2-19-2000

TrigMath Behavior 1.0 Event id's: 200000 - 200007

By Matthew Peterson

matthew@pinoko.berkeley.edu

Introduction

TrigMath is a set of Math functions (implemented as custom events) for calculating trig and inverse-trig functions in LiveStage Pro. It also has a Square Root function, and functions for converting angles between Radians and Degrees, and placing an angle between 0 and 360 (modulus function). Here are the functions:

1) **Trig** -- Computes Sine, Cosine and Tangent (angles in degrees)

2) **ArcTan** -- The arcTangent (returns angles in degrees).
Also known as ATAN or $\tan^{-1}$

3) **Polar2Cart** -- Conversion from polar to Cartesian coordinates

4) **Cart2Polar** -- Conversion from Cartesian to polar coordinates

5) **Deg2Rad** -- Angle conversion from degrees to radians

6) **Rad2Deg** -- Angle conversion from radians to degrees

7) **Sqrt** -- The square root function.

8) **Modulus** -- Mod function. Cyclical arithmetic.
For example -90 MOD 360 = 270

Purpose

The Quicktime API, and thus LiveStage, lacks a set of math functions extremely important for moving sprites around – trig and inverse trig functions. These functions are an attempt to fill this math gap. Very briefly, I'll explain the need for these functions:

1) Trig functions have an almost unlimited use. For instance, rotating a sprite around an arbitrary point, performing 3D perspective operations, orbital and other natural motion. It is used to convert from polar to Cartesian coordinates (see Cart2Polar function). Both coordinate systems have their

easy and difficult points. It is very useful to be able to convert between these two systems.

2) Inverse trig functions, in particular the inverse tangent (arcTan) are useful when wanting to know how much to rotate a sprite so that it points to an x,y location. This has applications in drawing lines, rotating knobs, and determining how much a sprite is rotated. It is used to convert from polar to Cartesian coordinates (see Polar2Cart).

3) The square root function is not a trig function, but is related in that it is often used along with the arcTan function to determine the distance between two points (using Pythagorean Theorem). It has other uses, such as determining how much to scale a sprite to make it x times as large.

4) LiveStage (i.e. QuickTime API) lacks a MOD function. The closest it has is a REM function. For positive numbers, the MOD and REM functions are the same, but the differ for negative numbers.

For example, -90 REM 360 = -90
and -90 MOD 360 = 270

When working with angles, the MOD function is what is needed. Using the REM function instead will give erroneous results.

Quick Reference

To use the following functions, place this behavior in a blank sprite. By "blank" I mean a sprite not used for other purposes. The reason is because sprite rotation is used to in many of these calculations. It doesn't mater what image the spite has, but it is best if it is a small non-vector graphic. After placing this behavior on the sprite, the sprite will become invisible and off the track on frameLoad.

• **Trig** : Set the input TrigAngle, and call this event.:

[input variable]
1) TrigAngle: input angle in degrees.

[Output variables]
1) Sine: width divided by the hypotnus
2) Cosine: height divided by the hypotnus
3) Tangent: height divided by the width

Example (Calculate the sine of 30 degrees)

```
GlobalVars TrigAngle Sine

TrigAngle = 30
```

```
spriteofid(1).executeevent($Trig)
//Assuming this behavior is in spriteofid(1)
debugstr(sine) //should equal 0.5
```

Note: The tangent of 90 degrees is infinity. Since Qscript doesn't do a good job with infinity, or very large numbers, I set this value to 999999.

- **ArcTan** : the inverse tangent. Supply the y/x ratio in the TrigInput Global variable:
  [input variables]
  1) TrigInput: The input to ArcTan(TrigInput)

  [output variables]
  1) TrigOutput: The angle in degrees.

  Example (what is the arc tangent of 0.3)

```
GlobalVars TrigInput  TrigOutput

TrigInput = 0.3
spriteofid(1).executeevent($ArcTan)
//Assuming this behavior is in spriteofid(1)
debugstr(TrigOutput) //should approximately equal 16.7
```

  Note: For most functions, Cart2Polar should be used.

