Transformations for Modeling & Animation

Open GL Lab 10

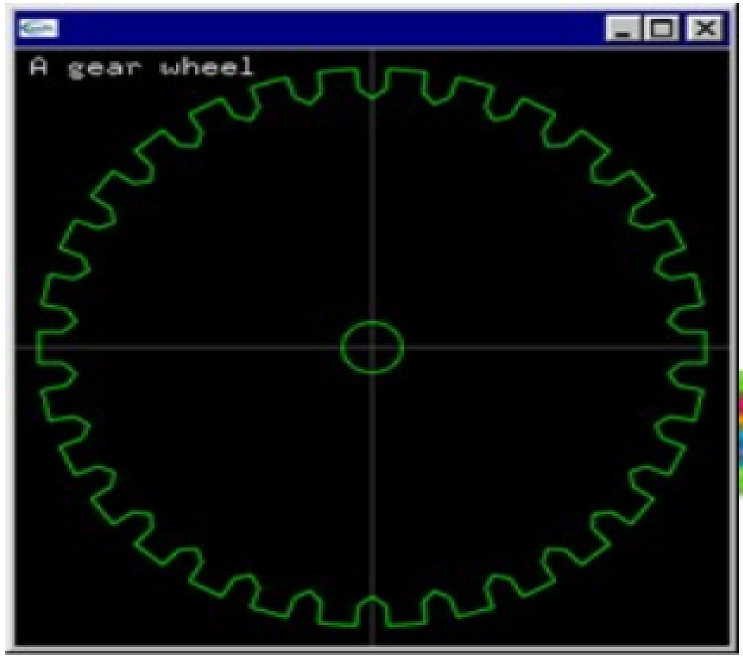
BSCS - 514 Computer Graphics
Instructor Humera

Lab Objectives / Tasks

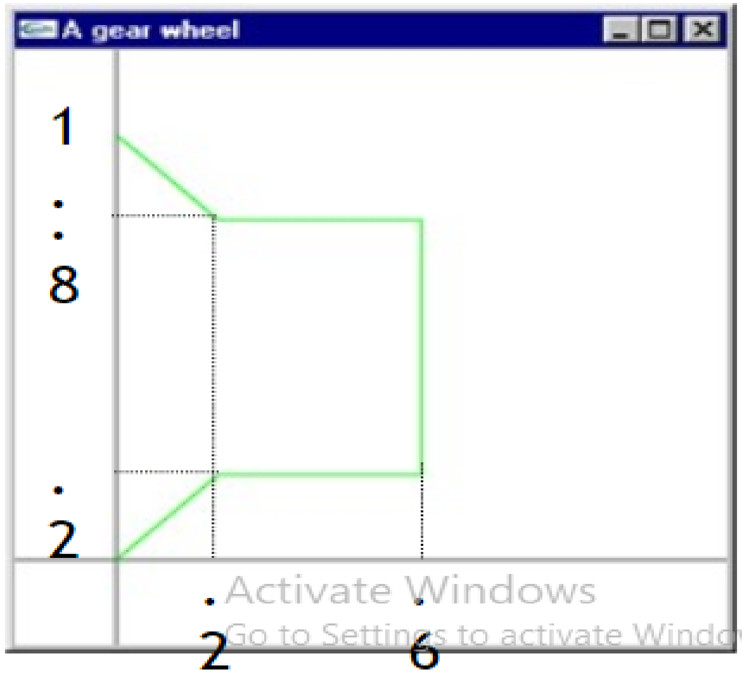
1. Modeling & Animating Gear Wheel
2. Drawing Symmetric Object
3. Making Patterns
4. Square & Hexagonal Tiling
5. Next More on Transformations

1. Modeling a gearwheel

- Drawing the axes, title and circle are easy
- There are 30 teeth. Each is transformed from a basic tooth

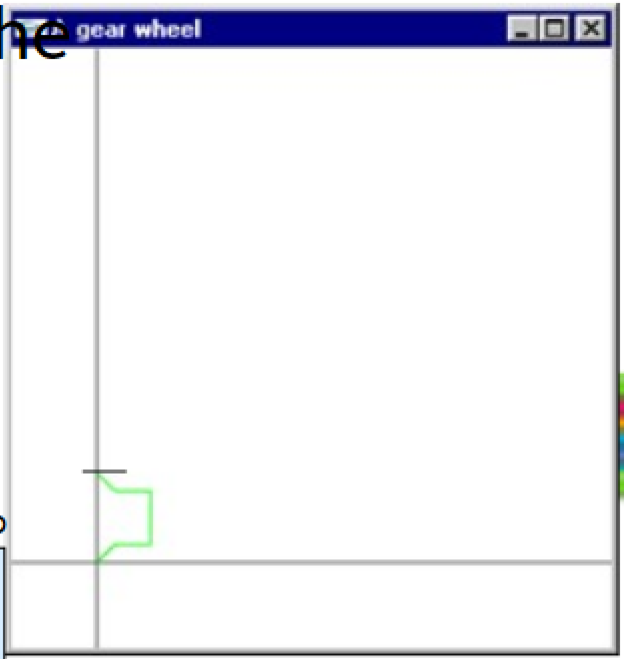



```
void  tooth0()  {
        glBegin( GL_LINE_STRIP);
                glVertex2f(0.0, 0.0);
                glVertex2f(0.2, 0.2);
                glVertex2f(0.6, 0.2);
                glVertex2f(0.6, 0.8);
                glVertex2f(0.2, 0.8);
                glVertex2f(0.0, 1.0);
                glEnd();
        }
```



