Computer Graphics HW02

소프트웨어학부
소프트웨어학과
2017012197
여채린

문영식 교수님

1. Source code for the main part of the program

- Problem01

```
void filltriangle(){
    //random color
    glColor3ub(rand()%256 , rand()%256 , rand()%256);
    glLineWidth(rand()%10); //선굵기
    float x = rand()%225;
    float y = rand()%225;
    float r = rand()%50;

    //float x = rand()%(bx+1-ax)+ax;
    //float y = rand()%(by+2-ay)+ay;
    //float x = rand_FloatRange(cos(60)*r,225-cos(60)*r);
    //float y = rand_FloatRange(sin(30)*r,225-sin(30)*r);

    //삼각형 세 정점 좌표들 계산
    printf("\n triangle coor : %f %f %f",x,y,r);
    float c_x = x;
    float c_y = y+r;
    float b_x = x + r*cos(60);
    float b_y = y - r*0.5;
    float a_x = x - r*cos(60);
    float a_y = y- r*0.5;

    glBegin(GL_POLYGON);
    glVertex2f(c_x, c_y);
    glVertex2f(b_x, b_y);
    glVertex2f(a_x, a_y);
    glEnd();
}
```

```
void unfilltriangle(){
    //random color
    glColor3ub(rand()%256 , rand()%256 , rand()%256);
    glLineWidth(rand()%10); //선굵기
    float x = rand()%225;
    float y = rand()%225;
    float r = rand()%50;
    //float x = rand()%(bx+1-ax)+ax;
    //float y = rand()%(by+2-ay)+ay;
    //float x = rand_FloatRange(cos(60)*r,225-cos(60)*r);
    //float y = rand_FloatRange(sin(30)*r,225-sin(30)*r);

    //삼각형 세 정점 좌표를 계산
    printf("\n triangle coor : %f %f %f",x,y,r);
    float c_x = x;
    float c_y = y+r;
    float b_x = x + r*cos(60);
    float b_y = y - r*0.5;
    float a_x = x - r*cos(60);
    float a_y = y- r*0.5;

    glBegin(GL_LINE_LOOP);
    glVertex2f(c_x, c_y);
    glVertex2f(b_x, b_y);
    glVertex2f(a_x, a_y);
    glEnd();
}
//GL_LINE_LOOP는 정점끼리 선으로 이음
//GL_POLYGON은 안을 채움
void line(){
    glColor3ub(rand()%256 , rand()%256 , rand()%256);//random color
    glLineWidth(rand()%10); //선굵기
    float x1 = rand()%225;
    float y1 = rand()%225;
    float x2 = rand()%225;
    float y2 = rand()%225;
    printf("\nline coor : %f %f\n",x1,y1);
    //printf("\nline coor : %f %f\n",x2,y2);
    GLint k;
    glBegin(GL_LINES);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glEnd();
}
```

```
void unfillsquare()
{
    glColor3ub(rand()%256 , rand()%256 , rand()%256);//random color
    glLineWidth(rand()%10); //선굵기
    float x = rand()%100;
    float y = rand()%100;
    double sidelength = rand()%100;
    double halfside = sidelength / 2;
    glBegin(GL_LINE_LOOP);//unfill
    glVertex2d(x + halfside, y + halfside);
    glVertex2d(x + halfside, y - halfside);
    glVertex2d(x - halfside, y - halfside);
    glVertex2d(x - halfside, y + halfside);
    glEnd();
}
void fillsquare()
{
    glColor3ub(rand()%256 , rand()%256 , rand()%256);//random color
    glLineWidth(rand()%10); //선굵기
    float x = rand()%100;
    float y = rand()%100;
    double sidelength = rand()%100;
    double halfside = sidelength / 2;
    glBegin(GL_POLYGON);//unfill
    glVertex2d(x + halfside, y + halfside);
    glVertex2d(x + halfside, y - halfside);
    glVertex2d(x - halfside, y - halfside);
    glVertex2d(x - halfside, y + halfside);
    glEnd();
}
//unfill circle
void unfillcircle() {
    glColor3ub(rand()%256 , rand()%256 , rand()%256);
    glLineWidth(rand()%10); //선굵기
    float dx = rand()%100;
    float dy = rand()%100;
    float r = rand()%100;
    //printf("unfillcircle coor : %f %f\n",x,y);
    glBegin(GL_LINES);
    for (int i = 0; i < 180; i++)
    {
        double x = r * cos(i) + dx;
        double y = r * sin(i) + dy;
        glVertex3f(x, y, 0);

        x = r * cos(i + 0.1) + dx;
        y = r * sin(i + 0.1) + dy;
        glVertex3f(x, y, 0);
    }
    glEnd();
}
```

```
void fillcircle() {
    glColor3ub(rand()%256 , rand()%256 , rand()%256);
    glLineWidth(rand()%10); //선굵기
    float dx = rand()%100;
    float dy = rand()%100;
    float r = rand()%100;
    //printf("unfillcircle coor : %f %f\n",x,y);
    glBegin(GL_POLYGON);
    for (int i = 0; i < 180; i++)
    {
        double x = r * cos(i) + dx;
        double y = r * sin(i) + dy;
        glVertex3f(x, y, 0);

        x = r * cos(i + 0.1) + dx;
        y = r * sin(i + 0.1) + dy;
        glVertex3f(x, y, 0);
    }
}
void displayFcn(){
    GLfloat tx = rand(), ty = rand();
    GLfloat sx = rand(), sy = rand();
    GLdouble theta = pi/2.0;
    glClear(GL_COLOR_BUFFER_BIT);
    int i;
    for (i=0; i<num; i++){
        isfill = rand()%2;
        shape = rand() % 4;
        if(shape==0){    //선
            line();
            printf("\n선\n");
        }
        else if(shape==1){  //정삼각형
            printf("\n정삼각형 \n");
            if(isfill == 1)
                filltriangle();
            else
                unfilltriangle();
        }
        else if(shape==2){  //정사각형
            printf("\n정사각형 \n");
            if(isfill==1)
                fillsquare();
            else
                unfillsquare();
        }
```

```
        else if(shape==3){  //원
            printf("\n원 \n");
            if(isfill == 1)
                fillcircle();
            else
                unfillcircle();
        }
    }
    glFlush();
}
```

