(eye[0], eye[1], eye[2], center[0], center[1], center[2], 0, 1, 0);

    glLineWidth(5);
    //x y z축
    glBegin(GL_LINES);
    glColor3f(1.0,0.0,0.0);
    glVertex3f(0.0,0.0,0.0);
    glVertex3f(1000.0,0.0,0.0);
    glColor3f(0.0,1.0,0.0);
    glVertex3f(0.0,0.0,0.0);
    glVertex3f(0.0,1000.0,0.0);
    glColor3f(0.0,0.0,1.0);
    glVertex3f(0.0,0.0,0.0);
    glVertex3f(0.0,0.0,1000.0);
    glEnd();

    glRotatef(rot[0], 1.0f, 0.0f, 0.0f);
    glRotatef(rot[1], 0.0f, 1.0f, 0.0f);
    glRotatef(rot[2], 0.0f, 0.0f, 1.0f);
    glTranslatef(trans[0], 0.0f, 0.0f);
    glTranslatef(0.0f, trans[1], 0.0f);
    glTranslatef(1.0f, 0.0f, trans[2]);

    glScalef(scale[0], scale[1], scale[2]);

    glPushMatrix();//이전 상태 저장
    glTranslatef(0,0,0.1);
    glScalef(1,1,1);
    glColor3f(1.0, 0.0, 0.0);
    if(fillOn) glutSolidCube(60);
    glColor3f(0,0,0);
    glutWireCube(60);
    glPopMatrix();//t,r,s한것 보여주기.

    glFlush();
    glutSwapBuffers();
}
```

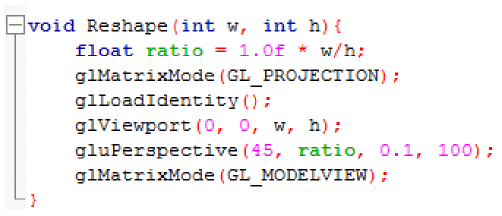


```
void myKeyboard(unsigned char key, int x, int y){
    switch(key){
    case 'a'://Problem02-1 axis moving
        //trans[0]+=1;//x axis moving
        //trans[1]+=1;//y axis moving
        trans[2]+=1;//z axis moving
        break;
    case 'b'://Problem02-1 axis rotation
        //rot[0]+=1;//x axis rotation
        //rot[1]+=1;//y axis rotation
        rot[2]-=1;//z axis rotation
        break;
    case 'c'://Problem02-1 axis scaling
        //scale[0]+=0.01;//x axis scaling
        //scale[1]+=0.01;//y axis scaling
        scale[2]+=0.01;//z axis scaling
        break;
    case 'e'://Problem02-2 VPN fixed and move camera
        eye[0] -= 1;
        center[0] -= 1;
        break;
    case 'd':
        depthOn = !depthOn;
        if(depthOn) glEnable(GL_DEPTH_TEST);
        else glDisable(GL_DEPTH_TEST);
        break;
```

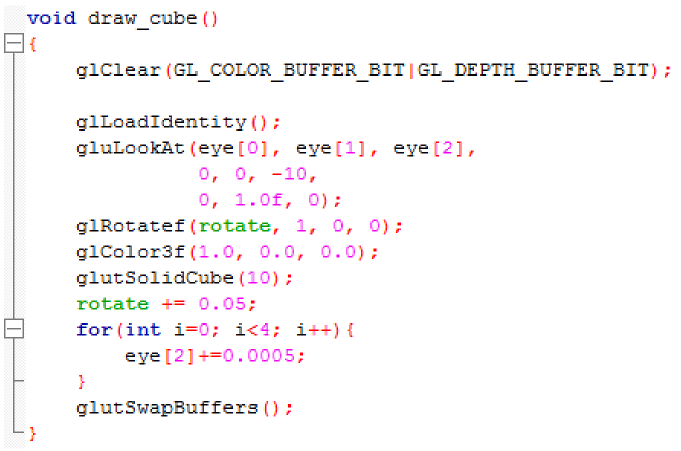


```
void myLookAt(int key){
    if (key == GLUT_KEY_UP) eye[2]-=3;//Problem02-2 camera moving closer
    else if(key==GLUT_KEY_DOWN) eye[2] +=3;//Problem02-2 camera moving closer
    else if(key==GLUT_KEY_LEFT) eye[0] -=3;//Problem02-2 camera moving sideways
    else if(key==GLUT_KEY_RIGHT) eye[0] +=3;//Problem02-2 camera moving sideways
}
```

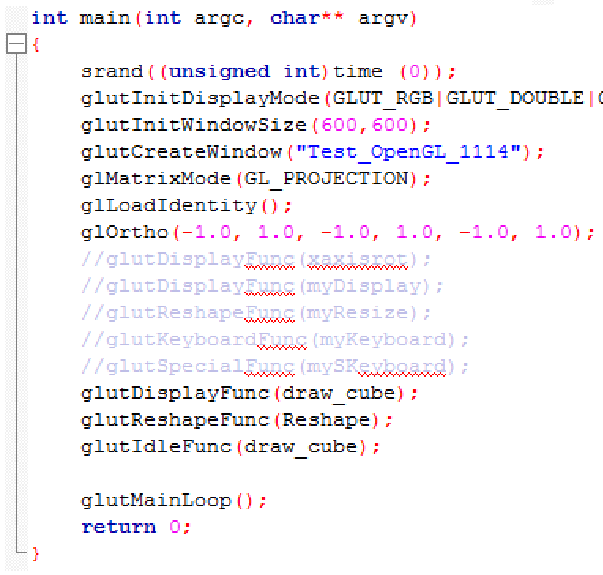
- When use glutIdleFunc() for Problem02-(3)



