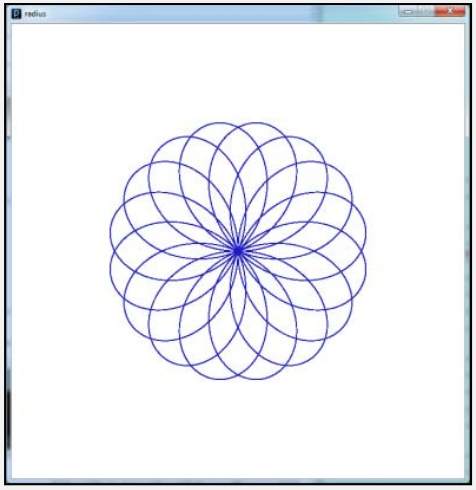
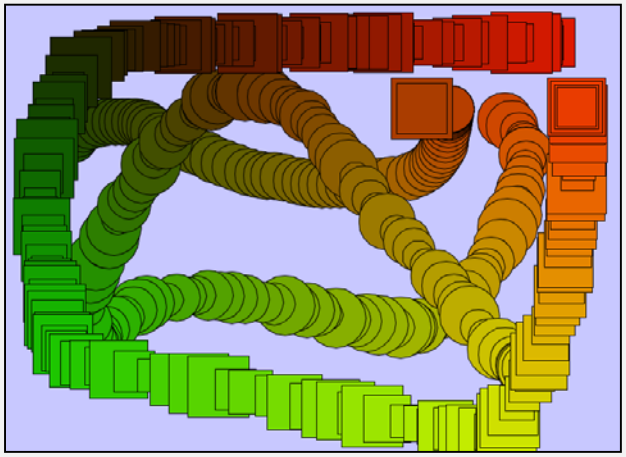
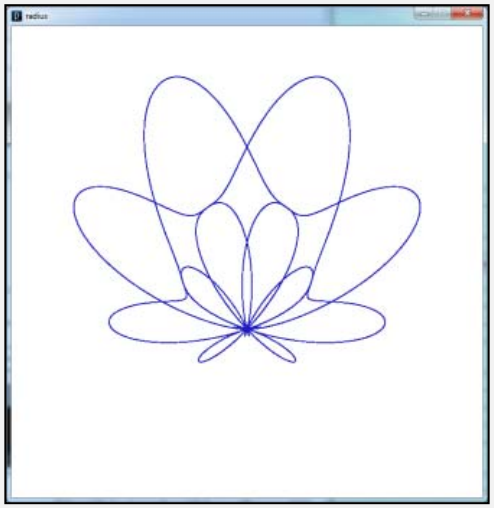
Drawing Circles and Other Regular Polygons







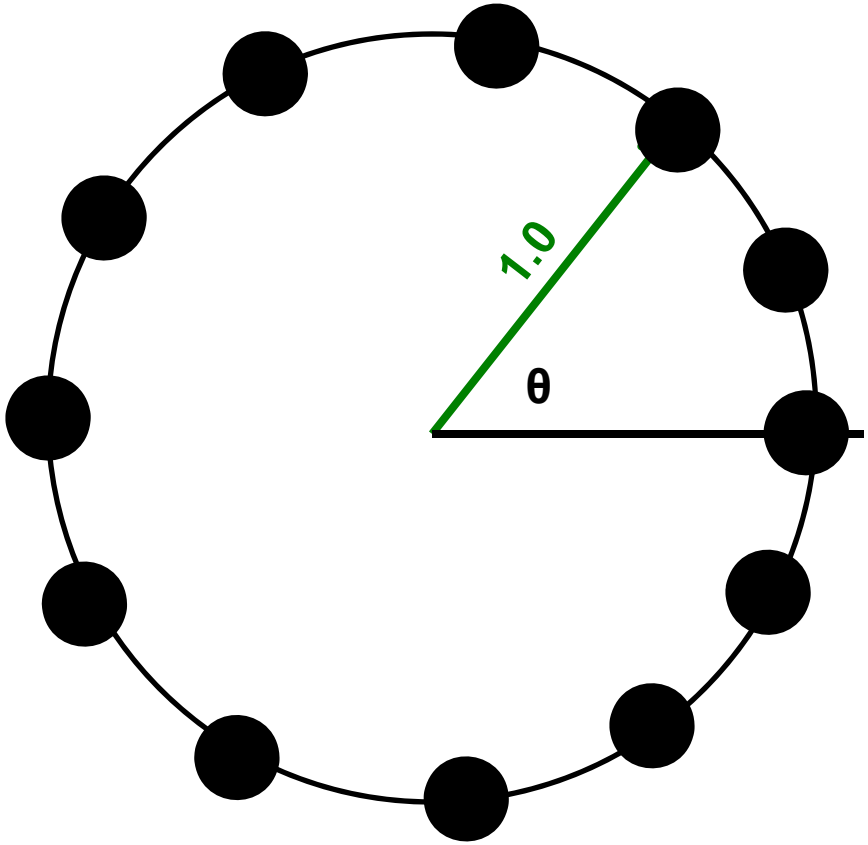
Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