- Problem02

  - Do not use matrix to do translate, rotate, and scaling.

```
void translatePolygon(wcPt2D*verts, GLint nVerts, GLfloat tx, GLfloat ty){
    GLint k;
    for(k=0; k<nVerts; k++){
        //Move x by tx, move y by ty
        verts[k].x = verts[k].x + tx;
        verts[k].y = verts[k].y + ty;
    }
}

void rotatePolygon_v2(wcPt2D*verts, GLint nVerts, wcPt2D pivPt){
    GLint k;
    GLfloat t = alpa* 3.14 / 180;
    for(k=0; k<nVerts; k++){
        //Move to origin,
        //and then rotate
        //Move back to the original position
        wcPt2D temp = verts[k];

        verts[k].x = verts[k].x - pivPt.x;
        verts[k].y = verts[k].y - pivPt.y;

        temp.x = verts[k].x * cos(t) - verts[k].y * sin(t);
        temp.y = verts[k].x * sin(t) + verts[k].y * cos(t);
        verts[k] = temp;

        verts[k].x = verts[k].x + pivPt.x;
        verts[k].y = verts[k].y + pivPt.y;
    }
}
```

```
void scalePolygon_v2(wcPt2D*verts, GLint nVerts, wcPt2D pivPt, GLfloat sx, GLfloat sy){
    //Perform pivot scale
    GLint k;
    for(k=0; k<nVerts; k++){
        //scale x by sx, scale y by sy
        verts[k].x = verts[k].x - pivPt.x;
        verts[k].y = verts[k].y - pivPt.y;

        verts[k].x = verts[k].x * sx;
        verts[k].y = verts[k].y * sy;

        verts[k].x = verts[k].x + pivPt.x;
        verts[k].y = verts[k].y + pivPt.y;
     }
}

void triangle(wcPt2D* verts){
    GLint k;
    glBegin(GL_LINE_LOOP);
    for(k=0; k<3; k++)
        glVertex2f(verts[k].x, verts[k].y);
    glEnd();
}
void displayFcn(){
    int t;
    GLint nVerts=3;
    wcPt2D tri_1[3] = {{25.0, 0.0},{15.0, 10.0},{35.0, 10.0}};
    wcPt2D centroidPt;
    centroidPt.x = 0;
    centroidPt.y = 0;
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 0.0, 0.0);
    triangle(tri_1);
    for(t=1; t<10; t++){
        //matrix3x3SetIdentity(matComposite);
        GLdouble tx = v*cos(pi*theta/180)*t;
        GLdouble ty = v*sin(pi*theta/180)*t - 5*t*t;//pivot포인트 바뀌는거 업데이트 ->센트로이드
        // 180나누는건, 라디안으로 바꿔줘야 해서.
        translatePolygon(tri_1, 3, v*cos(pi*theta/180), 5-10*t + v*sin(pi*theta/180));

        centroidPt.x = tx;
        centroidPt.y = ty;//업데이트 할 때마다 tx ty만큼. 중점좌표 옮김

        rotatePolygon_v2(tri_1, 3, centroidPt);

        scalePolygon_v2(tri_1, 3, centroidPt, s, s);
        triangle(tri_1);
    }
    glFlush();
}
```