- **Polar2Cart**: Convert from Polar to Cartesian coordinates.
  [input variables]
  1) TrigAngle: angle in degrees
  2) TrigDistance: Distance in pixels

  [output variables]
  1) Trig1X: The x distance in pixels
  2) Trig1Y: The y distance in pixels

  Example (Move a sprite so it is 50 pixels and 30 degrees from the top-left corner of the sprite track)

```
GlobalVars TrigAngle TrigDistance Trig1X  Trig1Y

TrigDistance = 50
TrigAngle = 30
spriteofid(1).executeevent($Polar2Cart)
//Assuming this behavior is in spriteofid(1)
spriteofid(2).moveto(Trig1X, Trig1Y)
```

• **Cart2Polar**: Cartesian to Polar coodinates

[input variables]
1) Trig1X: The x position of the start point
2) Trig1Y: The y position of the start point
3) Trig2X: The x position of the end point
4) Trig2Y: The y position of the end point

[output variables]
1) TrigAngle: angle in degrees
2) TrigDistance: Distance in pixels

Example (Rotate a sprite so that it points to (50, 30))

```
GlobalVars TrigAngle TrigDistance Trig1X  Trig1Y Trig2X  Trig2Y
//The start point
Trig1X = spriteofid(2).boundsleft
Trig1Y = spriteofid(2).boundsTop
//The end point
Trig2X = 50 //x distance
Trig2Y = 30 //y distance

spriteofid(1).executeevent($Cart2Polar)
//Assuming this behavior is in spriteofid(1)
spriteofid(2).rotate(TrigAngle)
```

• **Deg2Rad**: Many equations use Radians. The trig function in this behavior use degrees, so it is often necessary to convert.

[Input variables]
1) TrigInput: Angle in degrees

[Output variables]
1) TrigOutput: Angle in Radians

Example (convert 90 degrees to radians)

```
GlobalVars TrigInput  TrigOutput

TrigInput = 90
spriteofid(1).executeevent($Deg2Rad)
//Assuming this behavior is in spriteofid(1)
debugstr(TrigOutput) //should equal 1.57
```

• **Deg2Rad**: Many equations use Radians. The trig function in this behavior use degrees, so it is often necessary to convert.

[Input variables]
1) TrigInput: Angle in Radians

[Output variables]
1) TrigOutput: Angle in Degrees

Example (convert pi/2 radians to degrees)

```
GlobalVars TrigInput  TrigOutput

TrigInput = pi/2
spriteofid(1).executeevent($Rad2Deg)
//Assuming this behavior is in spriteofid(1)
debugstr(TrigOutput) //should equal 90
```

- **Sqrt**: Square root function
  [Input variables]
  1) TrigInput: input to the square root function.

  [Output variables]
  1) TrigOutput: = sqrt(TrigInput)

  Example (calculate the squareroot of 25)

```
GlobalVars TrigInput  TrigOutput

TrigInput = 25
spriteofid(1).executeevent($Sqrt)
//Assuming this behavior is in spriteofid(1)
debugstr(TrigOutput) //should equal 5
```

- **Modulus**: TrigOutput = TrigInput MOD TrigMod
  [Input variables]
  1) TrigInput: value
  1) TrigMod: the modulus

  [Output variables]
  1) TrigOutput: result

  Example (what is -90 MOD 360)

```
GlobalVars TrigInput  TrigOutput TrigMod

TrigInput = -90
TrigMod = 360
spriteofid(1).executeevent($Modulus)
//Assuming this behavior is in spriteofid(1)
```

```
debugstr(TrigOutput) //should equal 270
```

Note: TrigMod is set to 360 in the Frame load, but you can change it to whatever value you want. This function differs from the REM function included in LiveStage.

Reserved Variable Names for this Behavior:

Input/Output globalvars:

TrigAngle, TrigDistance, Sine, Cosine, Tangent, TrigInput, TrigOutput, Trig1X, Trig1Y, Trig2X, Trig2Y, TrigMod

Note: Internal variables all start with "MP_" and thus should not overlap with other variables.

Technical Notes:

I will briefly explain the general principles behind the functions in this behavior. For a more in depth understanding, look at the comments within the behavior itself.

Although QuickTime doesn't supply these trig functions for use in Qscript, QuickTime is having to use these functions in order to rotate sprites correctly. We can take advantage of this fact by rotating sprites and using resulting point locations to calculate our trig functions. This is much faster, and much more accurate than using a table-lookup method, or a taylor series.

Revision History:

2-19-2000 Version 1.0 first written.
These functions have been posted by me to the LiveStage email list many times over the last year. They were compiled into a Math Behavior early January, 2000. The functions supplied here are more refined, and the event ID's now match my Registered ID range (200000-200999).