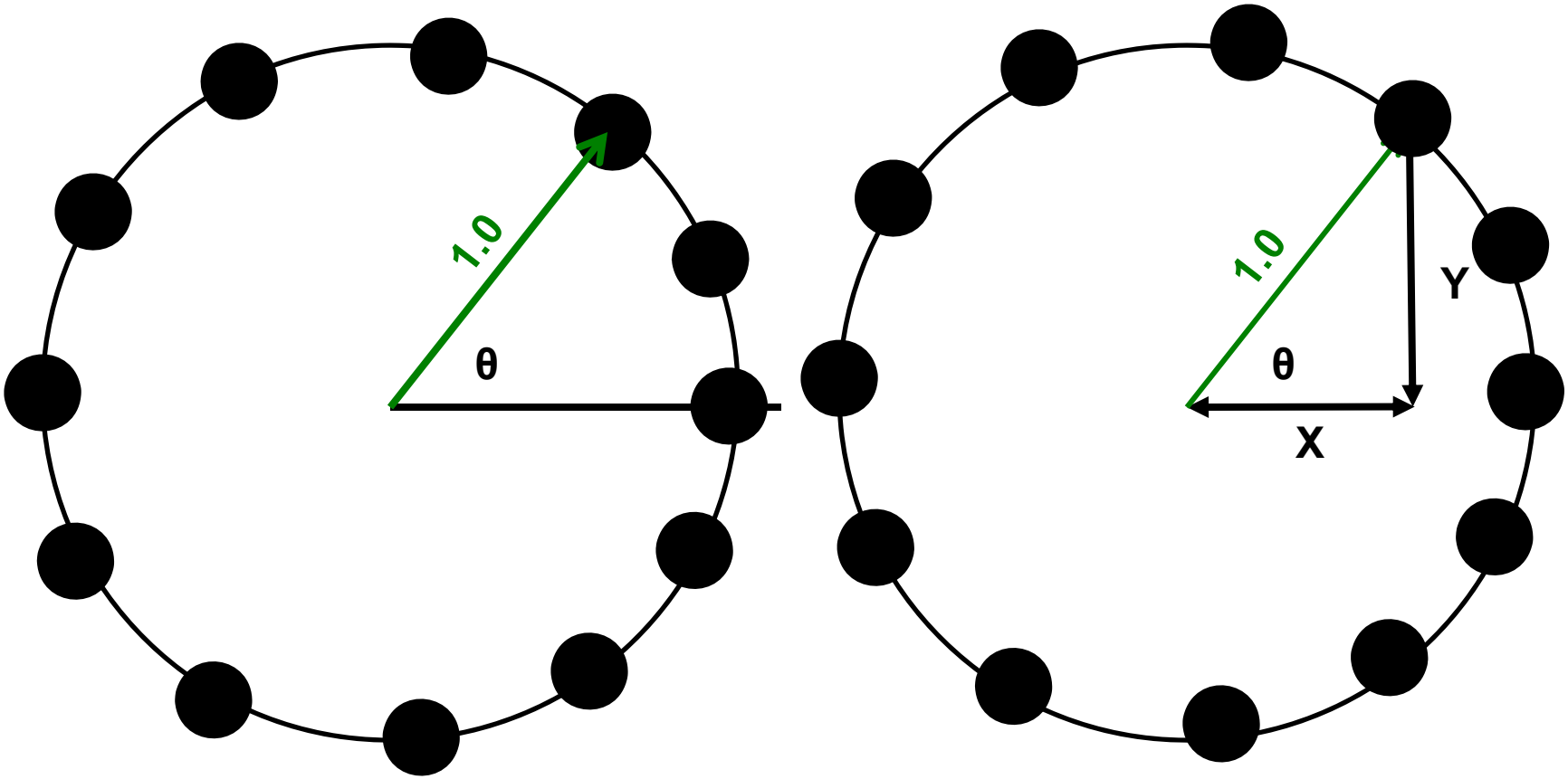
Oregon State University
Computer Graphics

First, We Need to Understand Something about Angles



If a circle has a radius of 1.0, then we can march around it by simply changing the angle that we call **θ.**

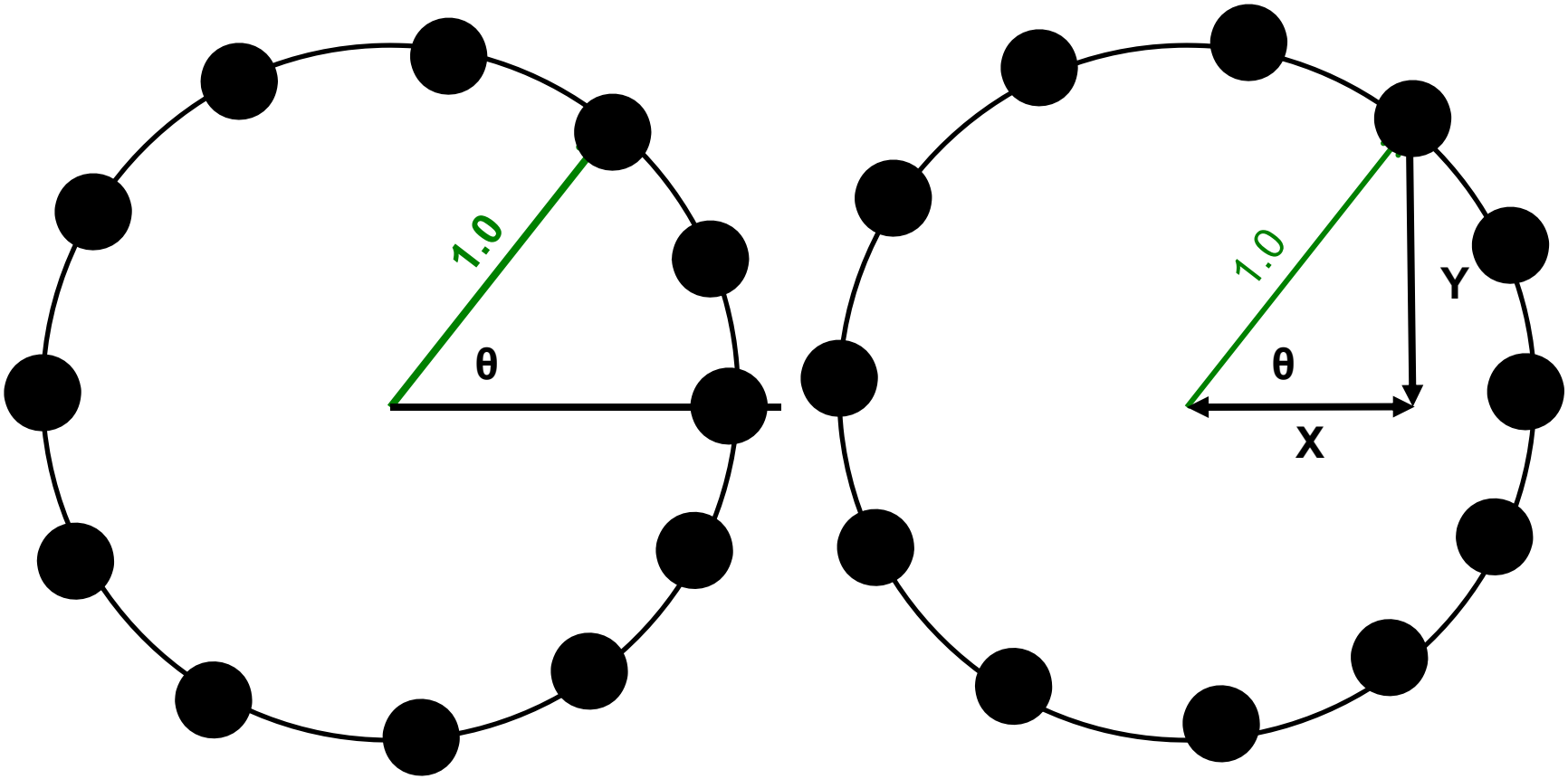
First, We Need to Understand Something about Angles



