

Sines and Cosines for Animating Computer Graphics



Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu



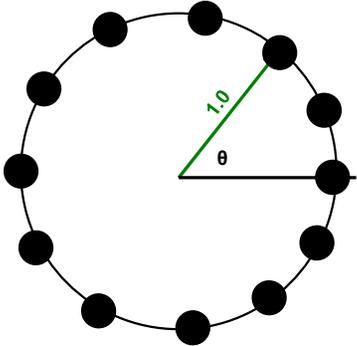
Oregon State University
Computer Graphics

SinesAndCosines.pptx mjb - September 26, 2023

1

You Know about Sines and Cosines from Math, but They are Very Useful for Animating Computer Graphics

First, We Need to Understand Something about Angles:





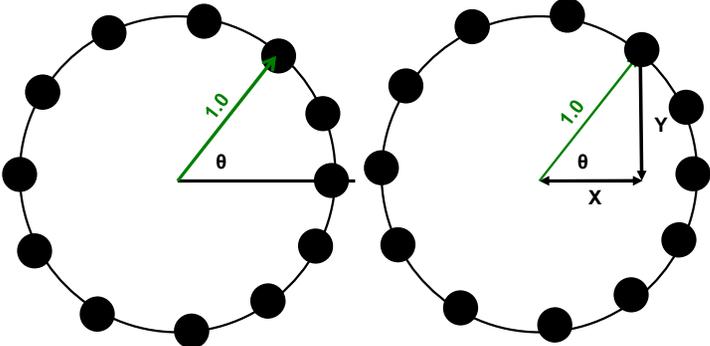
Oregon State University
Computer Graphics

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

mjb - September 26, 2023

2

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 .

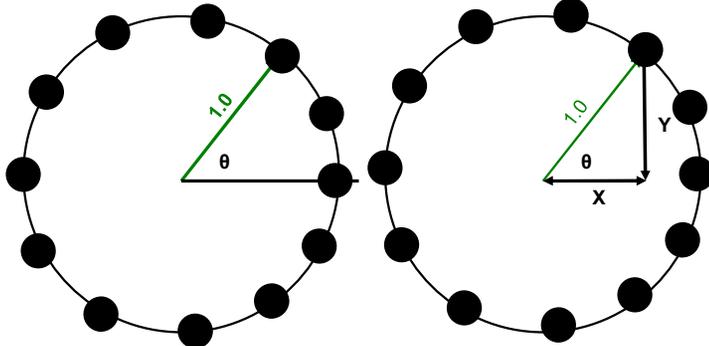


Oregon State University
Computer Graphics

mjb - September 26, 2023

3

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**.

$$\cos \theta = X$$

$$\sin \theta = Y$$



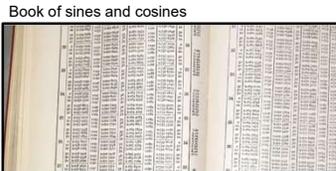
Oregon State University
Computer Graphics

mjb - September 26, 2023

4

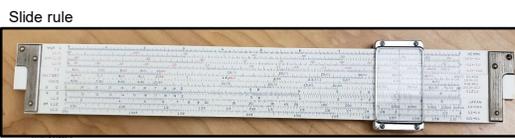
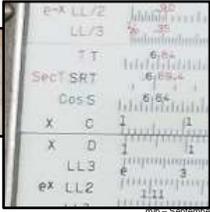
How People used to Lookup Sines and Cosines – Yuch! Fortunately We Now Have Calculators and Computers