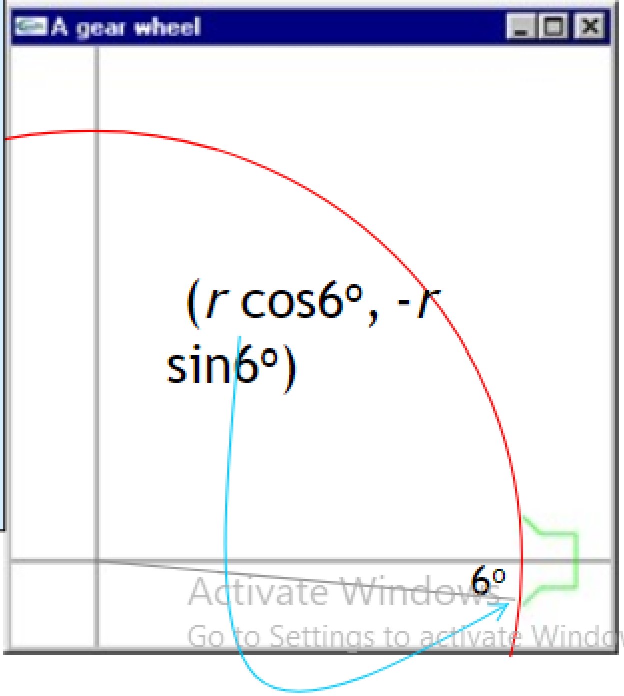
- To transform the basic tooth to the right tooth on the *x* axis
  - In both *x* and *y* dimensions, scale down by $2\,r \sin 6^\circ$
  - Translate the scaled tooth by $Tx = r \cos 6^\circ$, $Ty = -r \sin 6^\circ$

$2\,r \sin 6^\circ$

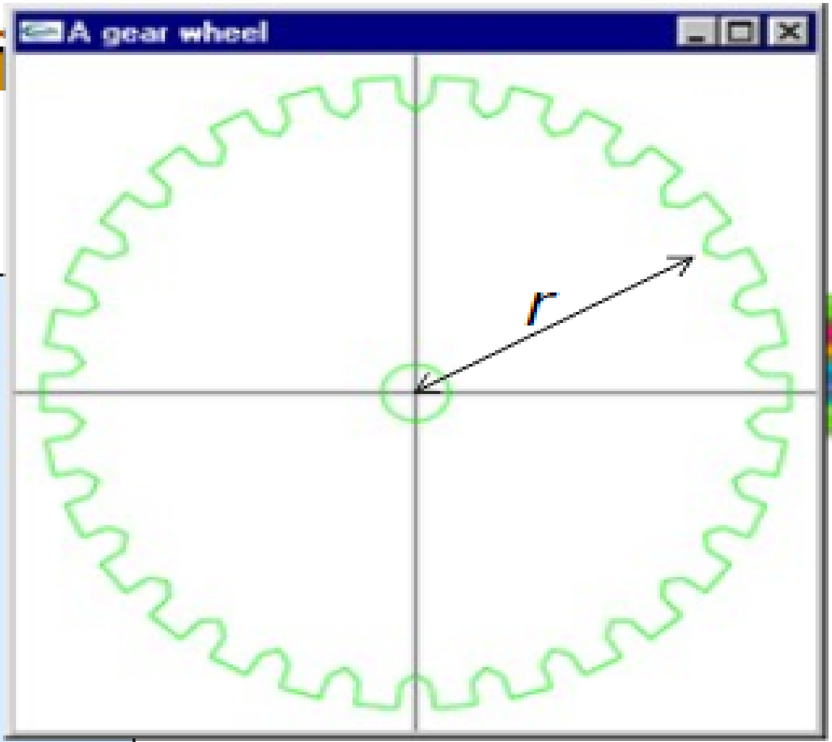



```
void  tooth1( double  r)  {
      double rad = 6.0 * 3.1416 / 180.0,
      sin6 = r * sin( rad), cos6 = r * cos( rad);

      glPushMatrix();
            glTranslatef( cos6, -sin6, 0.0);
            glScalef( 2.0*sin6, 2.0*sin6, 1.0);
            tooth0();
      glPopMatrix();
}
```



