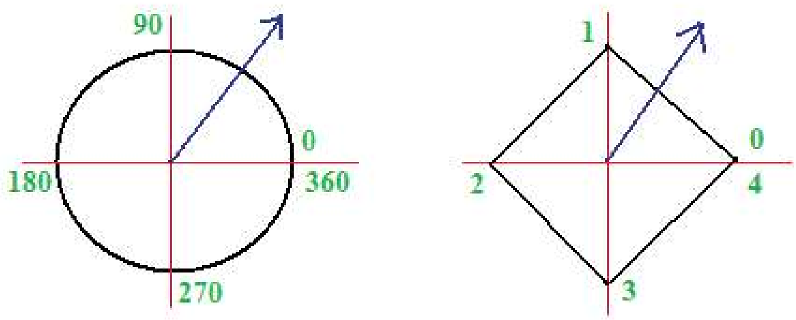
Freesteel Blog

- Front page
- Main blog
- Machining blog
- goatchurch

Freesteel Blog » Encoding 2D angles without trigonometry

Encoding 2D angles without trigonometry

Friday, June 5th, 2009 at 3:03 pm Written by: Julian



To prove I can still do work, here's for the machining category.

There are times when you need to encode the direction of a 2D vector. For example, the sequence of arc segments on the bottom rim of the cutter that are in material for the purposes of running the Adaptive Clearing algorithm.

These vector directions from the centre point to different points on the cutter need to be in angular order and analysed in relation to a reference direction.

Often when programmers see a problem to do with angles they use school mathematics and apply trigonometry and convert the direction into its representation in radians whose values they can then compare:

```
double RadianAngle(x, y)
{
    if (y > 0)
        return atan(x / y) + 1.5707963267948966;
    else if (y < 0)
        return atan(x / y) + 4.7123889803846897;
    else if (x > 0)
        return 0.0;
    else
        return 3.1415926535897931;
}
```

I really hate using radians. How can anyone justify a numbering system that cannot unambiguously represent the important perfect 90 degree right angle in floating point notation without needing to compare to `pi/2` to epsilon accuracy all the time? But that's another story.

The above function, which has the simple inverse `(cos(a), sin(a))`, has a lot of instability problems for small values of `y` that can be avoided by introducing more cases:

```
if ((y > 0) && (y < x))
   return atan(y / x);
else if ((y < 0) && (-y < x))
   return atan(y / x) + 6.2831853071795862;
```

```
    else if ((x < 0) && (x < fabs(y))
       return atan(y / x) + 3.1415926535897931
```

There is a standard C library function that encapsulates all of this mess into the single function `atan2(x, y)`, which programmers may be happy to use because it hides all this complicated and slow calculation.

But I'm not happy with it because I don't like trigonometry and I need the speed. So I use the following function (subject to the special cases):

```
double DiamondAngle(x, y)
{
    if (y >= 0)
        return (x >= 0 ? y/(x+y) : 1-x/(-x+y));
    else
        return (x < 0 ? 2-y/(-x-y) : 3+x/(x-y));
}
```

The result ranges from 0 to 4 with right angles being integers. Note how the calculation requires a single division and some sums, and you get a number you can use to sort a list of vectors by their angles. The inverse function is:

```
return P2((a < 2 ? 1-a : a-3),
            (a < 3 ? ((a > 1) ? 2-a : a) : a-4);
```

This doesn't result in a unit vector, so if you need it to be length 1 you need to normalize the result at the cost of 4 multiplications and one square root, which is still a lot better than any sine or cosine combined.

Check the calculation is correct for x,y>0:

```
DiamondAngle(x,y) = y/(x+y)
```

which is between 0 and 1, so the inverse comes out as:

```
InvDiamondArg(DiamondArg(x,y))
              = P2(1-y/(x+y), y/(x+y))
```

which is inline with `(x,y)` because both coordinates are positive and if we dot it with the perpendicular:

```
Dot(P2(1-y/(x+y), y/(x+y)), P2(y,-x))
              = y - y^2/(x+y) - x y /(x+y)
              = (x y + y^2 - y^2 - x y) / (x+y)
```

we get zero.

I guess the reason this way of doing it is unpopular is that it looks more complicated.