```
void Reshape(int w, int h){
    float ratio = 1.0f * w/h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, w, h);
    gluPerspective(45, ratio, 0.1, 100);
    glMatrixMode(GL_MODELVIEW);
}
```

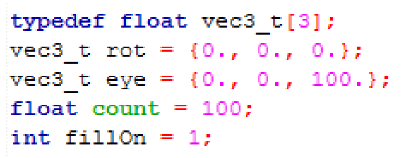


```
void draw_cube()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();
    gluLookAt(eye[0], eye[1], eye[2],
              0, 0, -10,
              0, 1.0f, 0);
    glRotatef(rotate, 1, 0, 0);
    glColor3f(1.0, 0.0, 0.0);
    glutSolidCube(10);
    rotate += 0.05;
    for(int i=0; i<4; i++){
        eye[2]+=0.0005;
    }
    glutSwapBuffers();
}
```

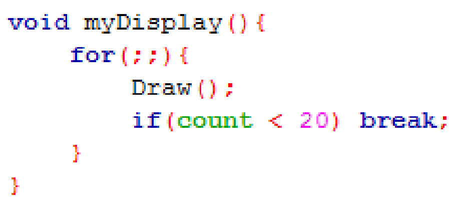


```
int main(int argc, char** argv)
{
    srand((unsigned int)time (0));
    glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_DEPTH);
    glutInitWindowSize(600,600);
    glutCreateWindow("Test_OpenGL_1114");
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    //glutDisplayFunc(xaxisrot);
    //glutDisplayFunc(myDisplay);
    //glutReshapeFunc(myResize);
    //glutKeyboardFunc(myKeyboard);
    //glutSpecialFunc(mySKeyboard);
    glutDisplayFunc(draw_cube);
    glutReshapeFunc(Reshape);
    glutIdleFunc(draw_cube);

    glutMainLoop();
    return 0;
}
```

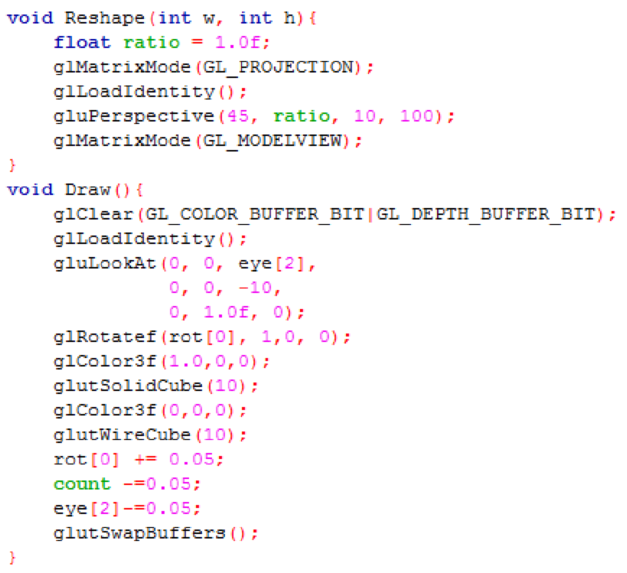
- When do not use glutIdleFunc() for Problem02-(3)



```
typedef float vec3_t[3];
vec3_t rot = {0., 0., 0.};
vec3_t eye = {0., 0., 100.};
float count = 100;
int fillOn = 1;
```



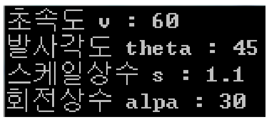
```
void myDisplay(){
    for(;;){
        Draw();
        if(count < 20) break;
    }
}
```



