Equal Spacing Along an Ellipse

nbodyphysics.com

Peter Musgrave

May 17, 2021

Gravity Engine draws the paths that objects will take in their orbits. Most of the time these orbits are ellipses. As you would expect in a game renderer the path must be specified as a sequence of line segments. The simplest approach to determining these points is to use a parametric form of the ellipse as a function of the angle from the origin (*not* the focus) by using a parametric equation for the ellipse. For an ellipse in (x, y) this could be $(a\cos\phi, b\sin\phi)$ with $\phi = 0$ corresponding to the x-axis. Here a is the semi-major axis of the ellipse and b is the semi-minor axis of the ellipse. When the ellipse is "not too elongated" stepping through equal angle increments gives a reasonable fit as shown below. However, when the ellipse is more eccentric (i.e. a is significantly larger than b) the density of points at the ends of the ellipse starts to give a jagged appearance.

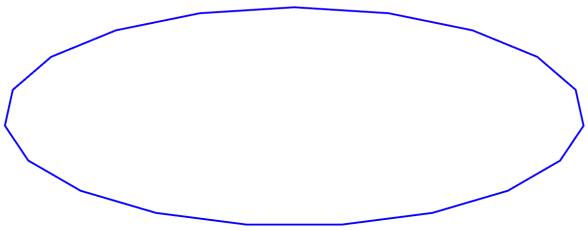

Figure 1: Ellipse with jagged perimeter (a small number of points was used to exaggerate the effect.)

A better approach would be to generate points along the ellipse that are spaced at equal distances along the perimeter. This seems like a straight-forward geometry problem that should yield to a bit of algebra. This turns out not to be true!

The length of a segment along the ellipse at a point p is given by the tangent at p times the length of the small segment. Let's make a slight change to the parametric equation and start the angle on the y-axis. The ellipse is defined as:

$$\gamma(\phi) = (a\sin\phi, b\cos\phi) \tag{1}$$

The tangent of this curve is the derivative with respect to ϕ:

$$\gamma\prime(\phi) = (a\cos\phi, -b\sin\phi) \tag{2}$$

The length along the ellipse is the result of integrating the magnitude of the tangent times the segment length, or:

$$s = \int |\gamma\prime(\phi)| d\phi = \int \sqrt{a^2\cos^2\phi + b^2\sin^2\phi} d\phi \tag{3}$$

(You can see that if this was a circle and $a = b = r$ then the trig. identity would reduce the integral to an integral of $rd\phi$ and if we integrated over the full circle we'd get $2\pi r$. This shows why the circle is a much simpler special case.)

Let's see what we can do to re-arrange this integral a bit. Let's push everything into the same trig. function, $\sin\phi$:

$$\begin{aligned} s &= \int \sqrt{a^2\cos^2\phi + b^2\sin^2\phi} d\phi \\ s &= \int \sqrt{a^2(1-\sin^2\phi) + b^2\sin^2\phi} d\phi \\ s &= \int a\sqrt{1 - \left(1 - \frac{b^2}{a^2}\right)\sin^2\phi} d\phi \end{aligned} \tag{4}$$

The final re-arrangement here may seem a bit contrived (why make the second term negative?) because it is. The reason for this re-arrangement comes from knowing that the form as written corresponds to a special kind of integral that we know something about. First of all, this is not one of those integrals where some clever substitution trick will allow progress. To be honest it does seem a bit weird that such an elementary integrand as $\sqrt{1 - c\sin^2\phi}$ is something we cannot solve in terms of elementary functions. That is however the case here. This integral comes up in enough places that it used as the *definition* of a new function $E(\phi, k)$:

$$E(\phi, k) = \int_0^\phi \sqrt{1 - k^2\sin^2\theta} d\theta \tag{5}$$

with $0 \leq k \leq 1$; where k is known as the elliptic modulus or eccentricity [4]. This function is given the name *elliptic integral of the second kind*. To be even more precise this may be called the incomplete elliptic integral of the second kind. The complete integral is the same integrand but with $\phi = \pi/2$. Note that there are notational variations in the definition and many numerical packages use a parameter $m = k^2$.

How does this help us in our quest to get points at equal distances around an ellipse? If we have a series of distances around the ellipse we would want to solve the equation:

$$s_n = aE(\phi, m) \tag{6}$$

for ϕ where $m = \sqrt{(1 - b^2/a^2)}$. This is again something for which there is not a simple closed form expression, since the variable we want to solve for appears in the limit of the integral. How annoying!