- To draw the entire set of teeth of radius *r* and centered at the origi




```
void gear( double r)
{
     glPushMatrix();
     for (int i=1; i<=30; ++i) {
          tooth1( r);
          glRotatef( 12.0, 0.0, 0.0, 1.0);
     }
     glPopMatrix();


}
```

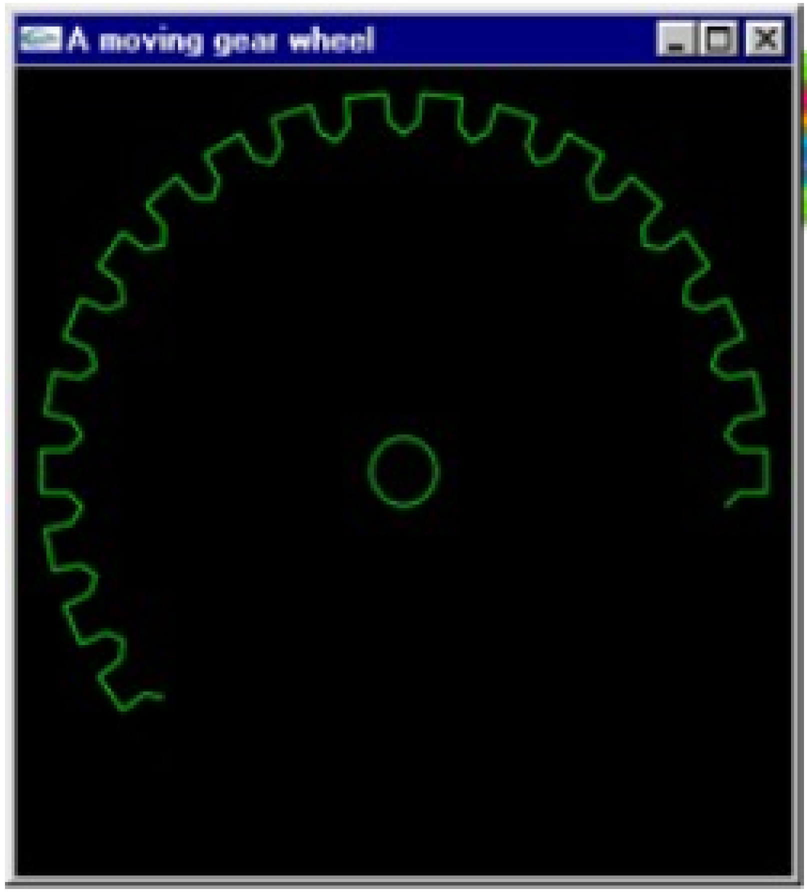
```
void  move()  {
//Standard Setup for animation

 float speed = 0.0001;
 static int oldTime = clock(), newTime;

 newTime = clock();
 deg += (newTime - oldTime) * speed;

 //printf("%d\n",newTime - oldTime);
 oldTime = newTime;

         glutPostRedisplay();
}
```



2. Drawing Symmetric Object

- It is easy to produce a complex snowflake by designing one half of a spoke, and drawing it 12 times.