Book of sines and cosines



24	0	0.216 1290	0.992 8175	0.263 3015	1186	0.738 0385	30	117
	10	0.216 2938	0.992 8116	0.263 4801	1186	0.736 5199	30	117
	20	0.216 4685	0.992 8058	0.263 6597	1186	0.735 0013	30	117
	30	0.216 6430	0.992 8001	0.263 8393	1186	0.733 4827	30	117
	40	0.216 8176	0.991 7944	0.264 0189	1186	0.731 9641	30	117
	50	0.216 9921	0.991 7887	0.264 1985	1186	0.730 4455	30	117
	60	0.217 1667	0.991 7830	0.264 3781	1186	0.728 9269	30	117
	70	0.217 3412	0.991 7773	0.264 5577	1186	0.727 4083	30	117
	80	0.217 5158	0.991 7716	0.264 7373	1186	0.725 8897	30	117
	90	0.217 6903	0.991 7659	0.264 9169	1186	0.724 3711	30	117
	100	0.217 8649	0.991 7602	0.265 0965	1186	0.722 8525	30	117
	110	0.218 0394	0.991 7545	0.265 2761	1186	0.721 3339	30	117
	120	0.218 2140	0.991 7488	0.265 4557	1186	0.719 8153	30	117
	130	0.218 3885	0.991 7431	0.265 6353	1186	0.718 2967	30	117
	140	0.218 5631	0.991 7374	0.265 8149	1186	0.716 7781	30	117
	150	0.218 7376	0.991 7317	0.265 9945	1186	0.715 2595	30	117
	160	0.218 9122	0.991 7260	0.266 1741	1186	0.713 7409	30	117
	170	0.219 0867	0.991 7203	0.266 3537	1186	0.712 2223	30	117
	180	0.219 2613	0.991 7146	0.266 5333	1186	0.710 7037	30	117
	190	0.219 4358	0.991 7089	0.266 7129	1186	0.709 1851	30	117
	200	0.219 6104	0.991 7032	0.266 8925	1186	0.707 6665	30	117
	210	0.219 7849	0.991 6975	0.267 0721	1186	0.706 1479	30	117
	220	0.219 9595	0.991 6918	0.267 2517	1186	0.704 6293	30	117
	230	0.220 1340	0.991 6861	0.267 4313	1186	0.703 1107	30	117
	240	0.220 3086	0.991 6804	0.267 6109	1186	0.701 5921	30	117
	250	0.220 4831	0.991 6747	0.267 7905	1186	0.700 0735	30	117
	260	0.220 6577	0.991 6690	0.267 9701	1186	0.698 5549	30	117
	270	0.220 8322	0.991 6633	0.268 1497	1186	0.697 0363	30	117
	280	0.221 0068	0.991 6576	0.268 3293	1186	0.695 5177	30	117
	290	0.221 1813	0.991 6519	0.268 5089	1186	0.693 9991	30	117
	300	0.221 3559	0.991 6462	0.268 6885	1186	0.692 4805	30	117

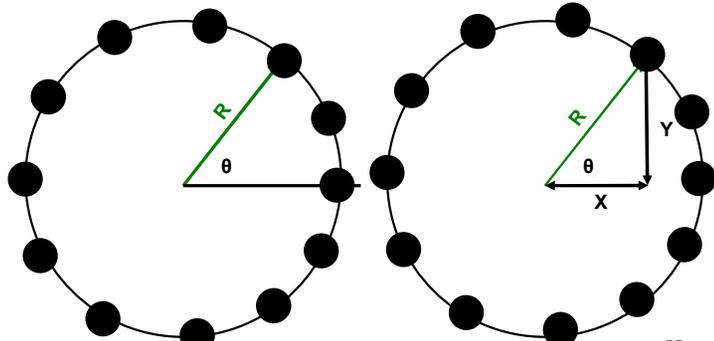
Slide rule

Oregon State University Computer Graphics
http - September 26, 2023

5

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.

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

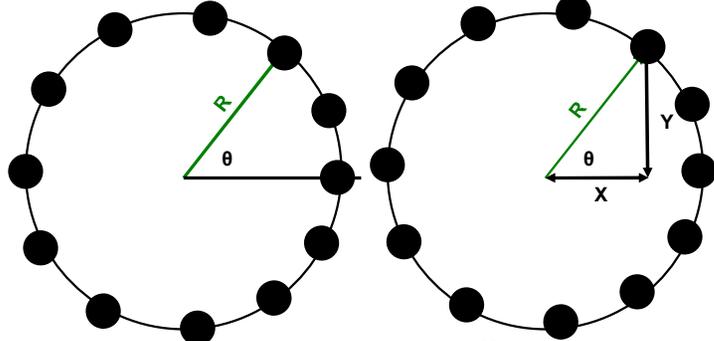
So, really the cos and sin are ratios of X and Y to the circle Radius

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

Oregon State University Computer Graphics
http - September 26, 2023

6

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.

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

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

$X = R * \cos \theta$
 $Y = R * \sin \theta$

Draw to this point

Oregon State University Computer Graphics
http - September 26, 2023

7

Thus, We Could Create Our Very Own Circle-Drawing Function

Circle center Circle radius

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.

```

void
Circle(float xc, float yc, float r, int numsegs)
{
    float dang = 2.f * F_PI / (float)numsegs;
    float ang = 0.;
    glBegin( GL_TRIANGLE_FAN );
    glVertex3f( xc, yc, 0. );

    for( int i = 0; i <= numsegs; i++ )
    {
        float x = xc + r * cosf(ang);
        float y = yc + r * sinf(ang);
        glVertex3f( x, y, 0. );
        ang += dang;
    }

    glEnd();
}
    
```

2π is how many radians are in a full circle

The C/C++ $\sin()$ and $\cos()$ functions use double-precision floating point.
The C/C++ $\sinf()$ and $\cosf()$ functions use single-precision floating point, and are faster.

