

Sines and Cosines for Animating Computer Graphics



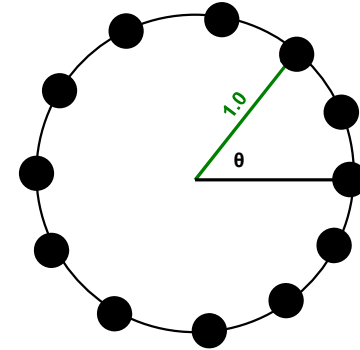
SinesAndCosines.pptx

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:

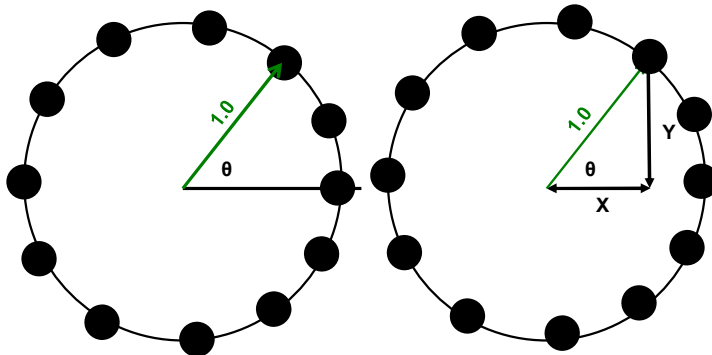


If a circle has a radius of 1.0, then we can march around it by simply changing the angle that we call θ .

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.

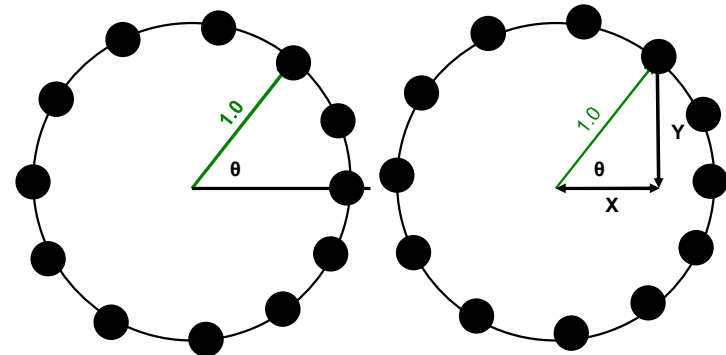
These are different for each θ .



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.

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

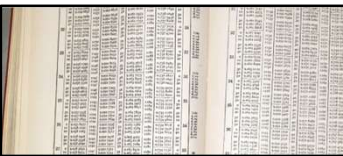


mjb - September 26, 2023

4

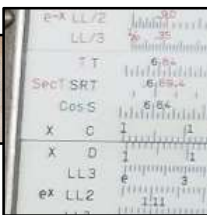
How People used to Lookup Sines and Cosines – Yuch! Fortunately We Now Have Calculators and Computers

Book of sines and cosines



24	0	0.316 1790	0.992 8175	0.303 3015	1186	0.738 6385	30	117
40	0.316 2918	1147	0.992 8116	0.303 4801	1186	0.738 5199	20	117
50	0.316 4045	1148	0.992 8058	0.303 5977	1186	0.738 4013	10	117
10	0.316 5172	1147	0.992 7999	0.303 7153	1186	0.738 2827	0	36
20	0.316 6300	1147	0.992 8001	0.303 8319	1186	0.738 1641	50	36
30	0.316 7426	1147	0.991 7982	0.303 9545	1185	0.738 0455	40	36
40	0.316 8553	1146	0.992 7943	0.304 0730	1184	0.738 0270	30	36
50	0.316 9679	1146	0.992 7905	0.304 1914	1185	0.738 0086	20	36
0	0.317 0805	1145	0.992 7866	0.304 3099	1184	0.738 0001	10	36
10	0.317 1931	1145	0.992 7827	0.304 4283	1184	0.738 0017	0	35
20	0.317 3057	1145	0.992 7788	0.304 5467	1184	0.738 0033	50	35
30	0.317 4183	1145	0.992 7750	0.304 6651	1184	0.738 0049	40	35
40	0.317 5309	1144	0.992 7711	0.304 7834	1183	0.738 0065	30	35
50	0.317 6435	1144	0.992 7672	0.304 9017	1183	0.738 0081	20	35
0	0.317 7561	1144	0.992 7634	0.305 0200	1182	0.738 0097	10	35
10	0.317 8687	1143	0.992 7595	0.305 1382	1182	0.738 0113	0	34
20	0.317 9813	1143	0.992 7556	0.305 2564	1182	0.738 0129	50	34
30	0.318 0939	1143	0.992 7517	0.305 3746	1181	0.738 0145	40	34
40	0.318 2065	1142	0.992 7478	0.305 4927	1181	0.738 0161	30	34
50	0.318 3191	1142	0.992 7439	0.305 6108	1181	0.738 0177	20	34
0	0.318 4317	1142	0.992 7401	0.305 7289	1181	0.738 0193	10	34
10	0.318 5443	1141	0.992 7362	0.305 8470	1180	0.738 0209	0	33
20	0.318 6569	1141	0.992 7323	0.305 9650	1180	0.738 0225	50	33