This is the point where we need to turn to specific cases and numerical tools. First we need a numerical implementation of $E(\phi, m)$ and then we need to find the root of the equation $aE(\phi, m) - s_n = 0$ given m and s_n. Numerical libraries such as SciPy [2] provide the tools we need. In the world of the Unity game engine we are not so lucky, so it was necessary to port the SciPy implementation into C#. My free Numerical Tools asset now includes this functionality.

Before we go further with the implementation, let's back up and consider what happens if instead of using the y-axis as the origin we use the x-axis. Since the integral involves magnitude of the tangent changing the origin to the x-axis results in $\gamma^2(\phi) = a^2 \sin^2 \phi + b^2 \cos^2 \phi$ i.e. it's as if we swapped a and b in (3). Following this all the way through the result would be (6) with $m = 1 - a^2/b^2$. This violates the restriction on m of $0 \leq m \leq 1$. The curious thing is that when this is implemented using the numerical routines, everything works out ok even though the values for m are negative. Note that if the ellipse has $b > a$ then the parametric equation that defines $\phi = 0$ as the x-axis would be the required choice to ensure $0 \leq m \leq 1$.

1 Implementation in Unity

The free asset Numerical Tools has support for $E(\phi, m)$ via the function `Special.Ellipeinc`. To evaluate the angle ϕ for a specific distance s the code will need to find the root of the equation (7) using `Optimize.FsolveSimple`.

$$aE(\phi, m) - s = 0 \tag{7}$$

The Unity scene `EllipseSpacing` implements a script of the same name and it performs this evaluation for an ellipse of given a and b. The core mathematical calls are:

```
Optimize.FsolveSimple(FindPhiForDistance, x, fvec, tol:1e-4);
```

and

```
private void FindPhiForDistance(int n, double[] x, double[] f, ref int iflag)
{
   f[0] = a * Special.Ellipeinc(x[0], m) - s;
}
```

The solution to the `Fsolve` call is passed back in the x array. Note the function `FindPhiForDistance` makes use of global variables for the parameters s and m since this simple root finder does not support passing function arguments.

If this seems like too much work to do every frame to update a line drawing of an ellipse, I completely agree. Is there something we can do with this more detailed understanding of equal spacing on an ellipse to create a more efficient algorithm?

2 Heuristic Approach

My first thought on a heuristic is to see if there is some way to work our way around the ellipse perimeter with a non-constant $d\phi$ that is some approximate function of phi. I'd expect this function of phi to depend on the shape of the ellipse i.e. be some function of the eccentricity $e = \sqrt{1 - b^2/a^2}$ where $a > b$. Being a visual thinker, let's plot this function for a variety of eccentricity values. In the interests of keeping things simple I made use of SciPy and the plot library (Figure 2).

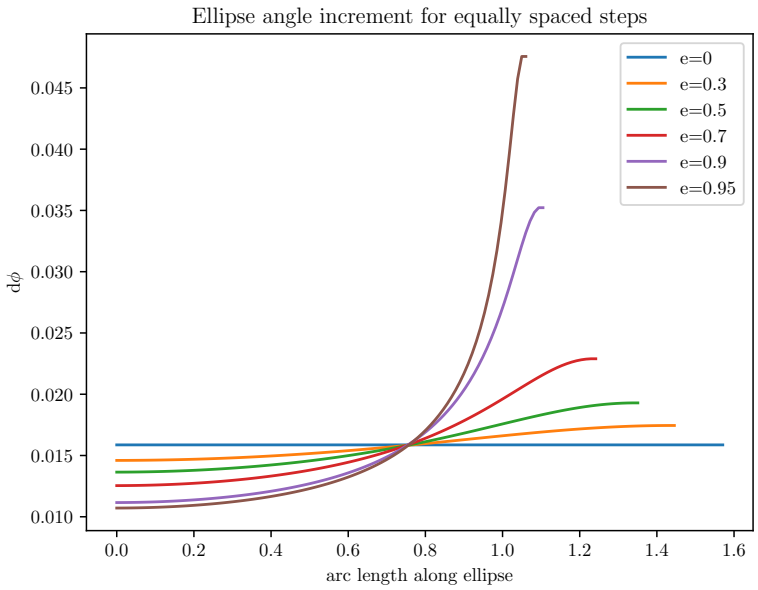

Figure 2: Plot of dϕ step required to cover ellipse quadrant with 100 steps.