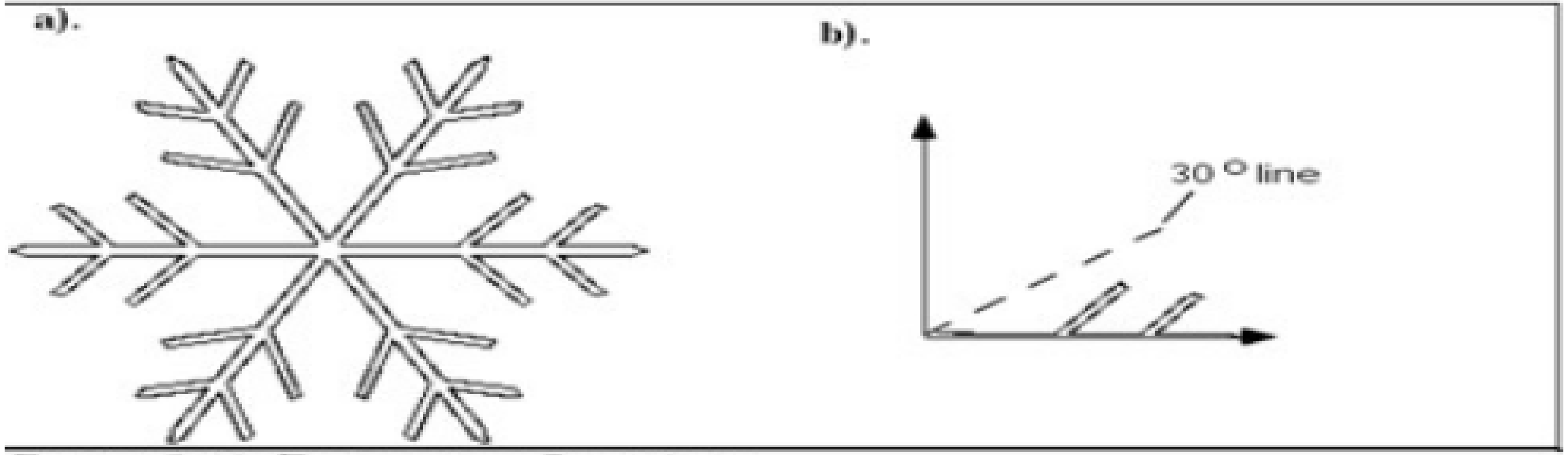


Figure 5.40. Designing a Snowflake.

2. Drawing Symmetric Object

- It is easy to produce a complex snowflake by designing one half of a spoke, and drawing it 12 times.

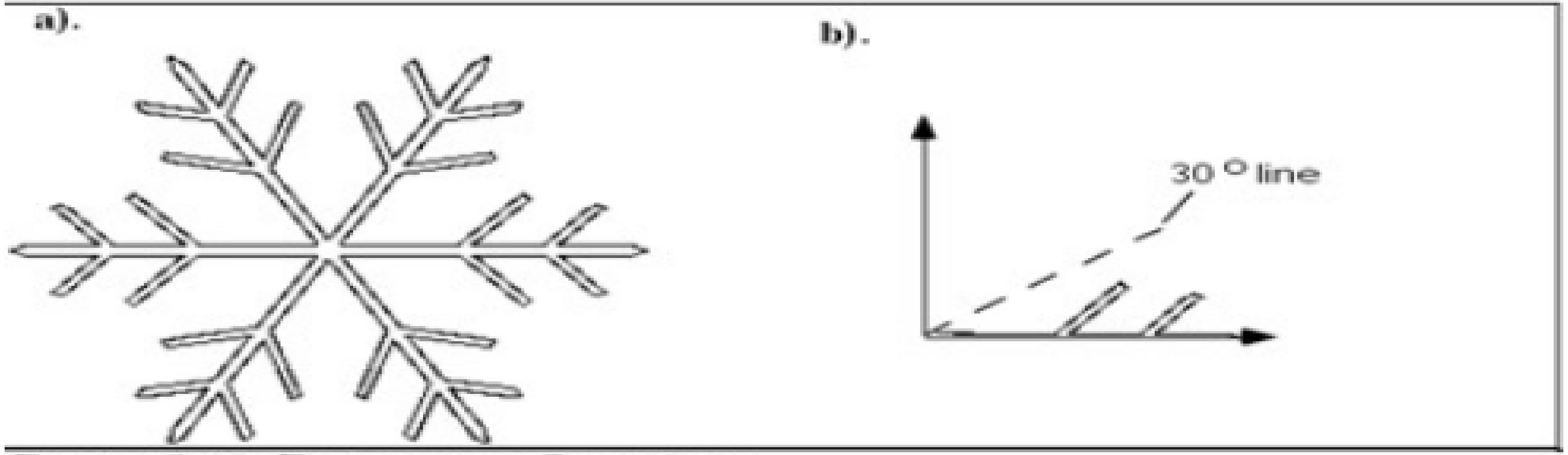


Figure 5.40. Designing a Snowflake.

Snowflake example continued....

- (a) # include "turtle.h"
- (b) gluOrtho2D(-10,10,-10,10)
- (c) L = 1 ; Implement void flakeMotif(float L)
- (d) Complete one spoke(reflection w.r.t x-axis)

```
flakeMotif();              // draw the top half
cvs.scale2D(1.0,-1.0);     // flip it vertically
flakeMotif();              // draw the bottom half
cvs.scale2D(1.0,-1.0);     // restore the original axis
```

- Draw entire snowflake

```
void drawFlake()
{
   for(int count = 0; count < 6; count++) // draw a snowflake
   {
         ?????????????????????
   }
}
```

```
void flakeMotif(float L)
{
    moveTo(0.0, 0.1*L);
    turnTo(0);
    forward(2*L, 1);
    turn(60);
    forward(1*L, 1);
    turn(270);
    forward(0.2*L, 1);
    turn(270);
    forward(0.9*L, 1);
    turn(120);
    forward(1*L, 1);
    turn(60);
    forward(0.9*L, 1);
    turn(270);
    forward(0.2*L, 1);
    turn(270);
    forward(0.8*L, 1);
    turn(120);
    forward(1*L, 1);
    turn(330);
    forward(0.2*L, 1);
    turn(30);
}
```