One of the things we notice is that each angle **θ** has a unique **X** and **Y** that goes with it.

These are different for each θ.

First, We Need to Understand Something about Angles



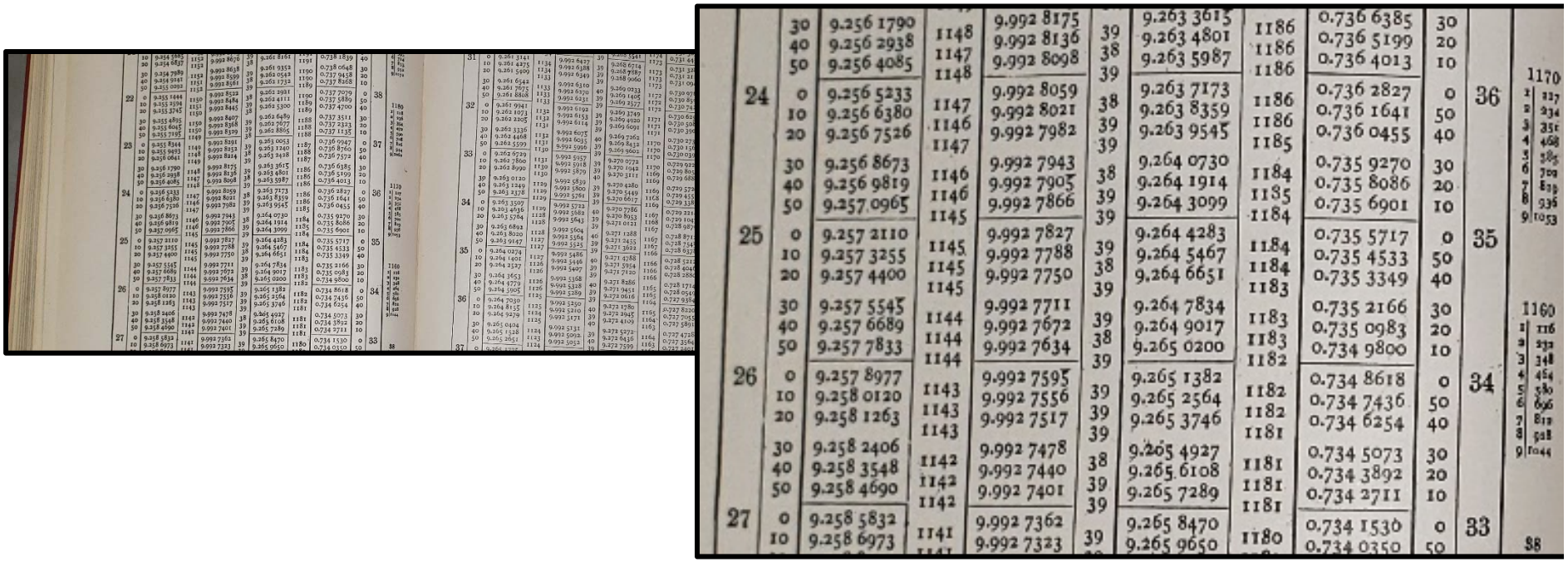
Fortunately, centuries ago, people developed tables of those X and Y values as functions of θ.

They called the X values cosines and the Y values sines. These are abbreviated cos and sin.

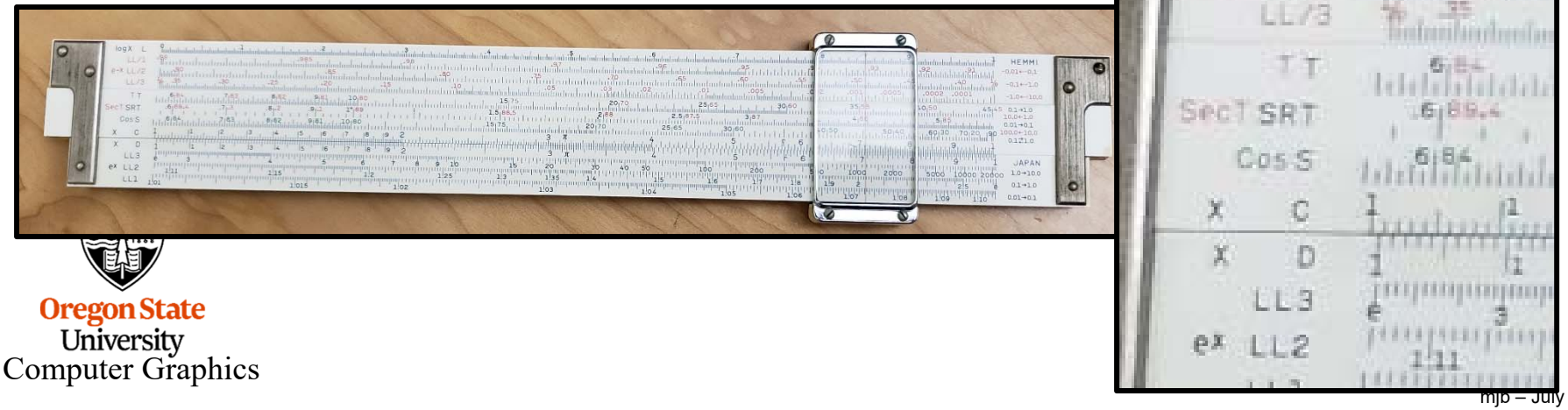
$\cos\theta = X$

$\sin\theta = Y$

How People used to Lookup Sines and Cosines – Fortunately We Now Have Calculators and Computers