- Use matrix to do translate, rotate, and scaling

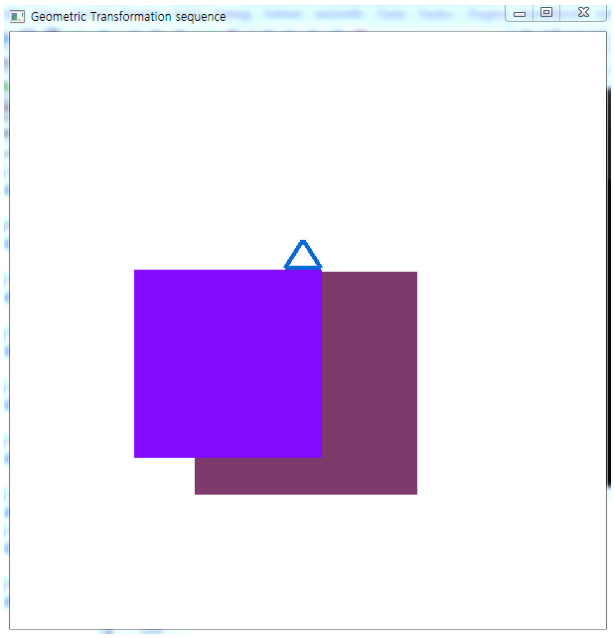
```
void matrix3x3SetIdentity(Matrix3x3 matIdent3x3){
    GLint row, col;
    for(row=0; row<3; row++){
        for(col=0; col<3; col++)
            matIdent3x3[row][col]=(row==col);
    }
}

void matrix3x3PreMultiply(Matrix3x3 m1, Matrix3x3 m2){
    GLint row, col;
    Matrix3x3 matTemp;

    for(row=0; row<3; row++){
        for(col=0; col<3; col++)
            matTemp[row][col]=m1[row][0]*m2[0][col] + m1[row][1]*m2[1][col] + m1[row][2]*m2[2][col];
    }

    for(row=0; row<3; row++){
        for(col=0; col<3; col++)
            m2[row][col]=matTemp[row][col];
    }
}
void translate2D(GLfloat tx, GLfloat ty){
    Matrix3x3 matTransl;
    matrix3x3SetIdentity(matTransl);

    matTransl[0][2]= tx;
    matTransl[1][2]= ty;

    matrix3x3PreMultiply(matTransl, matComposite);
}

void rotate2D(wcPt2D pivotPt, GLfloat theta){
    Matrix3x3 matRot;
    matrix3x3SetIdentity(matRot);
    matRot[0][0]=cos(theta);
    matRot[0][1]=-sin(theta);
    matRot[0][2]=pivotPt.x*(1-cos(theta))+pivotPt.y*sin(theta);
    matRot[1][0]=sin(theta);
    matRot[1][1]=cos(theta);
    matRot[1][2]=pivotPt.y*(1-cos(theta))-pivotPt.x*sin(theta);

    matrix3x3PreMultiply(matRot, matComposite);
}
```

```
void scale2D(GLfloat sx, GLfloat sy, wcPt2D fixedPt){
    Matrix3x3 matScale;
    matrix3x3SetIdentity(matScale);

    matScale[0][0]= sx;
    matScale[0][2]=(1-sx)*fixedPt.x;
    matScale[1][1]= sy;
    matScale[1][2]=(1-sy)*fixedPt.y;

    matrix3x3PreMultiply(matScale, matComposite);
}

void transformVerts2D(GLint nVerts, wcPt2D* verts){
    GLint k;
    GLfloat temp;

    for(k=0; k<nVerts; k++){
        temp = matComposite[0][0]*verts[k].x + matComposite[0][1]*verts[k].y + matComposite[0][2];
        verts[k].y = matComposite[1][0]*verts[k].x + matComposite[1][1]*verts[k].y + matComposite[1][2];
        verts[k].x = temp;
    }
}
void triangle(wcPt2D* verts){
    GLint k;
    glBegin(GL_LINE_LOOP);
    for(k=0; k<3; k++)
        glVertex2f(verts[k].x, verts[k].y);
    glEnd();
}
void displayFcn(){
    GLint nVerts=3;
    wcPt2D verts[3]={{25.0, 0.0},{15.0, 10.0},{35.0, 10.0}};//삼각형의 정점 verts
    wcPt2D centroidPt;
    GLint k, xSum=0, ySum=0;
    centroidPt.x = 0;
    centroidPt.y = 0;
    wcPt2D pivPt, fixedPt;
    pivPt = centroidPt;
    fixedPt = centroidPt;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    triangle(verts);

    for(t=1; t<=10; t++){
        matrix3x3SetIdentity(matComposite);
        GLdouble tx = v*cos(pi*theta/180)*t;
        GLdouble ty = v*sin(pi*theta/180)*t - 5*t*t;//pivot포인트 바뀌는거 업데이트 ->센트로이드
        // 180나누는건, 라디안으로 바꿔줘야 해서.
        translate2D(v*cos(pi*theta/180), 5-10*t + v*sin(pi*theta/180));
        centroidPt.x = tx;
        centroidPt.y = ty;//업데이트 할 때마다 tx ty만큼. 중점좌표 옮김
        rotate2D(centroidPt, pi*alpa/180);
        scale2D(s, s, centroidPt);
        transformVerts2D(nVerts, verts);
        triangle(verts);
    }
    glFlush();
}
```