```
void Reshape(int w, int h){
    float ratio = 1.0f;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45, ratio, 10, 100);
    glMatrixMode(GL_MODELVIEW);
}
void Draw(){
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(0, 0, eye[2],
              0, 0, -10,
              0, 1.0f, 0);
    glRotatef(rot[0], 1,0, 0);
    glColor3f(1.0,0,0);
    glutSolidCube(10);
    glColor3f(0,0,0);
    glutWireCube(10);
    rot[0] += 0.05;
    count -=0.05;
    eye[2]-=0.05;
    glutSwapBuffers();
}
```

**2.Screen capture to see the execution result**

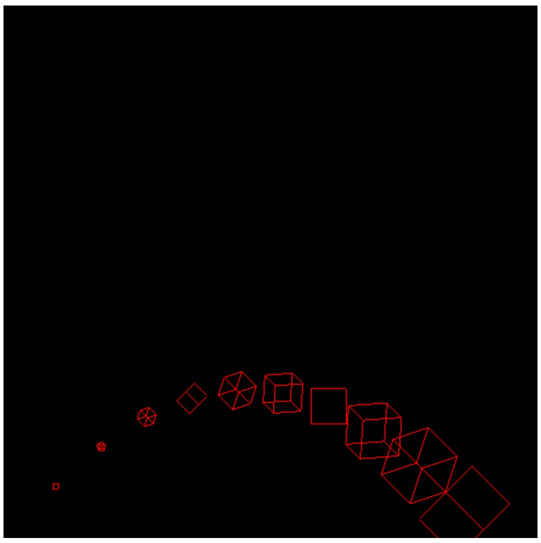
- Problem01

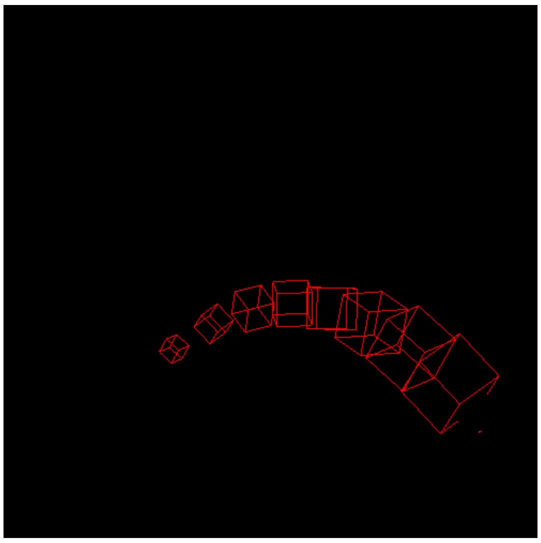


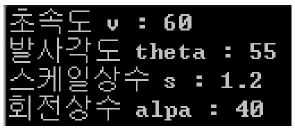
```
초속도 v : 60
발사각도 theta : 45
스케일상수 s : 1.1
회전상수 alpa : 30
```

- glOrtho(-250.0, 250.0, -250.0, 250.0, -200.0, 200.0);

- Without gluLookAt - gluLookAt(10,-30,80, 0.0,0.0,0.0, 0.0,1.0,0.0);



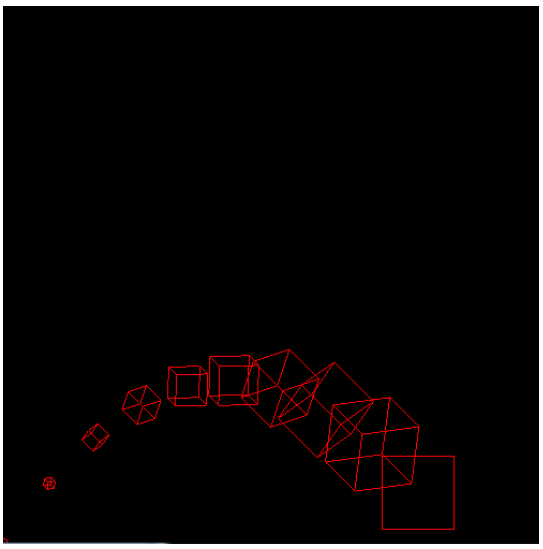


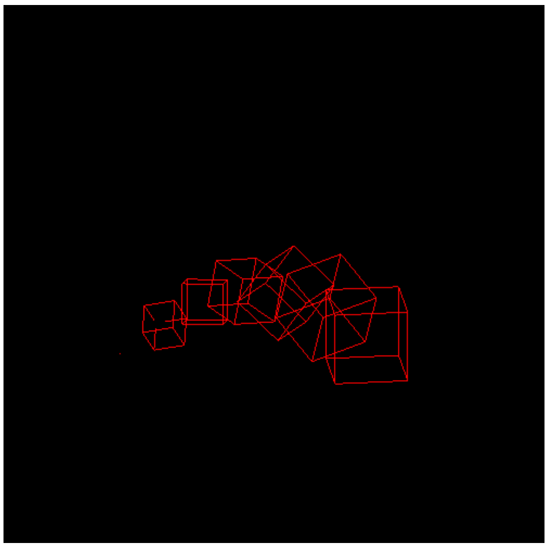


```
초속도 v : 60
발사각도 theta : 55
스케일상수 s : 1.2
회전상수 alpa : 40
```

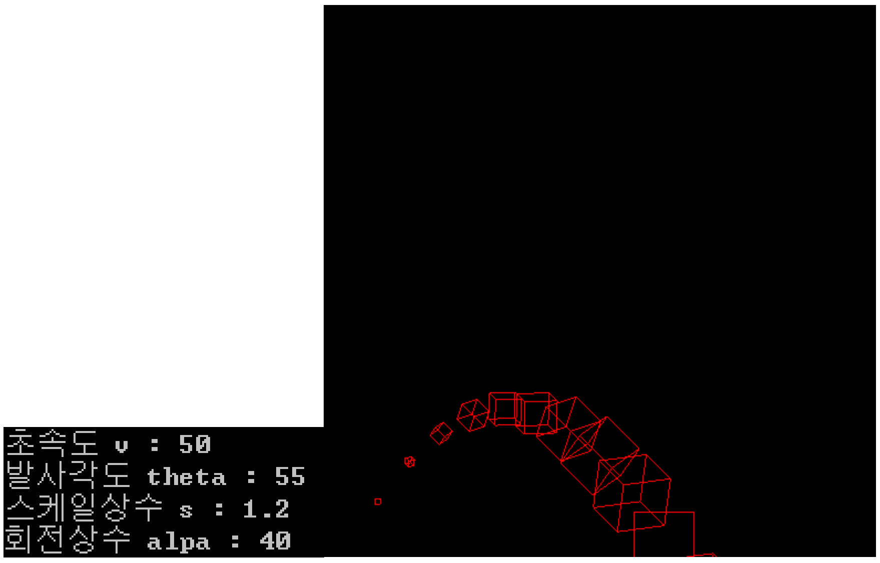
- glOrtho(-200.0, 200.0, -200.0, 200.0, -200.0, 200.0);

- Without gluLookAt - gluLookAt(10,-30,80, 0.0,0.0,0.0, 0.0,1.0,0.0);



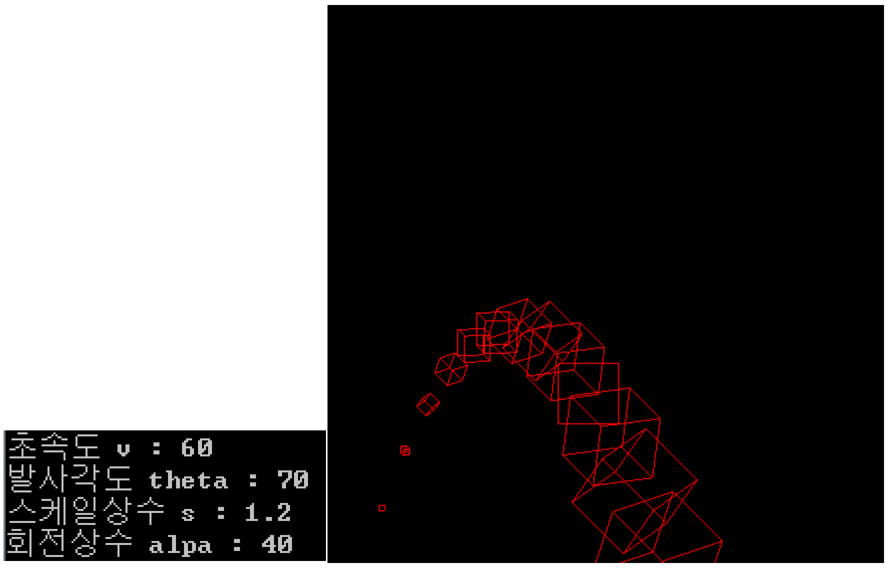


- glOrtho(-200.0, 200.0, -200.0, 200.0, -200.0, 200.0);, Without gluLookAt