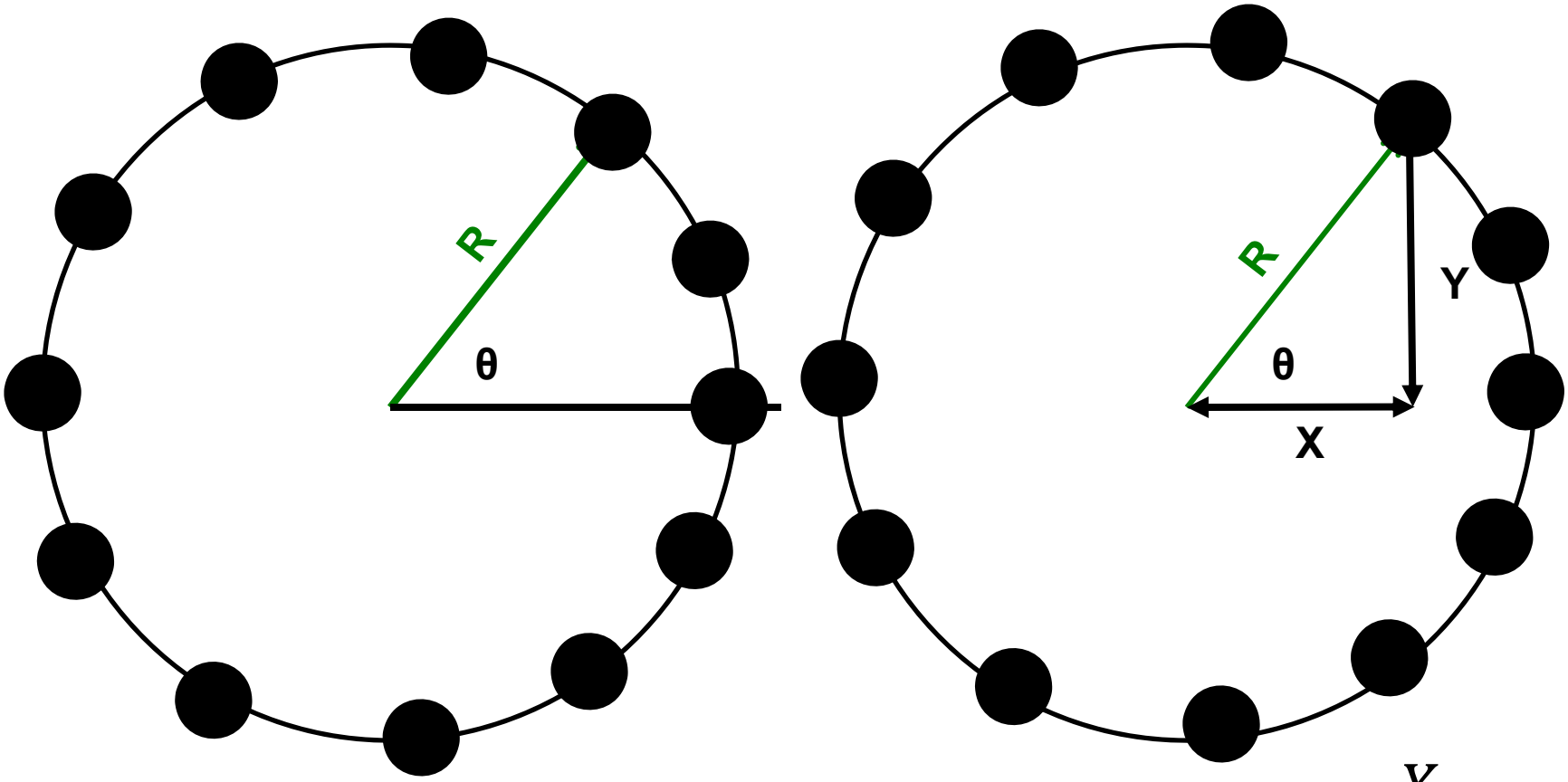


	30	9.256 1790	1148	9.992 8175	39	9.263 3613	1186	0.736 6385	30		
	40	9.256 2938	1147	9.992 8136	38	9.263 4801	1186	0.736 5199	20		
	50	9.256 4085	1148	9.992 8098	39	9.263 5987	1186	0.736 4013	10		
24	0	9.256 5233	1147	9.992 8059	38	9.263 7173	1186	0.736 2827	0	36	
	10	9.256 6380	1146	9.992 8021	39	9.263 8359	1186	0.736 1641	50		
	20	9.256 7526	1147	9.992 7982	39	9.263 9545	1185	0.736 0455	40		
	30	9.256 8673	1146	9.992 7943	38	9.264 0730	1184	0.735 9270	30		
	40	9.256 9819	1146	9.992 7905	39	9.264 1914	1185	0.735 8086	20		
	50	9.257 0965	1145	9.992 7866	39	9.264 3099	1184	0.735 6901	10		
25	0	9.257 2110	1145	9.992 7827	39	9.264 4283	1184	0.735 5717	0	35	
	10	9.257 3255	1145	9.992 7788	38	9.264 5467	1184	0.735 4533	50		
	20	9.257 4400	1145	9.992 7750	39	9.264 6651	1183	0.735 3349	40		
	30	9.257 5545	1144	9.992 7711	39	9.264 7834	1183	0.735 2166	30		
	40	9.257 6689	1144	9.992 7672	38	9.264 9017	1183	0.735 0983	20		
	50	9.257 7833	1144	9.992 7634	39	9.265 0200	1182	0.734 9800	10		
26	0	9.257 8977	1143	9.992 7595	39	9.265 1382	1182	0.734 8618	0	34	
	10	9.258 0120	1143	9.992 7556	39	9.265 2564	1182	0.734 7436	50		
	20	9.258 1263	1143	9.992 7517	39	9.265 3746	1181	0.734 6254	40		
	30	9.258 2406	1142	9.992 7478	38	9.265 4927	1181	0.734 5073	30		
	40	9.258 3548	1142	9.992 7440	39	9.265 6108	1181	0.734 3892	20		
	50	9.258 4690	1142	9.992 7401	39	9.265 7289	1181	0.734 2711	10		
27	0	9.258 5832	1141	9.992 7362	39	9.265 8470	1180	0.734 1530	0	33	
	10	9.258 6973		9.992 7323		9.265 9650		0.734 0350	50		