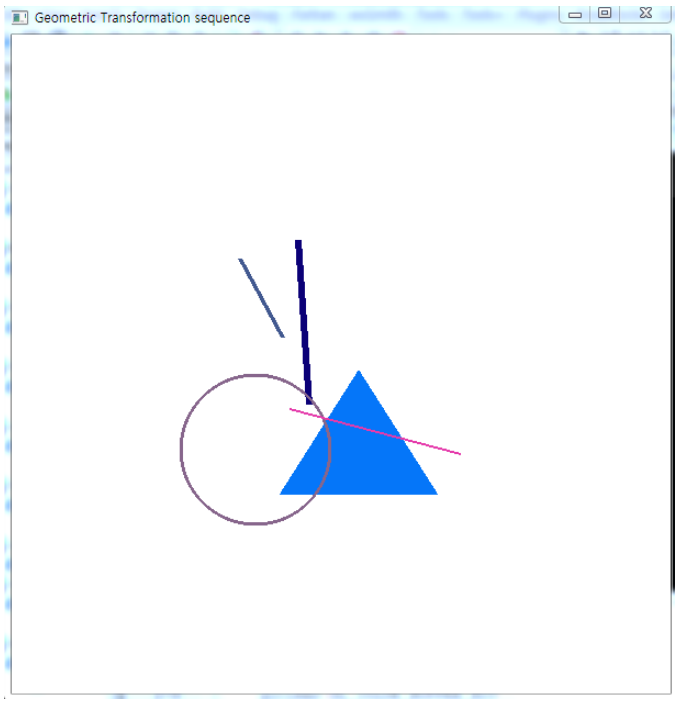
2. Screen capture to see the execution result