```
초속도 v : 50
발사각도 theta : 55
스케일상수 s : 1.2
회전상수 alpa : 40
```

- glOrtho(-200.0, 200.0, -200.0, 200.0, -200.0, 200.0);, Without gluLookAt



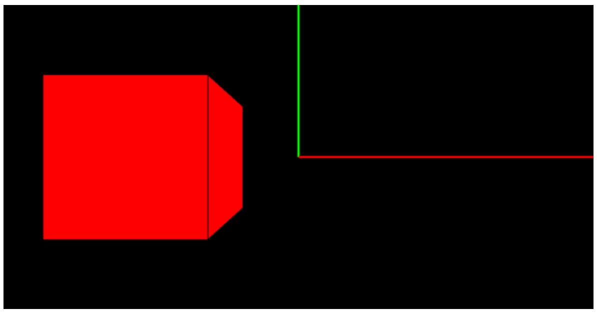
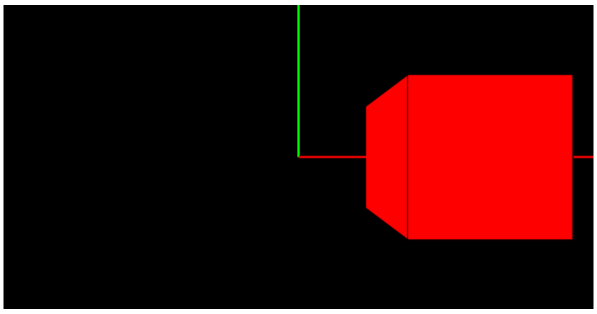
```
초속도 v : 60
발사각도 theta : 70
스케일상수 s : 1.2
회전상수 alpa : 40
```

- Problem02

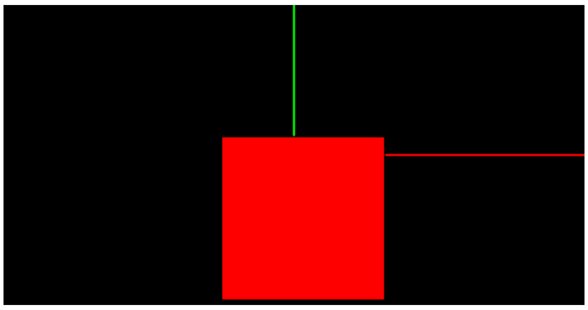
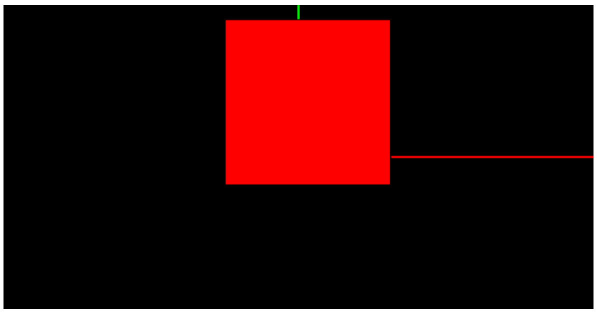
(1) With fixed camera position, and fixed ‘Loot-At Point’,

- When object moves in x direction(or y direction, or z direction)

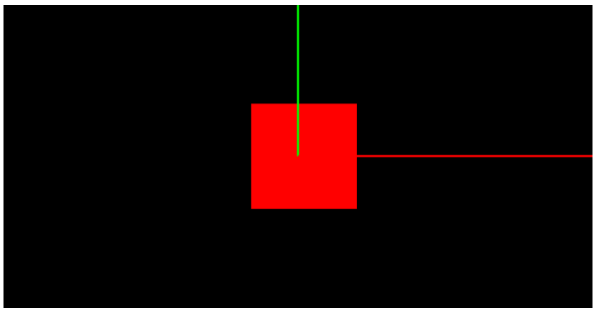
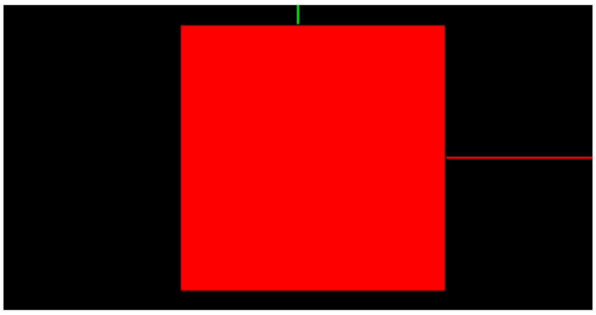
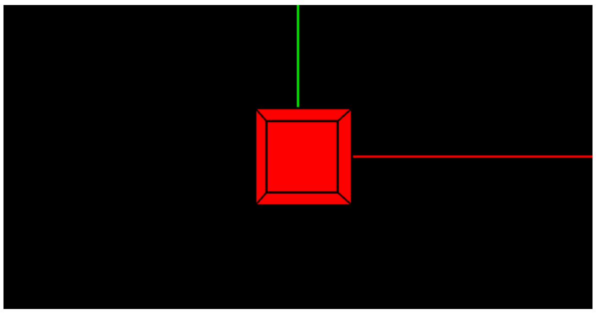
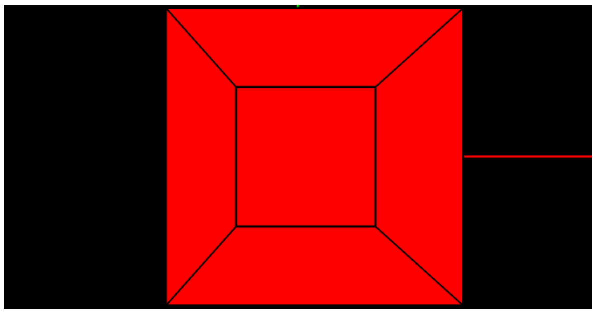
-x axis moving

-y axis moving

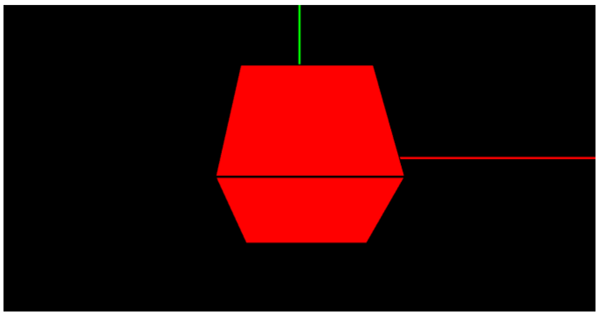
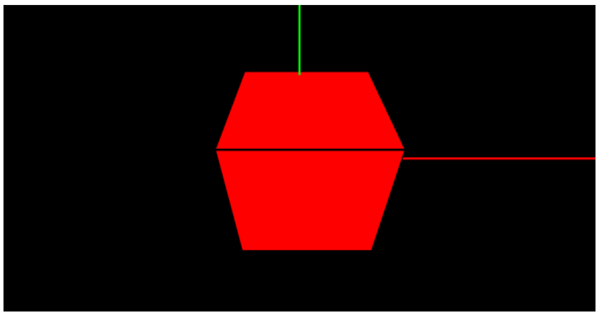



-z axis moving

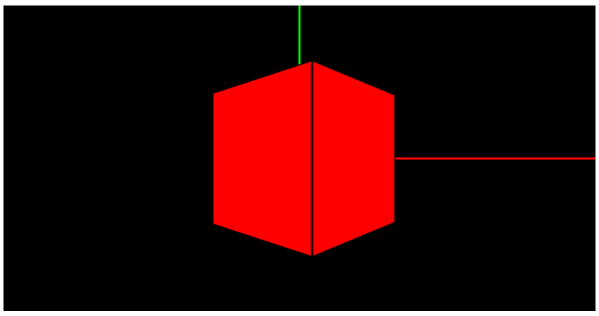
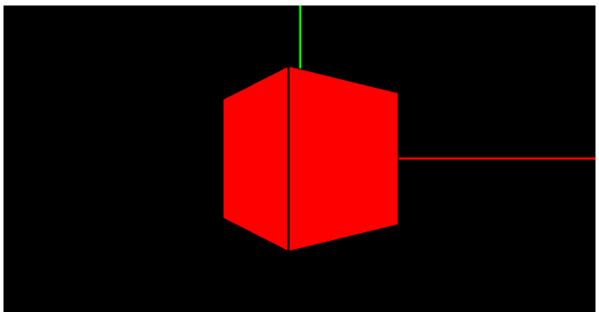





- When object rotates around x axis(or y axis, or z axis)

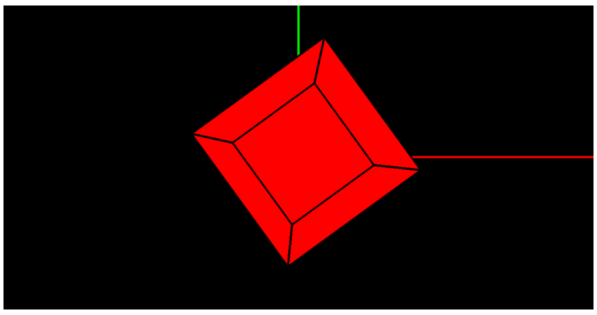
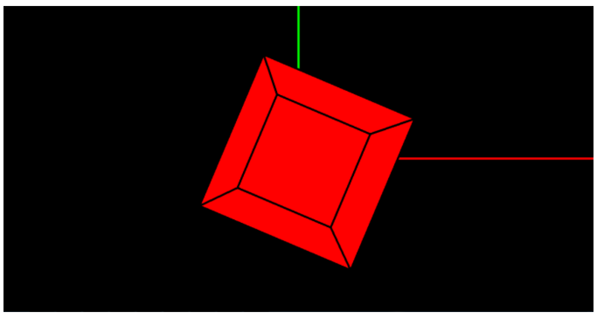
-x axis rotate

-y axis rotate

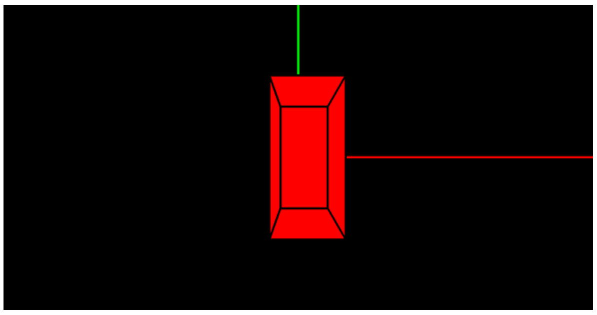
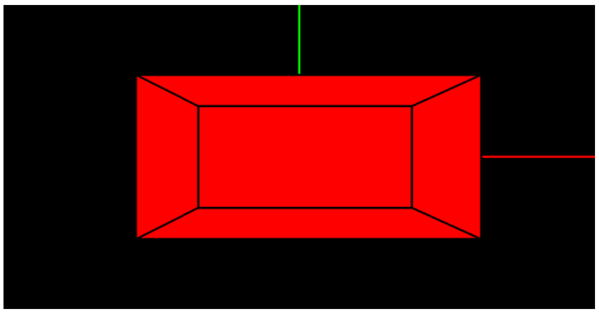



-z axis rotate

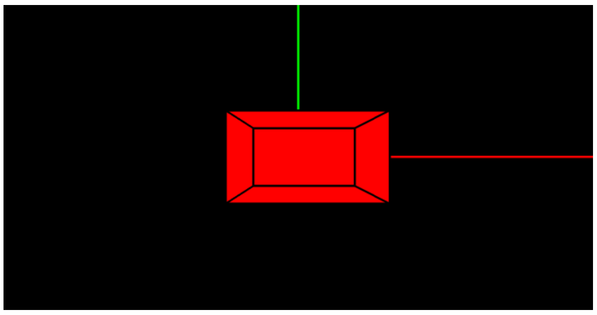
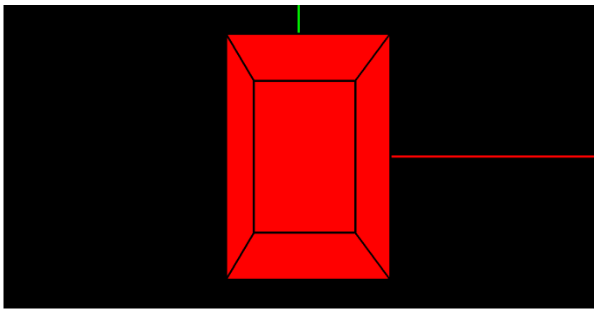



- When object scales centered by x axis(or y axis, or z axis)

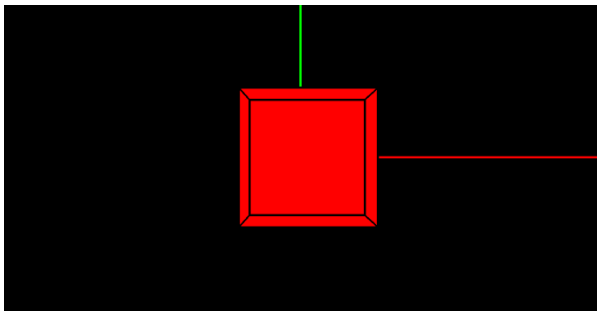
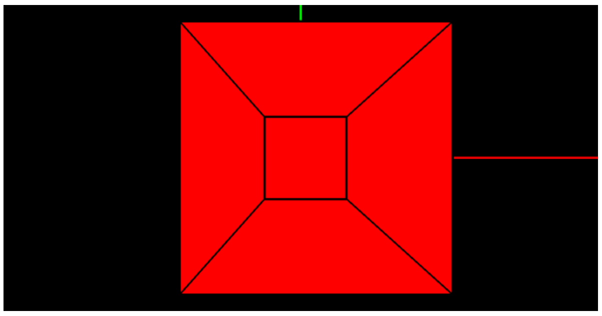
-x axis

-y axis

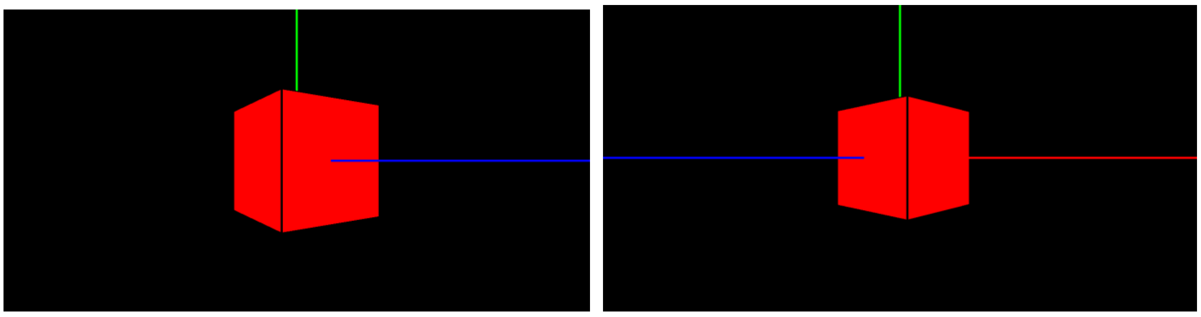



-z axis

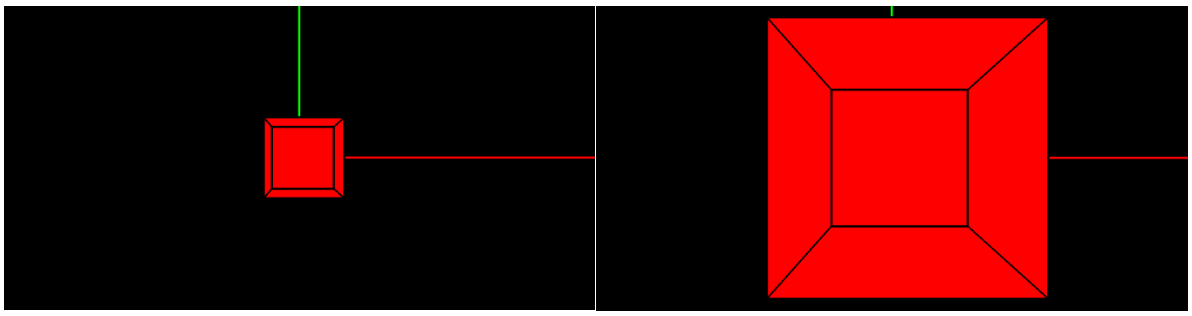



(2) With object fixed,

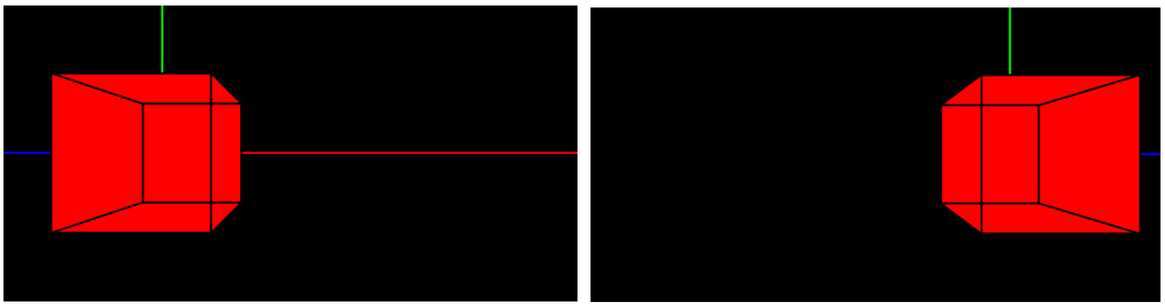
- Camera moving sideways while focusing on certain coordinate(for example, origin point)