Slide rule

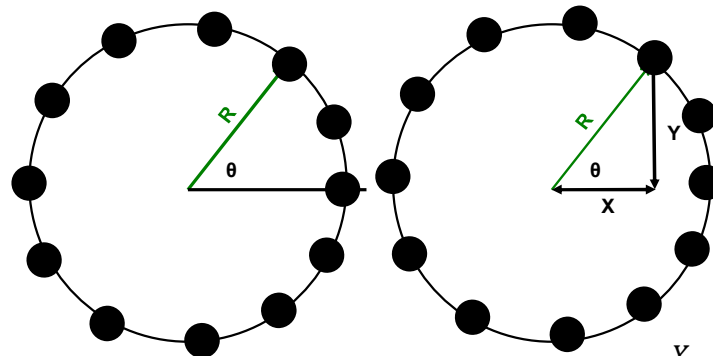


Oregon State
University
Computer Graphics

mjb - 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.

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

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

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

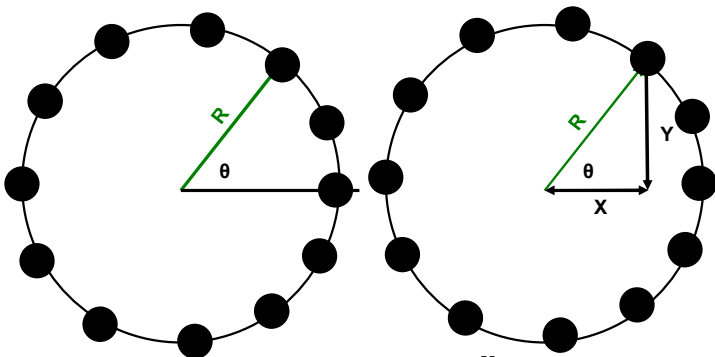


Oregon State
University
Computer Graphics

mjb - 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

mjb - 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.

2 π 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.

```

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();
}
    
```

GL_TRIANGLE_FAN

V₁ V₂ V₃ V₄ V₅

Oregon State
University
Computer Graphics

mjb - 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);
```



Oregon State
University
Computer Graphics

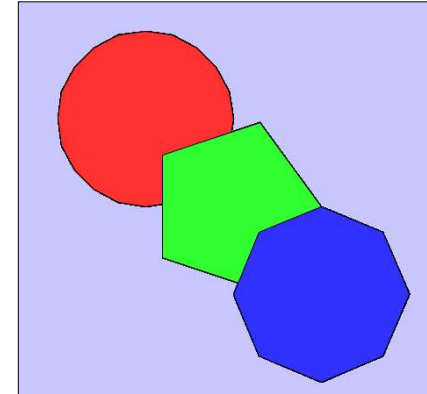
`glRotatef()` and `gluPerspective()` are the only two programming functions I can think of that use degrees. All others use radians!

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 );
```



Oregon State
University
Computer Graphics

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.



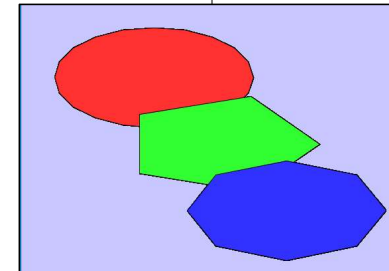
Oregon State
University
Computer Graphics

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_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();  
}
```



Oregon State
University
Computer Graphics

mjb - September 26, 2023

12

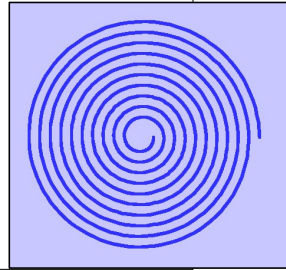
There is also no reason we can't gradually change the radius ...