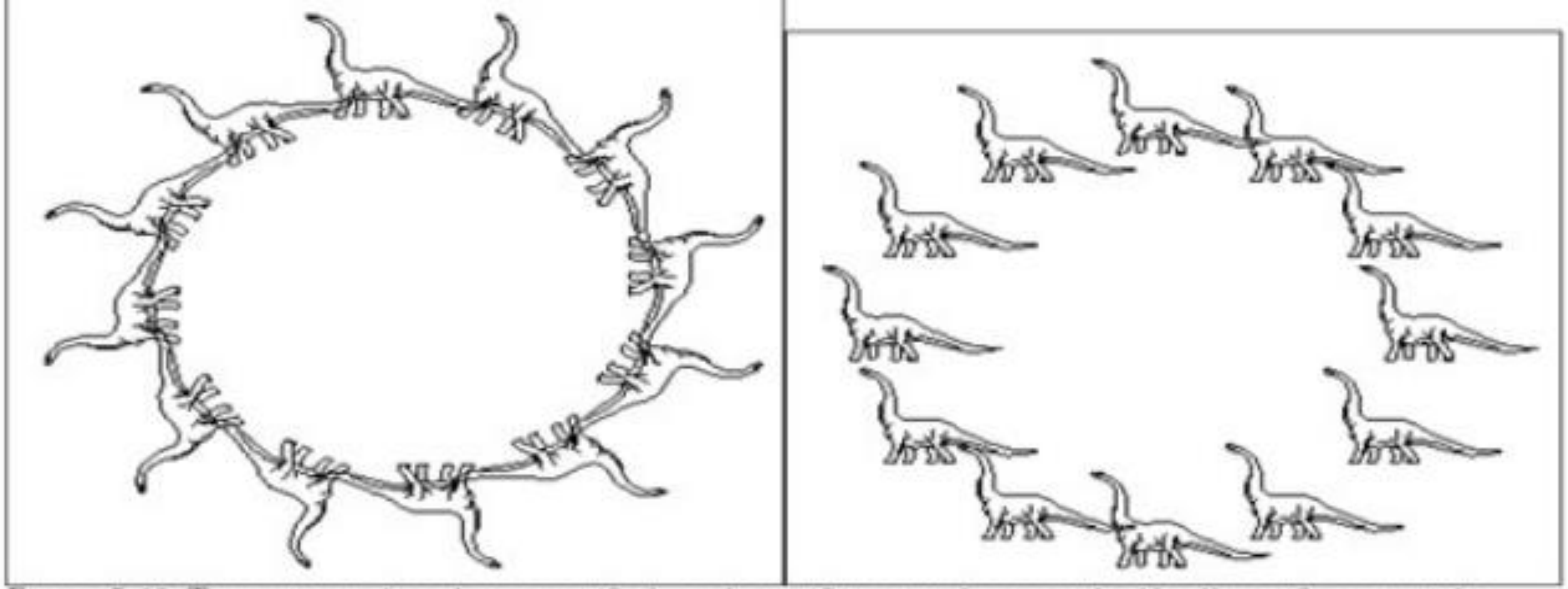


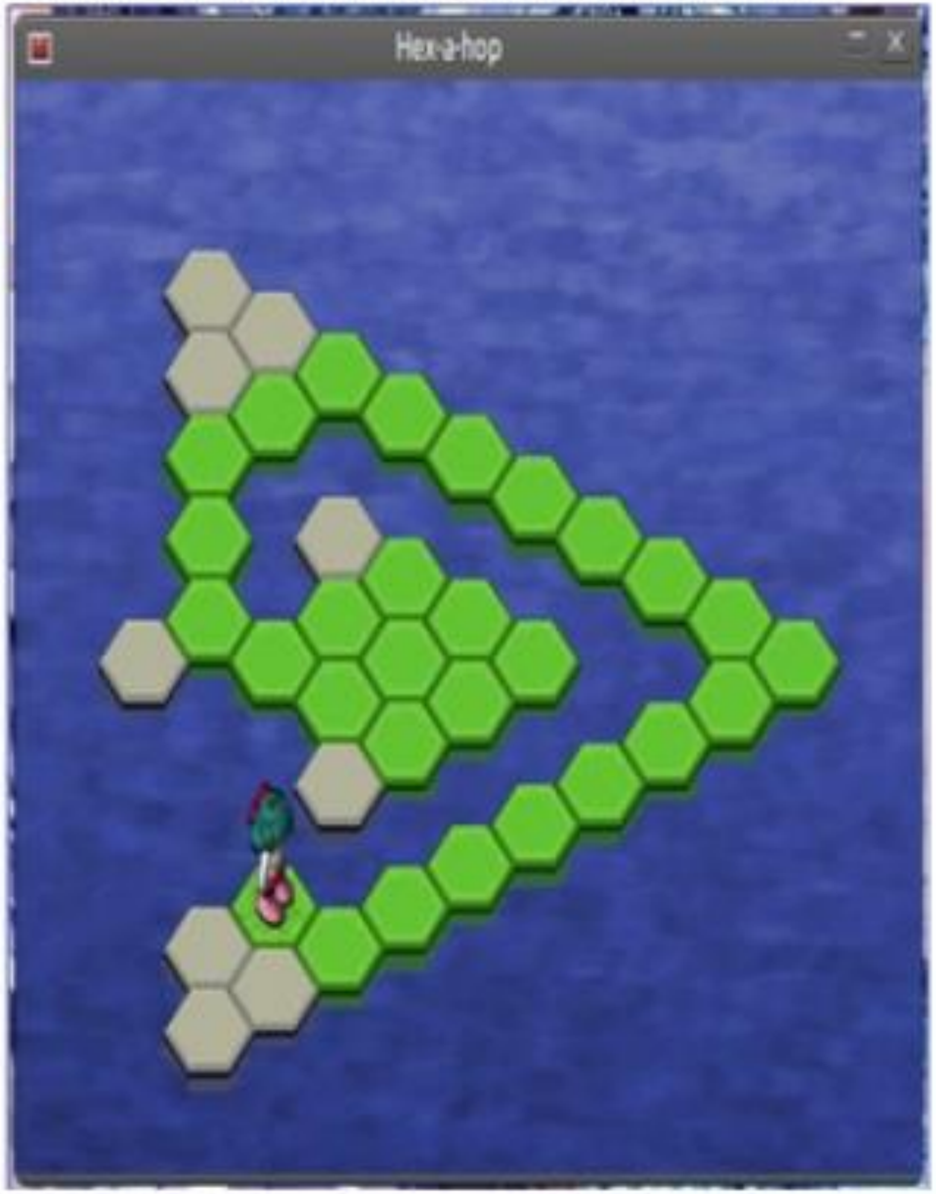
Figure 5.42. Two patterns based on a motif. a). each motif is rotated separately. b). all motifs are upright.

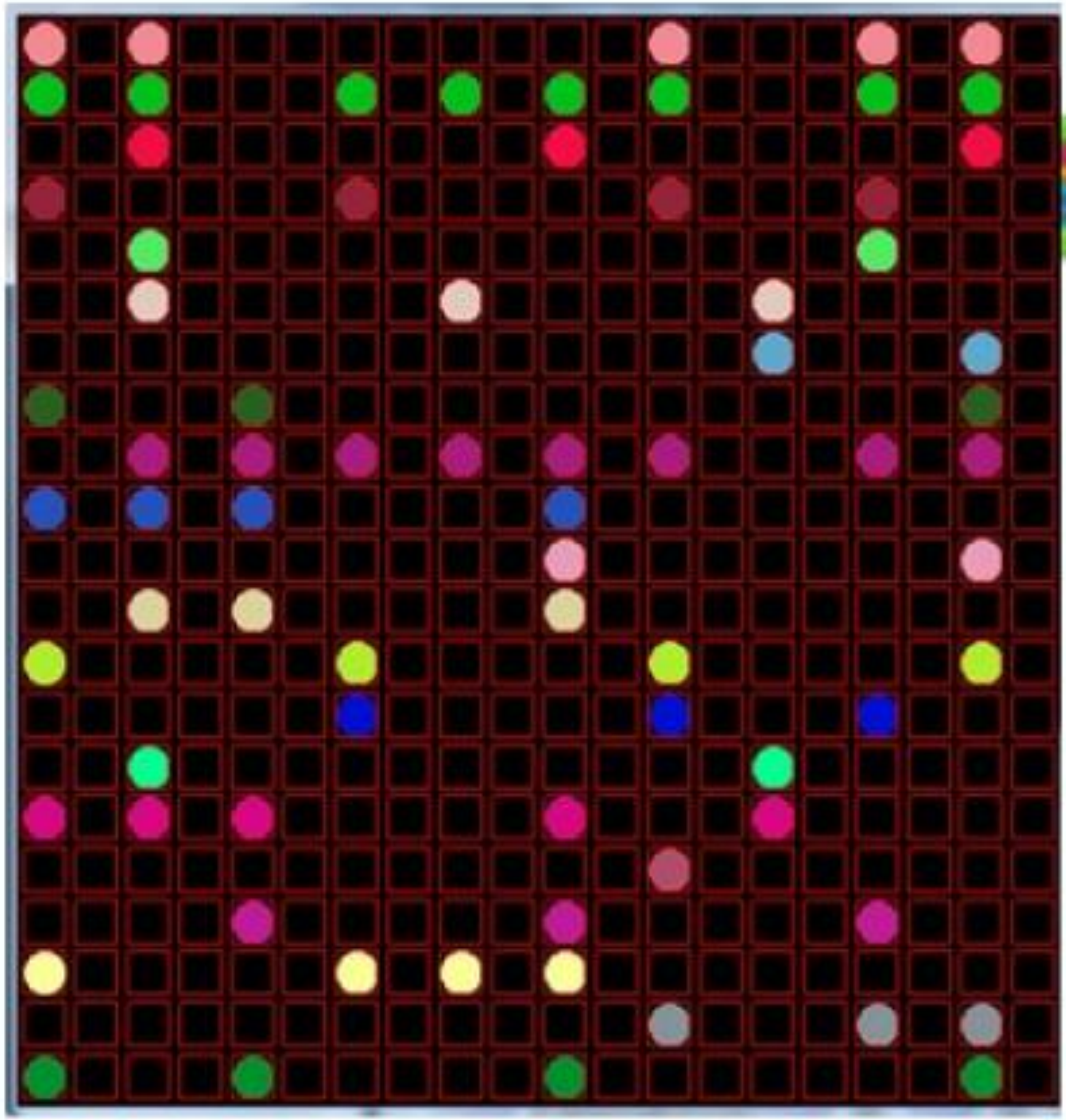
Suppose that `drawDino()` draws an upright dinosaur centered at the origin. In part a) the coordinate system for each motif is first rotated about the origin through a suitable angle, and then this coordinate system is translated along its y-axis by H units as shown in the following code. Note that the CT is reinitialized each time through the loop so that the transformations don't accumulate. (Think through the transformations you would use if instead you took the point of view of transforming points of the motif.)

```
const int numMotifs = 12;
for(int i = 0; i < numMotifs; i++)
{
      cvs.initCT(); // init CT at each iteration
      cvs.rotate2D(i * 360 / numMotifs); // rotate
      cvs.translate2D(0.0, H); // shift along y-axis
      drawDino();
}
```