Oregon State University Computer Graphics
http - September 26, 2023

8

Why 2.*PI ?

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

We humans commonly measure 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 cosf() and sinf() functions expect angles to be given in **radians**.

To convert between the two:

```
float rad = deg * ( F_PI/180.f);
float deg = rad * ( 180.f/F_PI);
```

glRotatef() and gluPerspective() are the only two programming functions I can think of that use degrees. All others use radians!



mjb - September 26, 2023

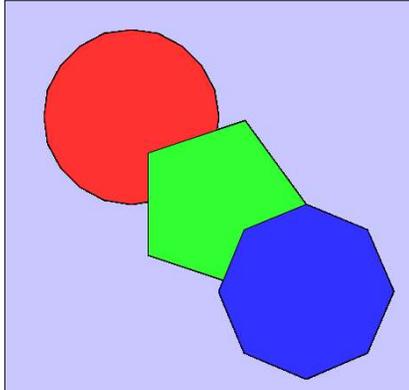
9

Circles and Pentagons and Octagons, Oh My!

```
glColor3f( 1., 0., 0. );
Circle( 1.f, 3.f, 1.f, 20 );

glColor3f( 0., 1., 0. );
Circle( 2.f, 2.f, 1.f, 5 );

glColor3f( 0., 0., 1. );
Circle( 3.f, 1.f, 1.f, 8 );
```




mjb - September 26, 2023

10

Easy as π: M_PI vs. F_PI

The math.h include file has a definition of π that looks like this:

```
#define M_PI      3.14159265358979323846
```

Which will work just fine for whatever you need it for.

But, Visual Studio goes a little crazy complaining about mixing doubles (which is what M_PI is in) and floats (which is probably what you use most often). So, your sample code has these lines in it:

```
#define F_PI      ((float) (M_PI))
#define F_2_PI   ((float) (2.*F_PI))
#define F_PI_2   ((float) (F_PI/2.*F))
```

I use the *F_* version a lot because it keeps VS quiet. You can use either.



mjb - September 26, 2023

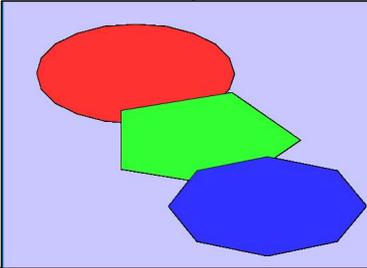
11

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

```
void
Ellipse( float xc, float yc, float rx, float ry, int numsegs )
{
  float dang = 2.f * F_PI / (float)numsegs;
  float ang = 0.;
  glBegin( GL_TRIANGLE_FAN );
  glVertex3f( xc, yc, 0. );

  for( int i = 0; i <= numsegs; i++ )
  {
    float x = xc + rx * cosf(ang);
    float y = yc + ry * sinf(ang);
    glVertex3f( x, y, 0. );
    ang += dang;
  }

  glEnd( );
}
```




mjb - September 26, 2023

12

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

```

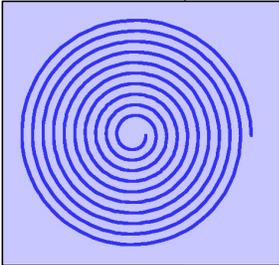
void
Spiral( float xc, float yc, float r0, float r1, int numsegs, int numturns )
{
    float dang = (float)numturns * 2.f * F_PI / (float)numsegs;
    float ang = 0.;
    glBegin( GL_LINE_STRIP );

    for( int i = 0; i <= numsegs; i++ )
    {
        float t = (float)i / (float)numsegs; // 0.-1.
        float newrad = (1.-t)*r0 + t*r1;
        // linearly interpolate from r0 to r1

        float x = xc + newrad * cosf(ang);
        float y = yc + newrad * sinf(ang);
        glVertex3f( x, y, 0. );
        ang += dang;
    }

    glEnd( );
}

```



Oregon State University Computer Graphics
mjb - September 26, 2023

13

Parametric Linear Interpolation (Blending)

What's this code all about?

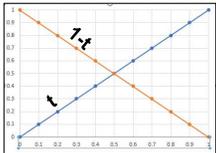
```

float t = (float)i / (float)numsegs; // 0.-1.
float newrad = (1.-t)*r0 + t*r1;

```

In computer graphics, we do a lot of linear interpolation between two input values. Here is a good way to do that:

1. Setup a float variable, t , such that it ranges from 0. to 1. The line `float t = (float) i / (float) numsegs;` does this.
2. Step through as many t values as you want interpolation steps. The line `for(int i = 0; i <= numsegs; i++)` does this.
3. For each t , multiply one input value by $(1.-t)$ and multiply the other input value by t and add them together. The line `float newrad = (1.-t)*r0 + t*r1;` does this.