But if you think about it as projecting the vector down to the piecewise linear diamond shape and then parametrizing by length, it's easy. There's a bit of mess to compress the calculations down and merge the different cases to shorten the code, but once it's there, you don't need to look at it again.

Next week: Encoding 3D angles using the Octohedron

Friday, June 5th, 2009 Written by: Julian Machining Trackback URL for this entry

6 Comments

- **1.** Freesteel&hellip replies at 10th September 2009, 11:34 am :

  […] is a follow-on from the Diamond Angle article about useful encoding of plane angles without […]

- **2.** anders replies at 2nd July 2010, 11:05 am :

  I did a test today which showed about 25% better speed in one direction, and roughly 3-fold better for the other transform:
  http://www.anderswallin.net/2010/07/radians-vs-diamondangle/

- **3.** Nick replies at 17th July 2010, 10:39 am :

(This comment is directed more towards visitors than the author, this is how I wrapped my head around this concept) I discovered your site a few years ago and recently returned because of this post which I found interesting. I wanted to try and wrap my head around it, and a recent grad class helped. Once I understood that it is simply re-defining angles using the L1 unit circle (a diamond) as opposed to the standard L2 unit circle (a circle) it became easy to understand, and the rest is just algebra. What DID throw me off is the range of your angle: a = [0,4). As you know, the 'radian' angular measurement is defined by the distance traveled along the circumference of the unit circle in L2 space. If you follow the same definition in L1 space, the distance traveled around the unit circle is 8, thus should be: a = [0,8). When I did the algebra to convert the angle 'a' into a (x,y) point and saw that 'a/2' keeps showing up, it makes sense to just halve the angle straight off. Either way preserves the L1-norm: |x|+|y|=1. (For those interested, general Lp-norm is ||x||p=(|x1|^p+|x2|^p+… +|xn|^p)^(1/p), think c=(a^2+b^2)^0.5). This should be taught at EVERY engineering college, a 25%-300% speed increase is no laughing matter when your calling the function several million times.

- **4.** anderswallin.net ›&hellip replies at 15th December 2011, 4:35 pm :

  […] at Freesteel, Julian talks about using a "DiamondAngle" in the interval [0,4] for representing a plane angle instead of the usual radian in [0,2PI]. The […]

- **5.** Santiago L. replies at 10th January 2018, 2:23 am :

  Altought not the same as discussed in the post, I would recommend looking into rational trigonometry, to anyone trying to replace trigonometric functions with something that may be faster. There are some very interesting lectures by Norman J. Wildberger about the topic.

- **6.** KUB replies at 20th July 2018, 1:08 pm :

  I needed an approximation for "true" L2 space degrees, so I measured the error between these L1 space angles and "true" L2 space angles. I came up with a max error of about 4 degrees. It's possible to reduce this error to below one degree with this:

```
/* return angle in deg, [0..360) */
double angle(double x, double y)
{
double ang, quad;
if (y >= 0)
if (x >= 0) quad = 0, ang = y/(x+y);
else quad = 1, ang = -x/(-x+y);
else
if (x < 0) quad = 2, ang = -y/(-x-y);
else quad = 3, ang = x/(x-y);
if (ang < 0.75)
return 90 * (quad + ang + 0.2 * fabs(ang-0.25) – 0.05);
else
return 90 * (quad + ang + 0.2 * fabs(ang-1.25) – 0.05);
}
```

Leave a comment

Name (required)

Mail (will not be published) (required)

Website

XHTML: You can use these tags: <a href="" title=""> <abbr title=""> <acronym title=""> <blockquote cite=""> <code> <em> <strong>

Send comment

Archive

- February 2008
- January 2008
- December 2007
- November 2007
- October 2007
- September 2007
- August 2007
- July 2007
- June 2007
- May 2007
- April 2007
- March 2007
- February 2007
- January 2007
- December 2006
- November 2006
- October 2006
- September 2006
- August 2006
- July 2006
- June 2006
- May 2006
- April 2006
- March 2006
- February 2006
- January 2006
- December 2005
- November 2005
- October 2005
- September 2005
- August 2005
- July 2005
- June 2005

Meta

- Log in