Oregon State University
Computer Graphics

First, We Need to Understand Something about Angles



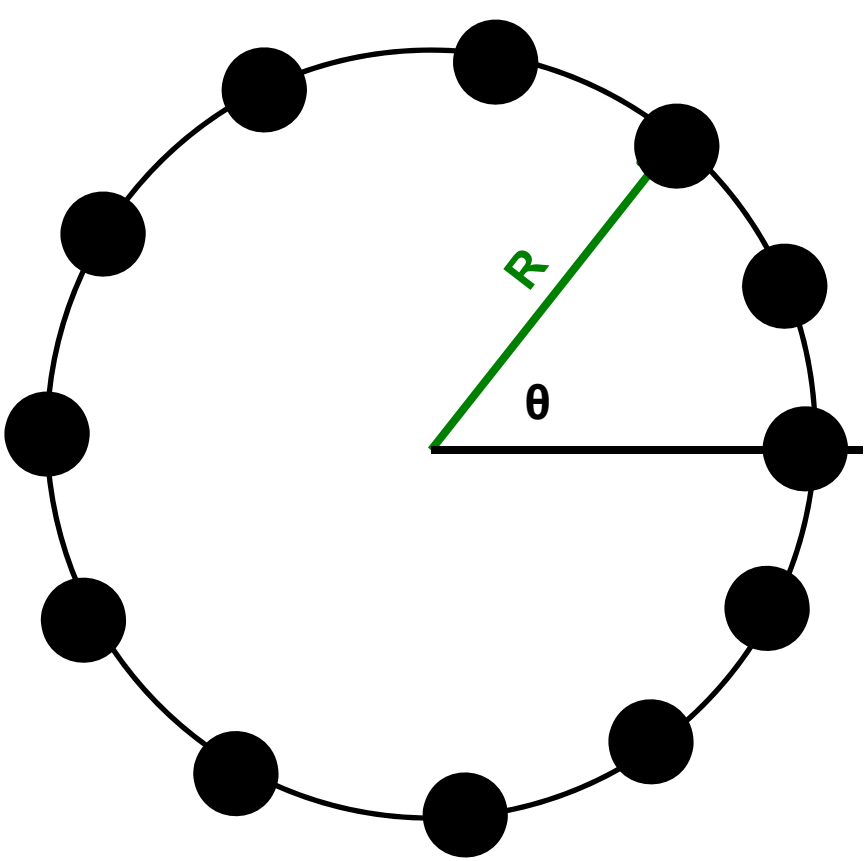
If we were to double the radius of the circle, all of the X's and Y's would also double.

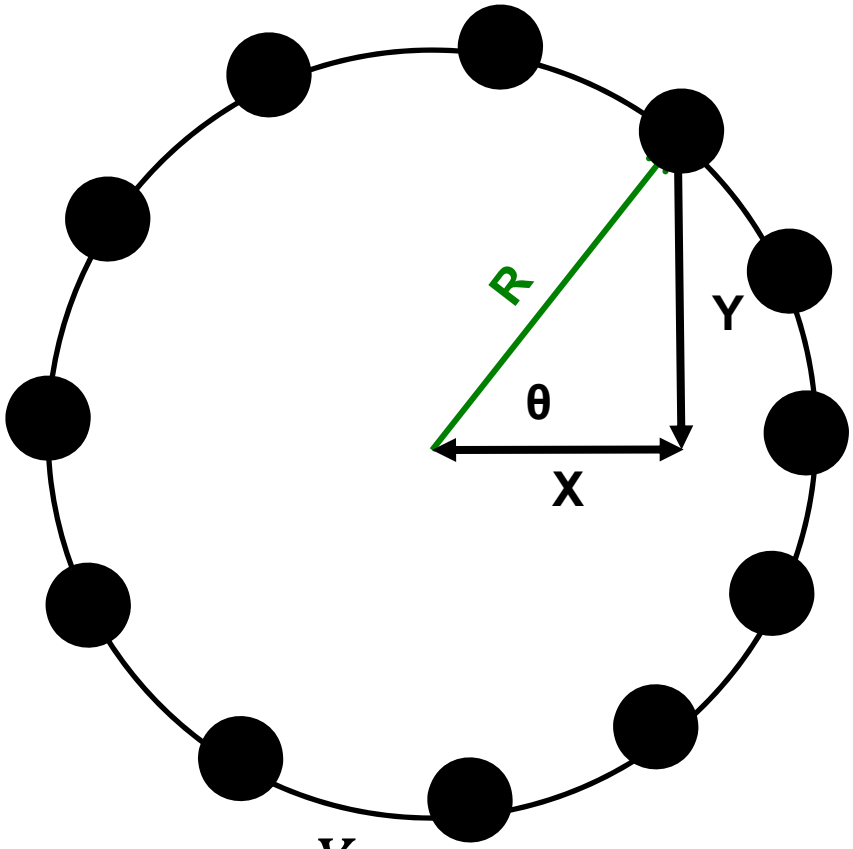
$$\cos\theta = \frac{X}{R}$$

So, really the cos and sin are *ratios* of X and Y to the circle Radius

$$\sin\theta = \frac{Y}{R}$$

First, We Need to Understand Something about Angles





So, if we know the circle Radius, and we march through a bunch of θ angles, we can determine all of the X's and Y's that we need to draw a circle.