- Camera moving closer to the object while focusing on certain coordinate(for example, origin point)

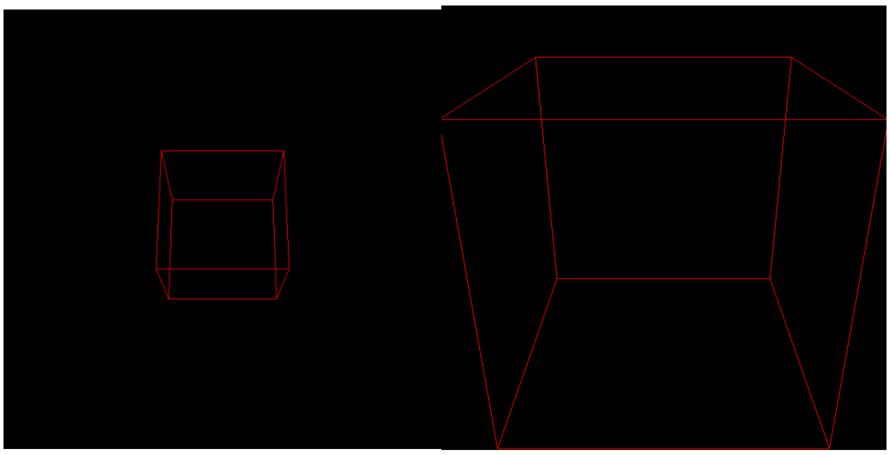


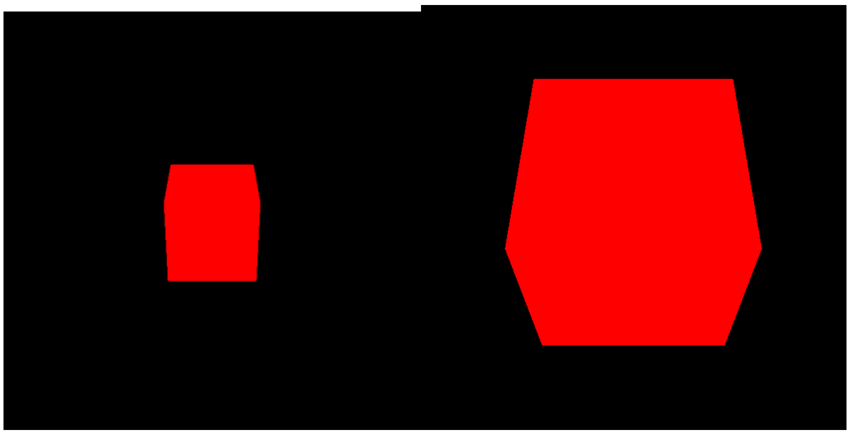
- Camera moving sideways with fixed VPN(View Plane Normal)



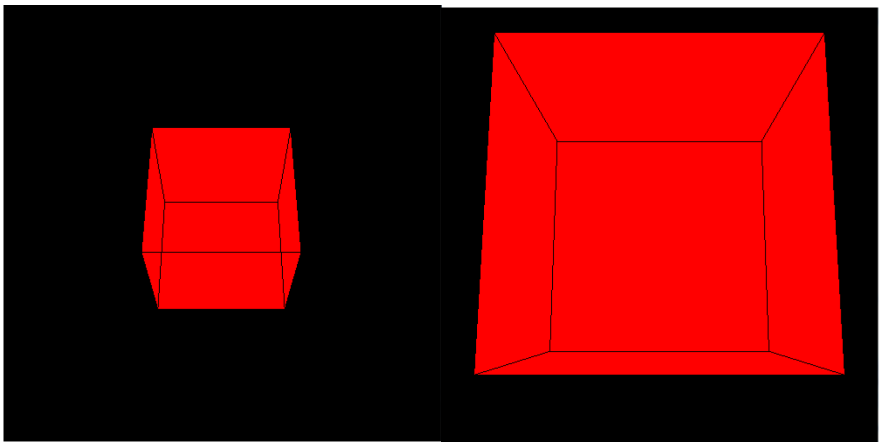
(3) Object rotating around x axis with constant speed, while camera move closer to the object , focusing on certain coordinate(for example, origin point)

-using glutIdleFunc()





-do not use glutIdleFunc()



## 2. Analysis and Discussion of Execution Result

In the first problem, x and y coordinates tx and ty after t seconds have GLdouble tx = v * cos (pi * theta / 180) * t;, GLdouble ty = v * sin (pi * theta / 180) * t 5 * t * t;. By running the for loop 19 times ( meaning that the cube moves for a total of 20 seconds, including outputting the initial cube before the for loop ), the tx and ty values are updated each time through the for loop.

After glPushMatrix (), pushes each matrix through translate, rotate, and scale. After that, the matrix for translate, rotate, and scale is shown through glPopMatrix () and disappears. tx, ty are updated by the input variable values: v, theta, s, and alpa (meaning initial velocity, firing angle, scale constant and rotation constant, respectively). In addition, when t = 1, the for loop running while increasing t by 1 until t is less than 20. Therefore, glRotatef and glScalef, where tx and ty values are not directly transmitted, should be multiplied by t respectively.