- Problem01

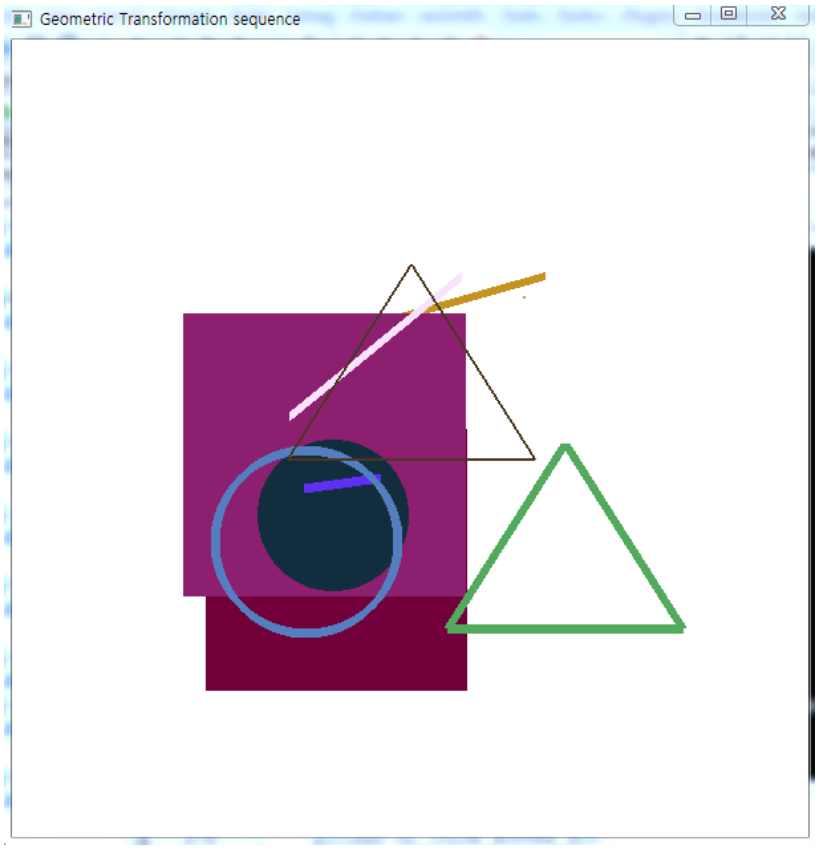
```
표시할 도형의 개수: 3
```

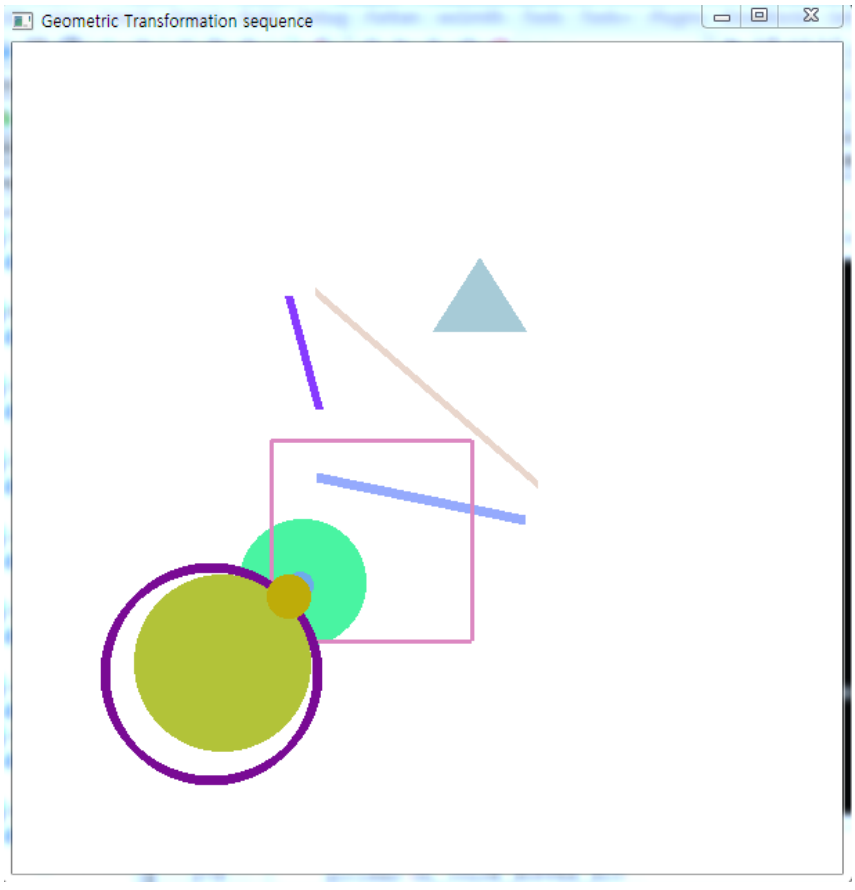



```
표시할 도형의 개수: 5
```

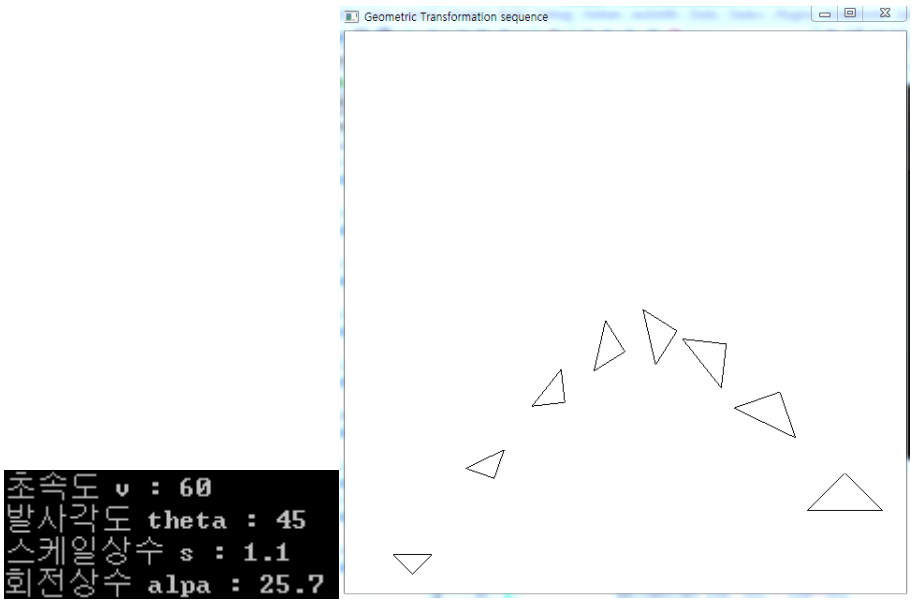


```
표시할 도형의 개수: 10
```

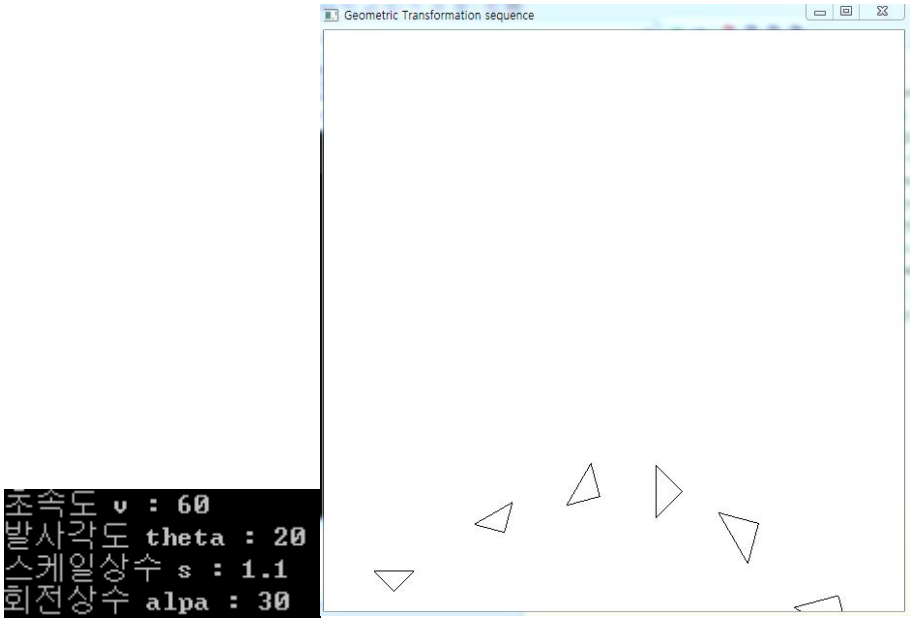

Geometric Transformation sequence


Geometric Transformation sequence

- Problem02

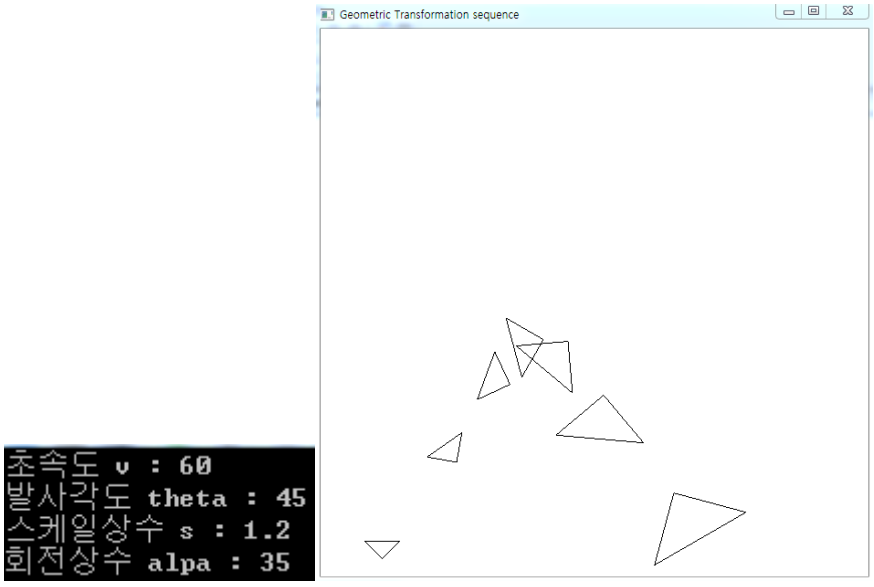


xwcMin=-10.0, xwcMax=280.0, ywcMin=-10.0, ywcMax=280.0



xwcMin=-10.0, xwcMax=280.0, ywcMin=-10.0, ywcMax=280.0

xwcMin=-10.0, xwcMax=300.0, ywcMin=-10.0, ywcMax=300.0

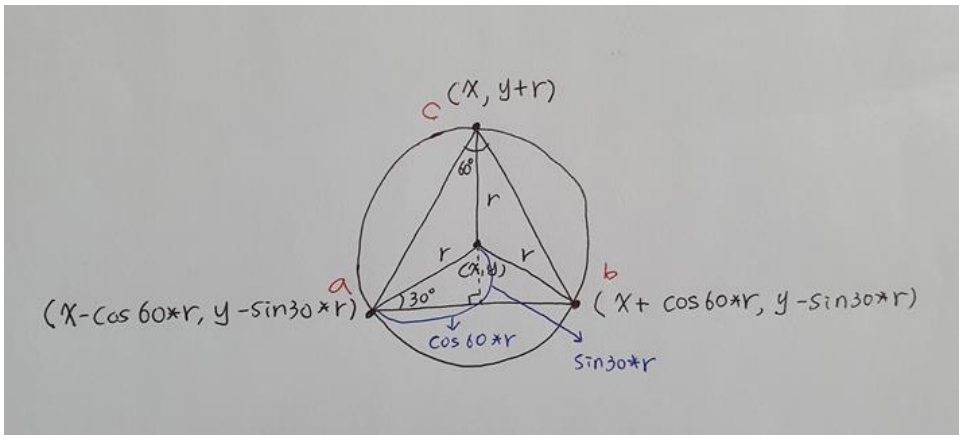


xwcMin=-10.0, xwcMax=300.0, ywcMin=-10.0, ywcMax=300.0

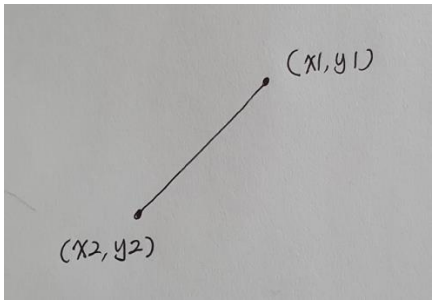
3. Analysis and Discussion of Execution Result

The first problem is that the displayFcn () function receives random values in the variables 'isfill' and 'shape'. 'isfill' takes a value of 0 or 1. If it is 0, the hollow figure is output. If it is 1, the filled figure is output. The variable 'shape' takes a number from 0 to 3 randomly. Output is a line if 0, an equilateral triangle if 1, a square if 2, and a circle if 3. Therefore, I implemented two types of functions for equilateral triangle, square, and circle. One is to print out a filled figure, and the other is to print out a hollow figure. Color and coordinate values were set randomly in the functions that output lines, equilateral triangles, squares, and circles. I used glColor3ub to paint colors randomly. At first, I tried to use glColor3f, but it was impossible to give rand ()% 256 as an argument. So I used glColor3ub to get the right value. To draw a hollow shape, I used GL_LINE_LOOP in glBegin (). GL_LINE_LOOP connects the start and end points. To draw a filled figure, I used GL_POLYGON in glBegin (). GL_POLYGON draws connected block polygons. I used glVertex2f () between glBegin () and glEnd () to insert the necessary vertices for each shape.