An easy way to keep the motifs upright as in part b) is to "pre-rotate" each motif before translating it. If a particular motif is to appear finally at 120°, it is first rotated (while still at the origin) through -120°, then translated up by H units, and then rotated through 120°. What ajustments to the preceding code will achieve this?

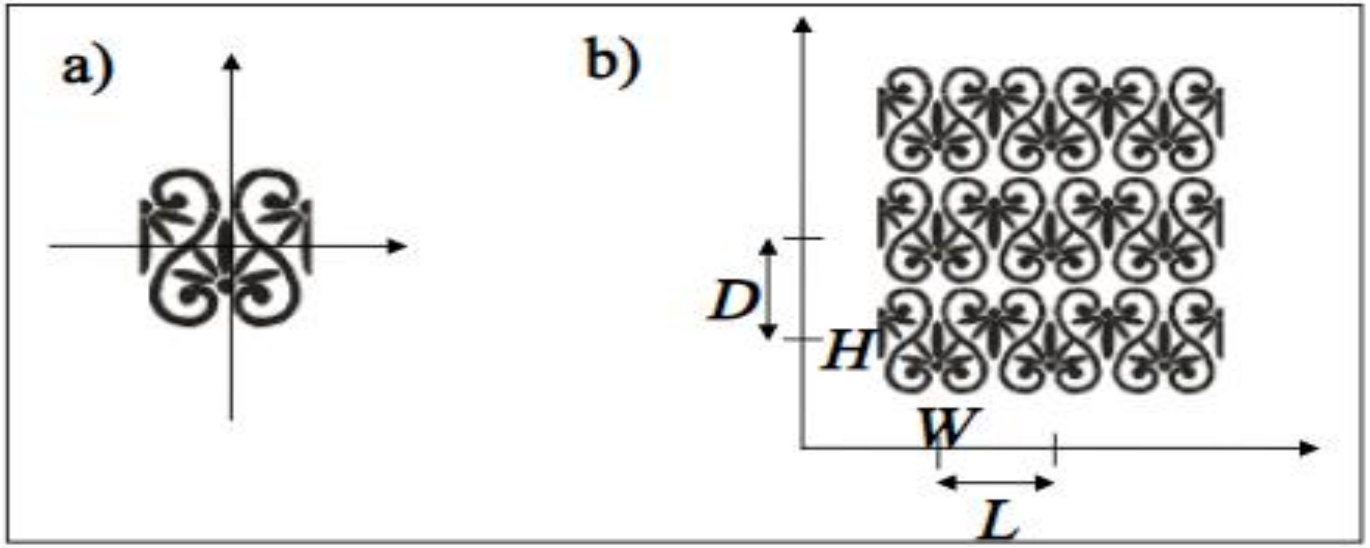
Hex-a-hop is a Fun Hexagon Tile-Based Game for Linux and Windows





Tiling example from book

Figure 5.46 shows how easily the coordinate system can be manipulated in a double loop to draw the tiling. The CT is restored after drawing each row, so it returns to the start of that row, ready to move up to start the next row. In addition, the whole block of code is surrounded with a pushCT() and a popCT(), so that after the tiling has been drawn the CT is returned to its initial value, in case more drawing needs to be done.