$$\cos\theta = \frac{X}{R} \qquad \mathrm{X} = \mathrm{R} * \cos\theta$$

$$\sin\theta = \frac{Y}{R} \qquad \mathrm{Y} = \mathrm{R} * \sin\theta$$

Processing Doesn't Include a Circle-Drawing Function, So We Add Our Own

```
void
Circle( int xc, int yc, int r, int numsegs )
{
  float dang = (2.*PI) / float( numsegs );
  float ang = 0.;
  beginShape( );

  for( int i = 0; i <= numsegs; i = i + 1 )
  {
    float x = xc + r * cos(ang);
    float y = yc + r * sin(ang);
    vertex( x, y );
    ang = ang + dang;
  }

  endShape( );
}
```

numsegs is the number of line segments making up the circumference of the circle.

numsegs=20 gives a nice circle.

5 gives a pentagon.
8 gives an octagon.
4 gives you a square. Etc.

Why **2.*PI** ?

Why 2.*PI ?

```
float dang = (2.*PI) / float( numsegs );
```

We commonly meaure angles in **degrees**, but science and computers like to measure them in something else called **radians**.

There are 360° in a complete circle.
There are 2π radians in a complete circle.

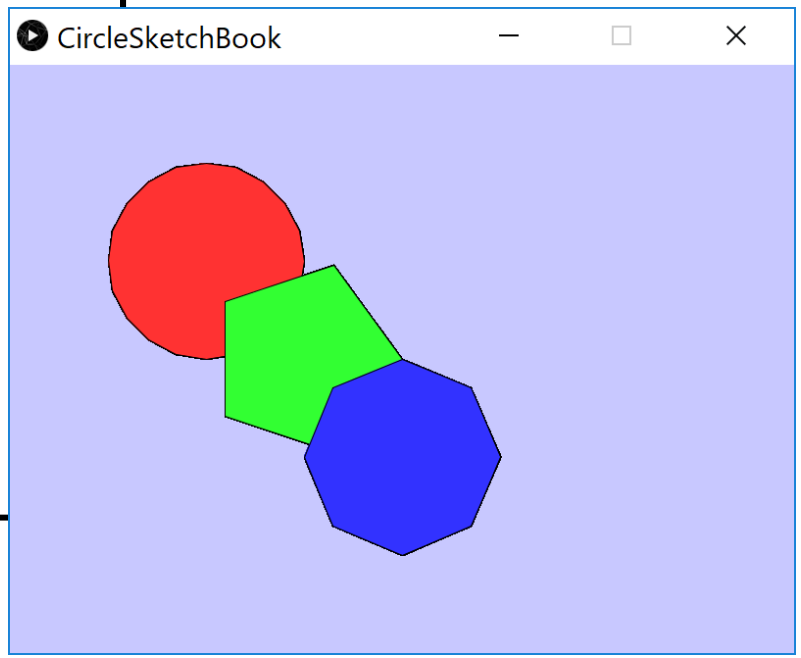
The built-in cos( ) and sin( ) functions expect angles given in radians.

Processing has build in functions to convert between the two:

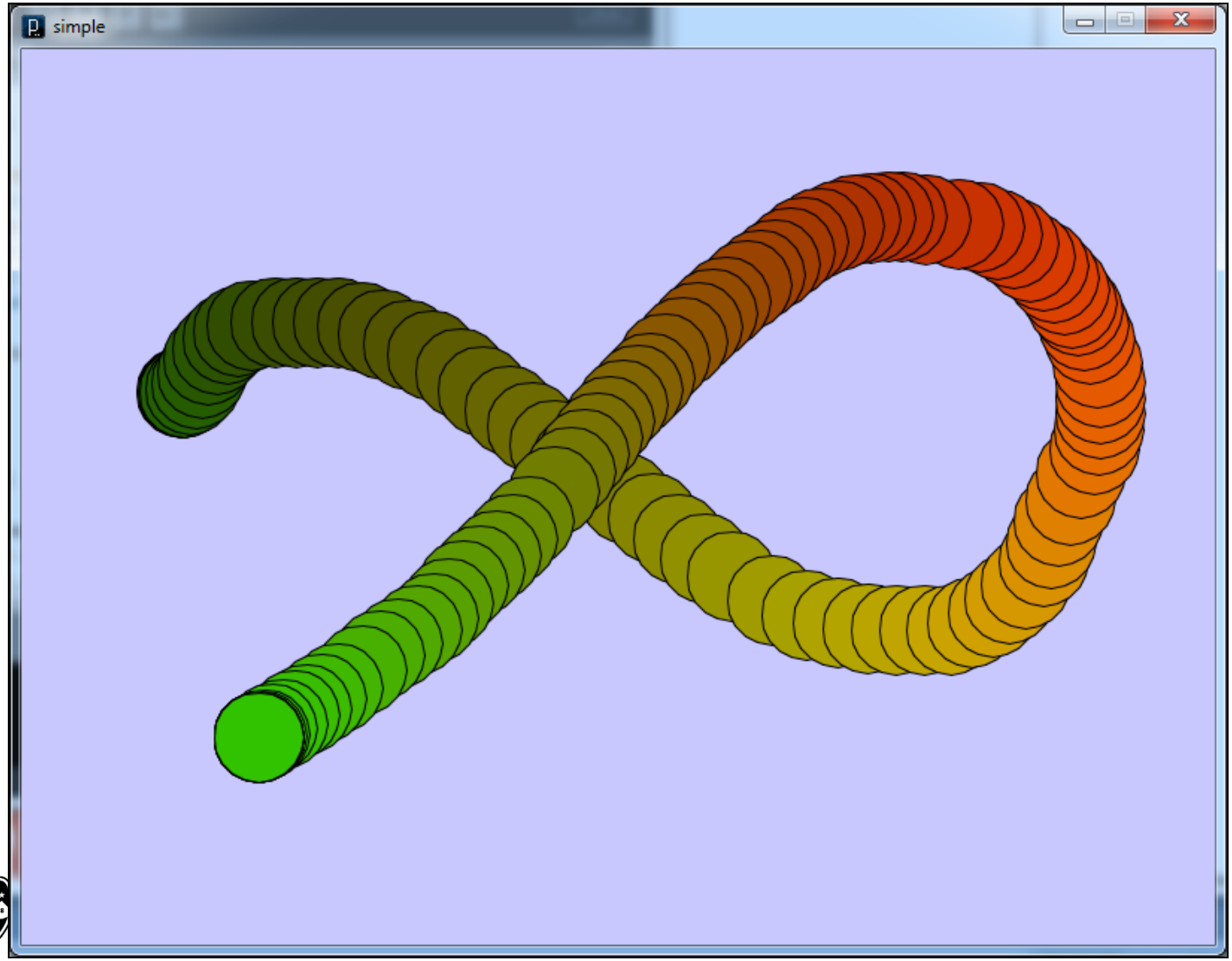
```
float rad = radians( deg );
float deg = degrees( rad );
```