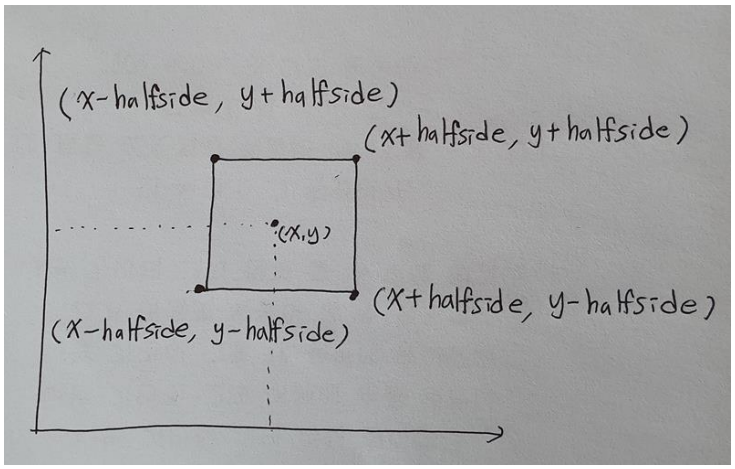
The following is how to implement the function to draw the figure for each figure.



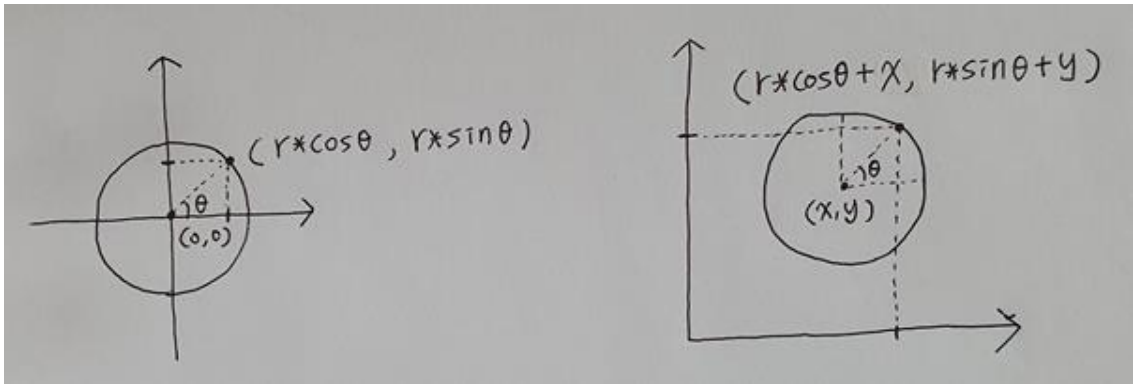
First, each triangle was found using the following properties. Using the properties of trigonometric functions, three coordinates a, b, and c that form an equilateral triangle can be found when the midpoint is randomly given. Assume that the given random midpoint coordinates are x and y. The radius is r. To draw an equilateral triangle, you need three values: x, y, and r. Then, the x and y coordinates of c become (x, y + r). The x and y coordinates of b are (x + r * cos60, y-r * sin30). The x and y coordinates of a are (x-r * cos60, y-r * sin30). When coding the coordinates thus calculated, one problem occurred. If the data type is float, the value of sin30 is displayed strangely. sin30 is 0.5, so we changed sin30 to 0.5 in the equation. Or you can see that the data type resolves to double. In the code above, we used a fallback method of 0.5.



Second, I will introduce how to draw lines. How to draw a line is the simplest. Set two coordinate values at random. The values of (x1, y1) and (x2, y2) are set randomly. We used GL_LINES to draw the line. You can link two different coordinates using GL_LINES.

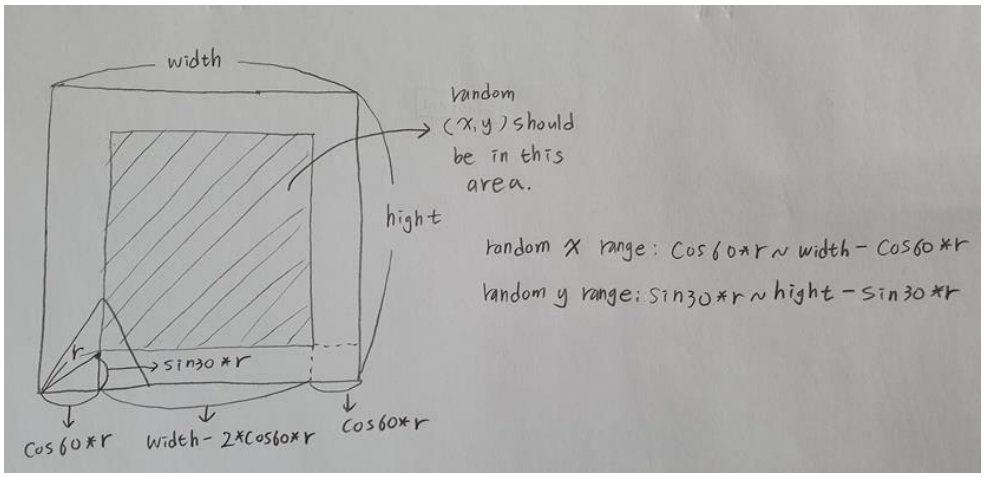


Third, I will introduce how to draw a square. First, the values of x, y, and 'sidelength' are randomly received. Create a 'halfside' by dividing the 'sidelength' value in half. Then use 'halfside' to create the coordinates for the four vertices of the square.