```
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( );
}
```



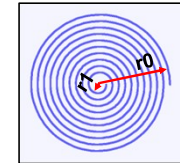
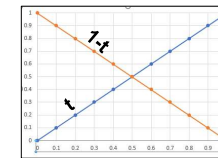
Parametric Linear Interpolation (Blending)

What's this code all about?

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

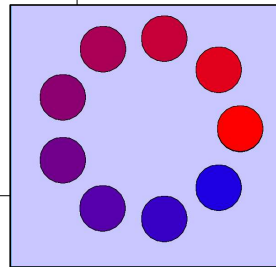
In computer graphics, we do a lot of linear interpolation between two input values. Here is a good way to do that:

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.



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;
}
```



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

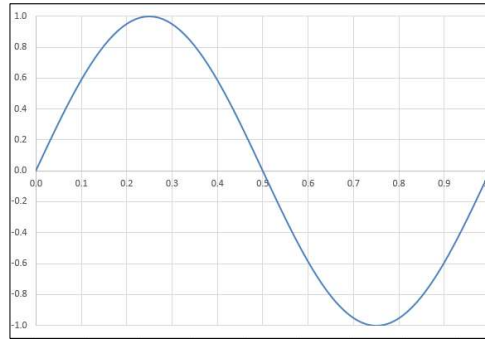
...

// 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.
```

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

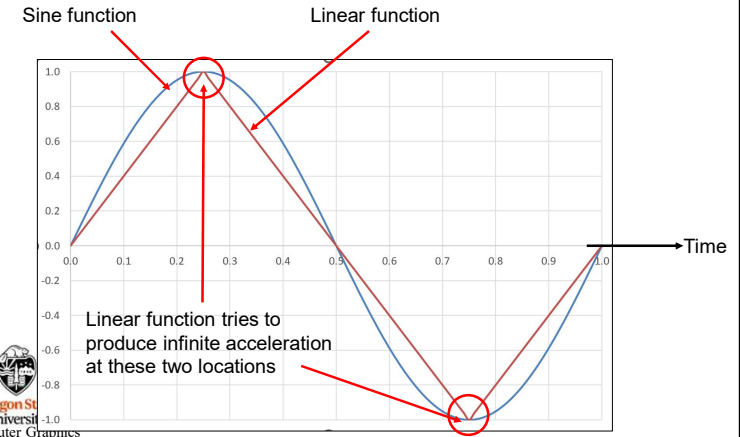
The sine function goes from -1. to +1., and does it very smoothly

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



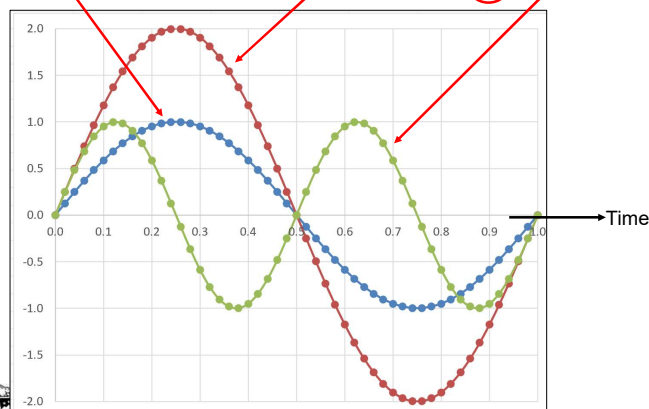
**By Understanding what the Sine Function Looks Like,
We Can Also Use it to Control Animations Based on Time**

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



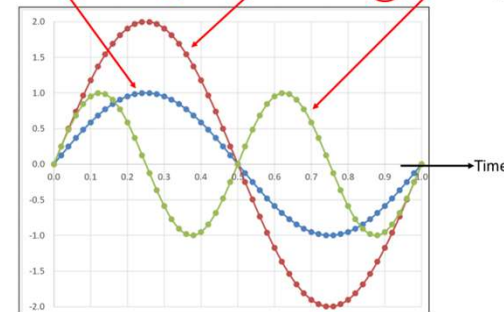
Increasing the Amplitude, Increasing the Frequency

$\sin(2. * \pi * \text{Time})$ $2. * \sin(2. * \pi * \text{Time})$ $\sin(2. * (2. * \pi * \text{Time}))$



Increasing the Amplitude, Increasing the Frequency

$\sin(2. * \pi * \text{Time})$ $2. * \sin(2. * \pi * \text{Time})$ $\sin(2. * (2. * \pi * \text{Time}))$



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

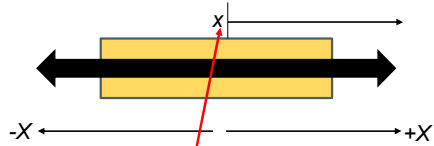
Changing this number
changes the Amplitude

Changing this number
changes the Frequency

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

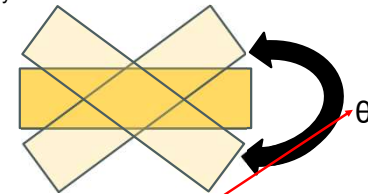
mjb - September 26, 2023

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