Oregon State University Computer Graphics
mjb - September 26, 2023

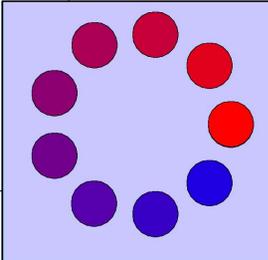
14

We Can Also Use This Same Idea to Arrange Things in a Circle and Linearly Blend Their Colors

```

int numObjects = 9;
float radius = 2.f;
float xc = 3.f;
float yc = 3.f;
int numSegs = 20;
float r = 50.f;
float dang = 2.*F_PI / (float) ( numObjects - 1 );
float ang = 0.;
for( int i = 0; i < numObjects; i++ )
{
    float x = xc + radius * cosf(ang);
    float y = yc + radius * sinf(ang);
    float t = (float)i / (float)(numObjects-1); // 0.-1.
    float red = t; // ramp up
    float blue = 1.f - t; // ramp down
    glColor3f red, 0., blue );
    Circle( x, y, r, numSegs );
    ang += dang;
}

```



Oregon State University Computer Graphics
mjb - September 26, 2023

15

By Understanding what the Sine Function Looks Like, We Can Also Use it to Control Animations Based on Time

In your sample.cpp file, we have some code that looks like this:

```

float Time; // global variable intended to lie between [0.,1.)
...
const int MS_PER_CYCLE = 10000; // 10000 milliseconds = 10 seconds
...
// in Animate( ):
int ms = glutGet(GLUT_ELAPSED_TIME);
ms %= MS_PER_CYCLE;
// makes the value of ms between 0 and MS_PER_CYCLE-1
Time = (float)ms / (float)MS_PER_CYCLE;
// makes the value of Time between 0. and slightly less than 1.

```

Oregon State University Computer Graphics
mjb - September 26, 2023

16

17

By Understanding what the Sine Function Looks Like, We Can Also Use it to Control Animations Based on Time

The sine function goes from -1. to +1., and does it very smoothly

$y = \sin(2. * \pi * Time)$

Oregon State University Computer Graphics

mjb - September 26, 2023

17

18

By Understanding what the Sine Function Looks Like, We Can Also Use it to Control Animations Based on Time

Sine functions produce a smoother set of motions than linear functions do (that's why we use them):

Sine function Linear function

Linear function tries to produce infinite acceleration at these two locations

Oregon State University Computer Graphics

mjb - September 26, 2023

18

19

Increasing the Amplitude, Increasing the Frequency

$\sin(2. * \pi * Time)$ $2. * \sin(2. * \pi * Time)$ $\sin(2. * (2. * \pi * Time))$

Oregon State University Computer Graphics

mjb - September 26, 2023

19

20

Increasing the Amplitude, Increasing the Frequency

$\sin(2. * \pi * Time)$ $2. * \sin(2. * \pi * Time)$ $\sin(2. * (2. * \pi * Time))$

$A * \sin(F * (2. * \pi * Time))$

Changing this number changes the Amplitude Changing this number changes the Frequency

Oregon State University Computer Graphics

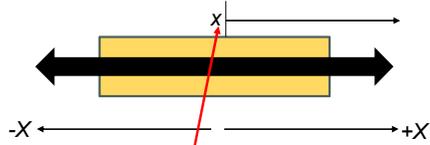
mjb - September 26, 2023

20

Oscillating Motion

21

Let's say you want a block to oscillate back and forth in x:



This code would cause it to do that:

```
// in Display():  
float x = X*sin(F*(2.* pi * Time) )  
...  
glTranslatef( x, 0., 0. );  
glCallList( BlockList );
```



Oregon State
University
Computer Graphics

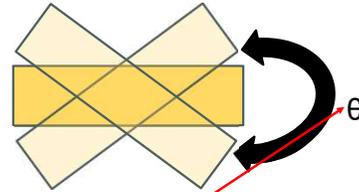
mjb - September 26, 2023

21

Rocking Motion

22

Let's say you want a block to rock back and forth:



This code would cause it to do that:

```
// in Display():  
float theta = 45.f * sin(F*(2.* pi * Time) )  
...  
glRotatef( theta, 0., 0., 1. );  
glCallList( BlockList );
```



Oregon State
University
Computer Graphics



mjb - September 26, 2023

22