Fourth, I will introduce how to draw a circle. Generate random midpoint coordinates with variables dx and dy. Create a random radius value with the variable r. You can use trigonometric functions to calculate specific coordinates around the circle. The x value is r * cos (theta) and the y value is r * sin (theta). This value should be changed each time the midpoint coordinates are randomly selected, so add r * cos (theta) and r * sin (theta) to the dx and dy values received at random. Theta is expressed here as i. Add theta values by 0.1 and print the coordinates around the circle

as the angle increases.



Also, I wanted to make sure that the shape would not exceed the overall window size. Therefore, I calculated the range of coordinate values that each figure can have randomly. The above picture is the process of calculating the random coordinate values without departing the window size for the triangle. In conclusion, this calculation did not apply. To find a random value within a range of a and b, use rand ()% (b + 1-a) + a. However, rand () could not calculate the real range. Therefore, I can calculate the real value using ((b-a) * ((float) rand () / RAND_MAX)) + a. But, I found a simpler and easier way. It is possible to adjust the values of the global variables xwcMin, xwcMax, ywcMin, ywcMax. The xwcMin and ywcMin values were set to -200. The xwcMax and ywcMax values were set to 400. If the range of random generation within the function that draws each figure is not within this range, the figure is created within the window size.

For Problem 2, I did two versions. The first version uses translate, rotate and scaling using matrices. The second version does not use the translate, rotate, and scaling using matrix.

Set the centroid point to 0 for both x and y. x and y are updated by tx and ty, respectively. tx and ty are already given in the problem. tx = v * cos (pi * theta / 180) * t and ty = v * sin (pi * theta / 180) * t - 5 * t * t. The reason for dividing theta by 180 is because we need to change it to radian. As time goes by (the time increases by 1 second), x and y move by tx and ty, respectively. The translate angle of theta is used to translate. As the object translate, the object must grow and grow at the same time. The rotation constant alpa is used to rotate at each point. The scale constant s is used to grow at each point.

After several experiments, small initial velocities shorten the x-axis of the parabolic motion of the object. An initial speed of about 60 was suitable for observing the motion of the object. The smaller the firing angle, the lower the parabolic trajectory. The larger the firing angle, and the higher the larger the parabolic trajectory. The scale constant was about 1.1. If the scale constant is

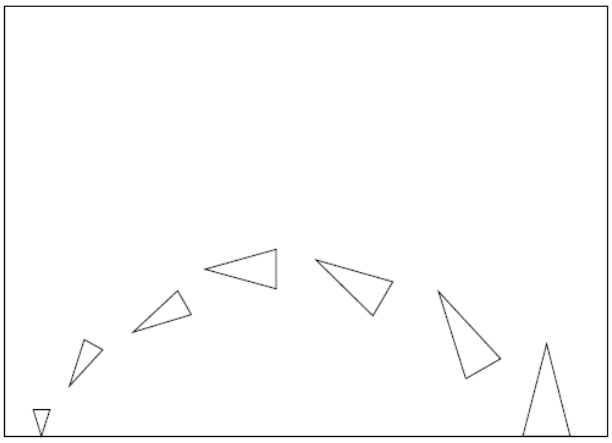
large, it is difficult to see the motion of the object at a glance. The rotational constant was good when its value range is about 20-30. If the rotation constant is small, the object turns slowly. The object turns little by little. If the window size is small, the object's movement is partially visible, so the object appears to be moving in a linear direction. Conversely, if the rotation constant is large, the movement of the object will not appear constant along the parabola, as shown in the example illustration of the problem.

There was a problem passing the arguments to the translate function. The gap between the movements of the objects was too big. To fix this, pass x (t) - x (t-1) as the argument to x. And, pass y (t) - y (t-1) as y factor. Each calculation is as follows. x (t)-x (t-1) = v * cos(pi * theta / 180) * t – v * cos(pi * theta / 180) * (t - 1) = v * cos (pi * theta / 180), y (t) - y (t - 1) = v * sin(p i * theta / 180) * t – 5 * t * t –v * sin(pi * theta / 180) * (t - 1) + 5 * (t - 1) * (t - 1) = 5-10 * t + v * sin (pi * theta / 180).

Also, the order of translate, rotate, and scale is important. Translate first, then rotate and scale. The object first moves its coordinates, then rotates and changes in size.

Also, I should adjust xwcMin and ywcMin values well. This is because the origin (0, 0) is in the center of the window. To observe the movement of the object, the starting point must be lower left in the window. To do this, I set xwcMin to -10 and ywcMin to -10. This prevents the figure from being drawn in only one quadrant in the window.

And range setting is important. At first time, instead of drawing an initial triangle at first, I used for (t = 0; t <10; t ++) and displayed object. The result is translate and scaling worked fine, but rotate did not. The reason for this is that the for loop was run without the initial triangle drawn. So I draw the initial triangle first, then modify the range with for (t = 1; t <= 10; t ++), and the triangle moving gradually from the beginning.



Also, if you want the object to rotate clockwise as shown in the problem, give it the negative value of theta. If theta is given without changing the sign(positive theta), the object rotates counterclockwise.

As much as possible, I tried to output similarly to the picture given, but various experiments were needed to experimentally derive the correct initial velocity, launch angle, scale constant, and rotation constant.

Also, isosceles triangles was used in the problem, but I think that using equilateral triangles can make a difference.