The output shows a smooth parabolic trajectory when the initial velocity is 60, the launch angle is 45, the scale constant is 1.1, and the rotation constant is 30. I can adjust the Look At Point by adjusting gluLookAt. I also print out parabolic trajectories with and without gluLookAt. A large initial speed slowed down the movement of the object. Thus, the parabolic trajectory becomes long. Smaller initial speeds make the object move faster. Thus the trajectory of the parabola is shortened. If the firing angle is large, the object is shot high. Smaller firing angles lower the object. The larger the scale constant, the faster the object grows. If the scale constant is small, the object grows slowly. If the rotation constant is small, the object rotates slowly. If the rotation constant is large, the object rotates quickly.

After several attempts, a parabolic trajectory was drawn, with the cube perfectly touching the ground, with an initial velocity of 60, a launch angle of 55, a scale constant of 1.2, and a rotation constant of 40. ( assuming do not adjusted gluLookAt )

--- * ---

In the second problem, to move the object in the x, y, z-axis direction while fixing the position of the camera and the viewing point, vec3_t trans = {0., 0., 0.}; is used. I used the trans [0], trans [1], and trans [2] to check the movement of objects on each axis. Adding a negative number to trans [0] moves the object to the left in the negative direction of the x-axis. Adding a positive number to trans [0] moves the object to the right, in the positive direction of the x-axis. Adding a negative number to trans [1] causes the object to move downward, in the negative direction of the y-axis. Adding a positive number to trans [1] moves the object upward in the positive direction of the y-axis. Adding a negative number to trans [2] moves the object to the back of the screen in the negative direction of the z axis. Adding a positive number to trans [2] moves the object toward the front of the screen in the positive z-direction.