Tiling Code for fig.5.46

```
cvs.pushCT();                    // so we can return here
cvs.translate2D(W, H);           // position for the first motif
for(row = 0; row < 3; row++){    // draw each row
    pushCT();
    for(col = 0 ; col < 3; col++){
        motif();
        cvs.translate2D(L, 0);}  //move to the right
    cvs.popCT();                 // back to the start of this row
    cvs.translate2D(0, D); }//move up to the next row
cvs.popCT();    //back to where we started
```

Tiling Important exercise

5.5.3. A hexagonal tiling. A hexagonal pattern provides a rich setting for tilings, since regular hexagons fit together neatly as in a beehive. Figure 5.49 shows 9 columns of stacked 6-gons. Here the hexagons are shown empty, but we could draw interesting figures inside them.

a). Show that the length of a hexagon with radius R *is also* R.

b). Show that the centers of adjacent hexagons in a column are separated vertically by $\sqrt{3}\ R$ and adjacent columns are separated horizontally by $3\ R / 2$.

c). Develop code that draws this hexagonal tiling, using pushCT() and popCT() and suitable transformations to keep track of where each row and column start.

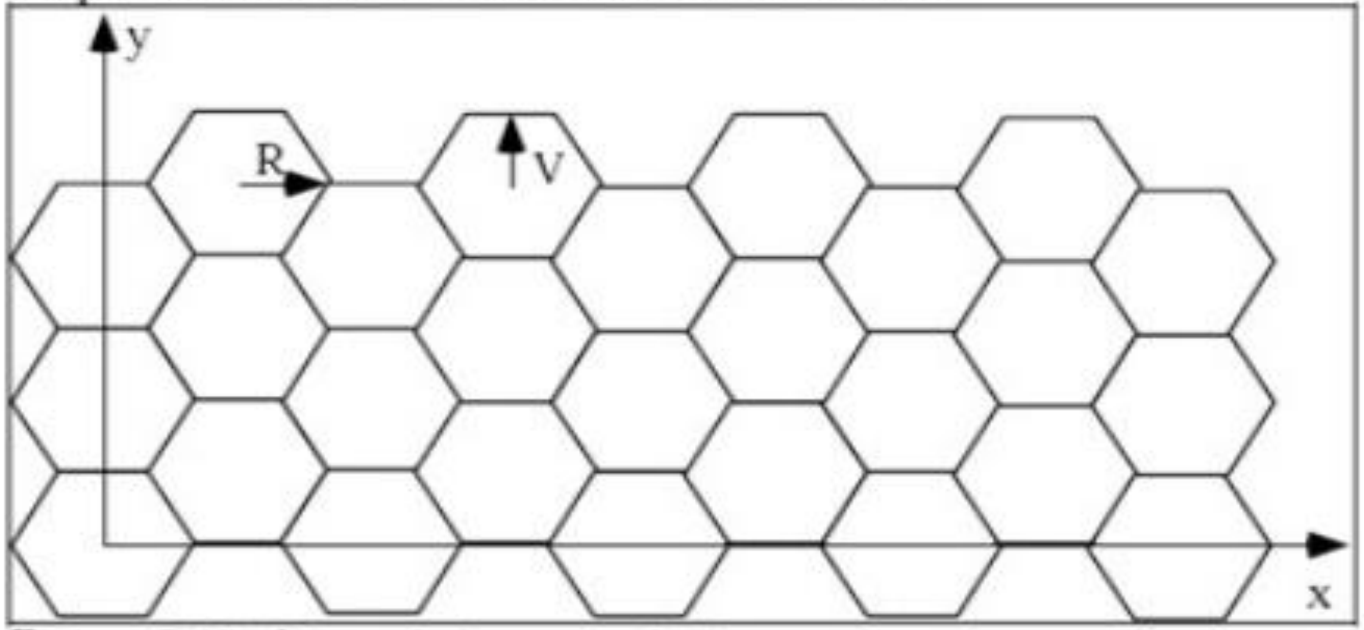


Figure 5.49. A simple hexagonal tiling.