# Avoiding some floating point pitfalls
Sam Buss

October 18, 2004

For the programming project #3 assignment, you will need to create a surface of rotation. This will require writing loops that draw all the vertices; indeed, you will have an outer loop that loops over all quad strips and an inner loop that outputs all the vertices of the quad strip. Likewise, you will need a loop that generates the vertices of the triangle fan.

The purpose of this note is to discuss some potential pitfalls. The short moral is: **Do not use floating point variables to control the stopping condition of a loop**.

Suppose you need to generate equally spaced real values that split the internal $[0, 2\pi]$ into equal length intervals. To begin with, here is a very bad piece of code:

**1. Bad code:**
```
float x;
for ( x = 0.0; x!=2.0*PI; x += 2.0*PI/MeshCount ) {
    // Use the value of x here.
}
```

For the above code, `PI` is a floating point value defined elsewhere. Also, `MeshCount` is a small positive integer value defined elsewhere.

The problem with the above code is that, due to floating point roundoff, the value `2.0*PI/MeshCount` will not be exactly correct with the exception of values of `MeshCount` that are powers of sixteen. Thus, $x$ may never *exactly* equal `2.0*PI`, and the loop will never exit!

A slightly better piece of code is next, but this is still bad:

**2. Bad code:**
```
float x;
for ( x = 0.0; x<2.0*PI; x += 2.0*PI/MeshCount ) {
    // Use the value of x here.
}
```

This second sample code will at least always terminate. However, due to roundoff errors, after `MeshCount` iterations, `x` will sometimes be slightly less than `2.0*PI` and sometimes slightly more than `2.0*PI`. In the former case, the loop will do one more iteration than expected.

A better method is to use an integer variable to control the loop parameter. For example:

**3. Good code:**
```
float x = 0.0;
for ( int i = 0; i<MeshCount; i++ ) {
    x += 2.0*PI/MeshCount;
    // Use the value of x here.
}
```

For the loops in the class programming assignments, the loop will probably include the last value $2\pi$ and will actually use the value $\sin(x)$ and $\cos(x)$. The first thing you might try is:

**3. Fair code:**
```
float x = 0.0;
for ( int i = 0; i<=MeshCount; i++ ) {
    x += 2.0*PI/MeshCount;
    float sinX = sin(x);
    float cosX = cos(x);
    // Use the values of sinX and cosX here.
}
```

We called this "fair" code since it might appear to work correctly if you test it lightly, but it will still have problems in some cases. The problem is that, due to roundoff errors, x, will not end up exactly equal to $2\pi$, and then for the last iteration of the loop, `sinX` and `cosX` will not have the desired values of 0 and 1. If these wrong values are used to draw a vertex with `glVertex`, then the vertex may end up on a wrong pixel, and leave a visible gap in the surface of revolution.

A better way to to write this code is:

**3. Good code:**
```
float x = 0.0;
for ( int i = 0; i<=MeshCount; i++ ) {
    x = (i%MeshCount)*2.0*PI/MeshCount;
    float sinX = sin(x);
    float cosX = cos(x);
    // Use the values of sinX and cosX here.
}
```

This code uses i%MeshCount so that x will be exactly zero when i equals `MeshCount`.