To rotate the object in the x, y, z direction with the camera's position and viewing point fixed, vec3_t rot = {0., 0., 0.}; is used. rot [0], rot [1], rot [2] is used to check the rotation of the object in each axis. rot [0] rotates the x-axis, rot [1] rotates the y-axis, and rot [2] rotates the z-axis.

To change the size of the object in the x, y, and z-axis directions while fixing the position of the camera and the viewing point, vec3_t scale = {0.3, 0.3, 0.3}; is used. I used the scale [0], scale [1], and scale [2] to determine the change in size of the object on each axis. The initial value of scale is 0.3. When the size changes along the x-axis, the object becomes longer or shorter on both sides. When the size changes on the y-axis, the object becomes longer or shorter up or down. When the size changes to the z axis, the object becomes longer or shorter back and forth.

With the object at rest, the camera used the arrow keys to move the camera sideways while still looking at the origin of the object. Pressing the left or right arrow key moves the camera left or right while increasing the value of eye [0] of vec3_t eye = {0., 0., -50.};. eye [0] adjusts the x-axis.

Using the arrow keys to move the camera closer and closer toward the object while still looking at the object's origin with the object still. Pressing the up and down arrow keys moves the camera back and forth while increasing and decreasing the value of eye [2] in vec3_t eye = {0., 0., -50.};. eye [2] adjusts the z axis.