Circle, Pentagon, Octagon!

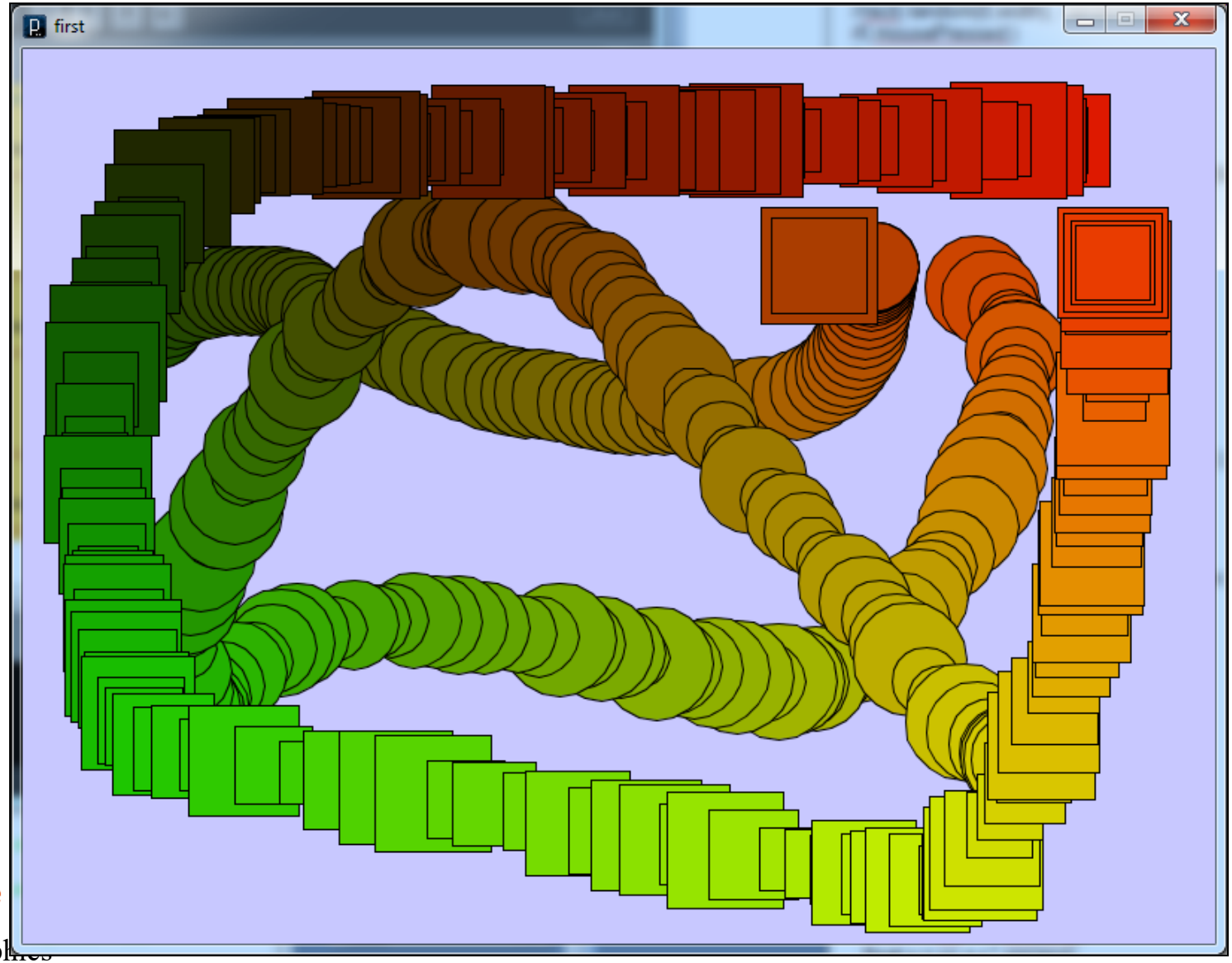
```
void
draw(  )
{
  stroke( 0, 0, 0 );

  fill( 255, 50, 50  );
  Circle( 200, 200, 100, 20 );

  fill( 50, 255, 50  );
  Circle( 300, 300, 100, 5 );

  fill( 50, 50, 255  );
  Circle( 400, 400, 100, 8 );
}
```



If We Move the Mouse, We Could Get:



Or, even:

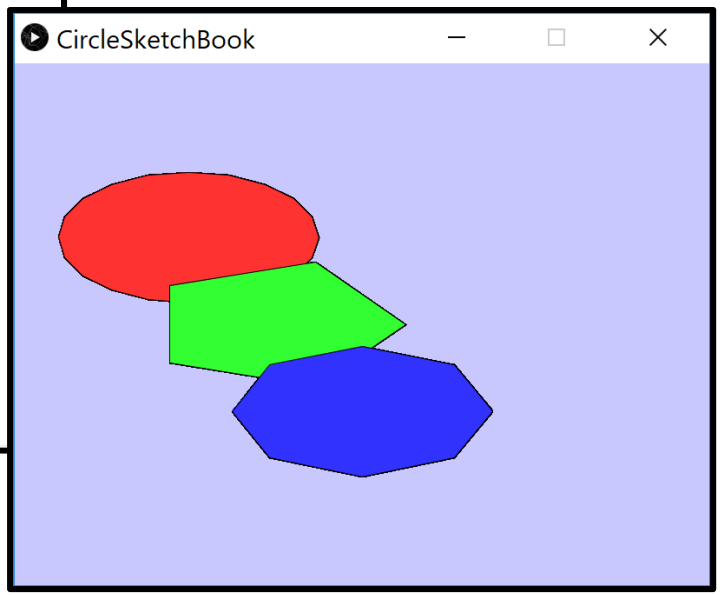


And, there is no reason the X and Y radii need to be the same…

```
void
Ellipse( int xc, int yc, int rx, int ry, int numsegs )
{
  float dang = (2.*PI) / float( numsegs );
  float ang = 0.;
  beginShape( );

  for( int i = 0; i <= numsegs; i = i + 1 )
  {
    float x = xc + rx * cos(ang);
    float y = yc + ry * sin(ang);
    vertex( x, y );
    ang = ang + dang;
  }

  endShape( );
}
```

There is actually no reason the X and Y radii need to be the same …

```
void
draw( )
{
  stroke( 0, 0, 0 );

  fill( 255, 50, 50 );
  Ellipse( 200, 200, 150, 75, 20 );

  fill( 50, 255, 50 );
  Ellipse( 300, 300, 150, 75, 5 );

  fill( 50, 50, 255 );
  Ellipse( 400, 400, 150, 75, 8 );
}
```

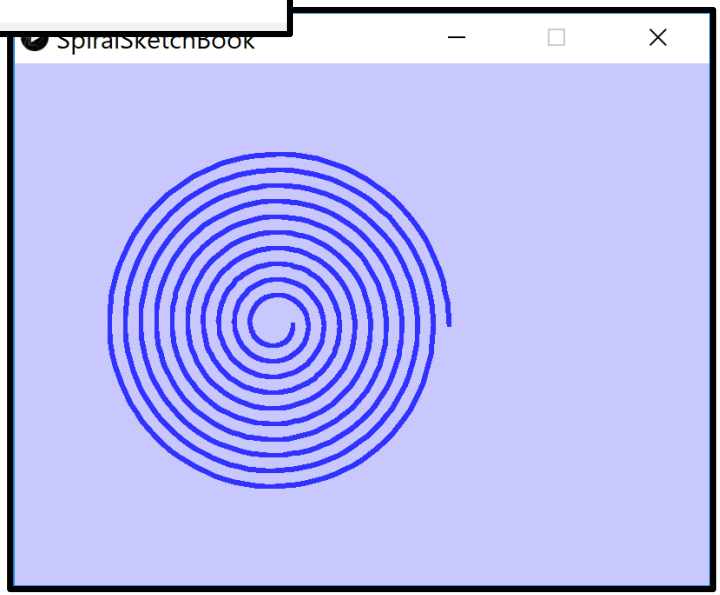


There is also no reason we can't gradually change the radius …

```
void
Spiral( int xc, int yc, int r0, int r1, int numsegs, int numturns )
{
  float dang = numturns * (2.*PI) / float( numsegs );
  float ang = 0.;
  beginShape( );

  for( int i = 0; i <= numsegs; i = i + 1 )
  {
    float newrad = map( i,  0, numsegs,  r0, r1 );
    float x = xc + newrad * cos(ang);
    float y = yc + newrad * sin(ang);
    vertex( x, y );
    ang = ang + dang;
  }

  endShape( );
}
```

There is also no reason we can't gradually change the radius …

```
void
draw(  )
{
  stroke( 50, 50, 255 );
  strokeWeight( 5 );
  noFill( );
  Spiral( 300, 300, 20, 200, 1000, 10 );
}
```



We Can Also Use This Same Idea to Arrange Things in a Circle

```
void
draw( )
{
  stroke( 0, 0, 0 );
  int numobjects = 10;
  float radius = 200.;
  int xc = 300;
  int yc = 300;
  int numsegs = 20;
  int r = 50;
  float dang = (2.*PI) / float( numobjects - 1 );
  float ang = 0.;
  for( int i = 0; i < numobjects; i = i + 1 )
  {
    float x = xc + radius * cos(ang);
    float y = yc + radius * sin(ang);
    int red   = int( map( i,   0, numobjects – 1,     0, 255 ) );
    int blue = int( map( i,   0, numobjects – 1, 255,     0 ) );
    fill( red, 0, blue );
    Circle( int(x), int(y), r, numsegs );
    ang = ang + dang;
  }
}
```

We Can Also Use This Same Idea to Arrange Things in a Circle

```
void
draw(  )
{
  stroke( 0, 0, 0 );
  int numobjects = 10;
  float radius = 200.;
  int xc = 300;
  int yc = 300;
  int numsegs = 20;
  int r = 50;
  float dang = (2.*PI) / float( numobjects - 1 );
  float ang = 0.;
  for( int i = 0; i < numobjects; i = i + 1 )
  {
    float x = xc + radius * cos(ang);
    float y = yc + radius * sin(ang);
    int red   = int( map( i,  0, numobjects-1,  0, 255 ) );
    int blue  = int( map( i,  0, numobjects-1,  255, 0 ) );
    fill( red, 0, blue );
    Circle( int(x), int(y), r, numsegs );
    ang = ang + dang;
  }
}
```