The plot shows that even in the case of a highly eccentric ellipse ($e = 0.95$) the variation of $d\phi$ ranges from about 0.01 to 0.05, a factor of five. Pragmatically, it seems far simpler to just increase the number points in the ellipse by $5x$ rather than implementing the somewhat complex algorithm shown above. However, perhaps there is a heuristic approach.

Spacing around an ellipse became a topic of conversation over the breakfast table. My wife has a Ph.D. in Electrical Engineering and has great intuition for math problems like this. Her suggestion was to look at some ratio of actual distance to maximum distance. Cool idea! I decided on an approach that picked a fixed angle step size $d\phi$ and then scaled this up by a/r. This resulted in a scale change of about 10x for an eccentricity of 0.9. Figure 1 shows this is a bit too big, we're looking for something more like 4.5x. After a bit of experimentation I settled on $(a/r)^{2/3}$. The result is Figure 3. Implementing this in Unity we can

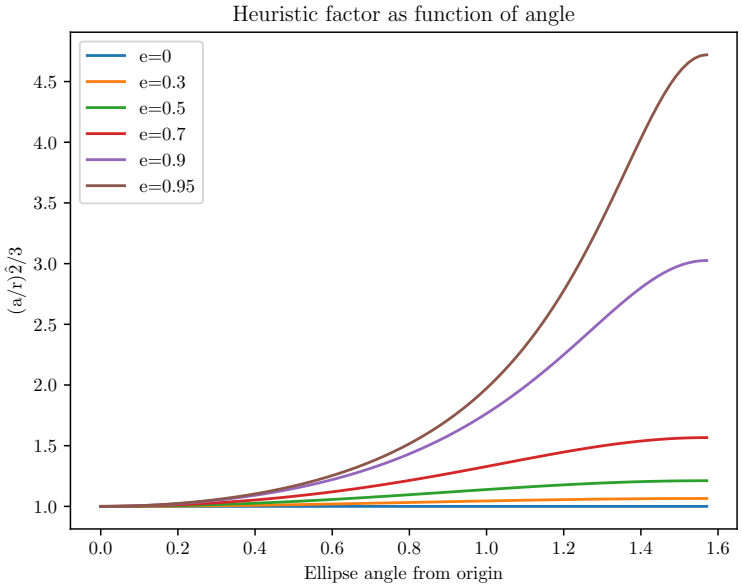

Figure 3: Plot of dϕ scale factor as a function of ϕ.

compare the exact solution with this heuristic (Figure 4). It does pretty well. This is something I'll try in the Gravity Engine ellipse renderers.

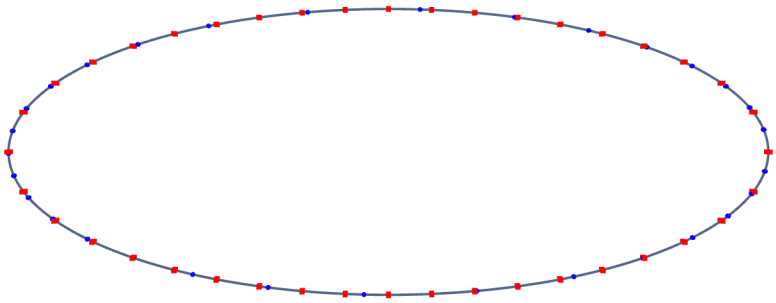

Figure 4: Comparison of equal arc length (red squares) versus $(a/r)^{2/3}$ heuristic (blue circles).

For those who want to tinker with these algorithms, the code is now in the Numerical Tools asset. Numerical Tools is available for free on the Unity asset store [1].

I'd like to acknowledge the thread [3] on Math Stack Exchange for inspiration.

References

[1] Peter Musgrave. *Numerical Tools Unity Asset*. 2021. URL: https://assetstore.unity.com/packages/tools/physics/numerical-tools-192894.

[2] SciPy.org. *SciPy: open source software for mathematics, science and engineering*. 2021. URL: https://docs.scipy.org/doc/scipy/reference/index.html.

[3] Math StackExchange. *Calculating equidistant points around an ellipse arc*. 2015. URL: https://math.stackexchange.com/questions/172766/calculating-equidistant-points-around-an-ellipse-arc.

[4] Wikipedia. *Elliptic Integral*. 2021. URL: http://en.wikipedia.org/wiki/Elliptic_integral.