In order to move the camera to the side with the VPN locked while the object is stationary, the eye [0] and center [0] values are adjusted. gluLookAt (eye [0], eye [1], eye [2], center [0], center [1], center [2], 0, 1, 0); In the eye, the eye is the observer's eye, and the center is the direction the camera looks. Since the VPN is fixed, so I just move the eye and center.

As the object rotates at a constant speed around the x-axis, the camera must move closer and closer to the object while continuing to look at the object's specific location. To do this, I created a new project and coded it. I created three kind of cubes, solidcube, wirecube, and a combination of the wire and solid cube. Initially, I used glutIdleFunc () to rotate the object. The eye [2], ie, the z-axis, was adjusted to bring the camera closer to the rotating object. But with glutIdleFunc (), I couldn't stop the rotating object in the middle. break or keyboard input was not applicable. So I wrote the second code that doesn't use glutIdleFunc (). The second also adjusts the value of eye [2] ( the initial value is 100 ) to bring the camera closer to the object. Continue to add negative numbers to eye [2] to decrease the z-axis value so the camera gets closer to the object. After setting the initial value of the variable to count to 100, it continues to decrease and stops when the value is less than 20. If adjust the degree of camera access, can adjust this value. Larger counts shorten the distance the camera approaches the object, and smaller counts increase the distance the camera approaches the object. Thus the camera gets closer to the object.

In addition, when problem 2 was solved, the x, y, and z axes were output to the screen in the colors of R, G, and B. I could see the movement of the object more accurately.