Fun with Bezier curves and for loops

by Benny Bottema - Thursday, July 08, 2010

http://www.bennybottema.com/2010/07/08/fun-with-bezier-curves-and-for-loops/

Bezier curves are cool. They are curves calculated through a formula with control points. These control points are crucial, as these determine how long, big and curved the Bezier curves are. There are optimized formulas for three, four and five control points, but in this post we'll deal with an arbitrary number of control points and interactively drawing the curves.

(download source)

To get some background information on Bezier curves, quadratic and cubic curves, visit Bezier, cubic Bezier curves in flash.

I was doing some personal research in Bezier curves and was interested in finding way to draw Bezier curves through a dynamic number of control points. The problem was, everywhere I looked for information it was always about Bezier curves up to 5 control points, which were calculated with an optimized formula for exactly that amount of control points (a simplified version of the general formula for n+1 control points). So I wrote my own take on it; one that can handle an unlimited amount of control points added dynamically and lets you control how smooth the curves are.

Bezier curves – from math to code

empty setup preview (source)

Here we go. Take a look at this fla. The only code in it is to create the draggable control points and the empty *drawBezierCurves* function that is being called when a controlpoint has changed (because you dragged it, or added new ones). The buttons at the bottom reset the movie with a number of control points, and the buttons below that change how smooth the line should be. The textfield in the upper right corner indicates how fast the Bezier curve was drawn. We'll add more to this movie as we go along.

Now take a look at this Bezier formula.

$$B(u) = \sum_{k=0}^{N} P_k \frac{N!}{k!(N-k)!} u^k (1-u)^{N-k} \Big\} 0 \le u \le 1$$

Converted into Actionscript, this is:

```
(factorial(N) / (factorial(k) * factorial(N - k))) * Math.pow(u, k) *
Math.pow(1 - u, N - k);
```

```
// where factorial '!' is our custom recursive function

function factorial(x:Number):Number {
 return (x <= 1) ? 1 : x * factorial(x - 1);
}
```

To actually calculate x and y, consider the following example, where we have 5 control points, and thus N is 4 (the formula says N+1 control points). The control points are the movieclips that you can drag.

```
var stepsize:Number = 1/30;
var N:Number = 4;
for (var u = 0; u <= 1; u += stepsize) {
  x = y = 0;
  for (var k = 0; k <= N; k++) {
   var blend:Number = (factorial(N) / (factorial(k) * factorial(N - k)
)) *
Math.pow(u, k) * Math.pow(1 -  u, N - k);
   x += this["controlPoint" + k]._x * blend;
   y += this["controlPoint" + k]._y * blend;
  }
  this.lineTo(x, y);
 }
```

First of, *u* is the ‘lifetime’ variable of the Bezier curve. For example u = 1 is all the way to the end of the curve on the last control point, u = 0 is at the beginning at the first control point, and u = 0.5 is exactly in the middle of the line. The smaller the steps you take, the smoother the line is going to be, because you use smaller intervals. Also, the stepSize has to be at least as big as the number of control points. Otherwise, the line won’t even reach the control points on the end.

This is what the movie does with this code: example1 (source)

Cool huh, this works for every N. Try it at 15 control points in overkill mode; it now is doing (15 control points * 6 smoothmodifier) steps * 15 = 1350 cycles of Bezier calculations on *each* update. See the speed at around 30ms in standalone player? As you can imagine, this is rather heavy duty for the cpu, as it has to do three factorials, two powers and some multiplications for every x,y point we calculate each cycle. In older players this used to be 220ms so things have been improving, but this is why most examples you find don’t use this method and instead use a formula that is optimized for (and limited to) up to 5 control points. For example this Bezier summary by Paul Bourke shows special cases for exactly 2 and 3 control points. This greatly increases speed, but you are limited to 2 and 3 controlpoints.

There is another way however to speed things up: Lookup tables. First, we’ll create a lookup table for the factorial calculations.

```
var factorialTable:Array = new Array();

for (var k = 0; k <= N; k++) {
 factorialTable[k] = factorial(k);
}
```

You see, we only calculate the values that we’ll need for the Bezier curves. No more, no less. However, this means we have to update the lookup table every time we add or remove a control point.

Now the lookup table for power.

```
var powerTable:Array = new Array();

for (var u = 0; u <= 1; u += stepsize) {
 if (powerTable[u] == undefined) {
  powerTable[u] = new Array();
 }
 if (powerTable[1 - u] == undefined) {
  powerTable[1 - u] = new Array();
 }
 for (var k = 0; k <= N; k++) {
  powerTable[u][k] = Math.pow(u, k);
  powerTable[1 - u][k] = Math.pow(1 - u, k);
 }
}
```

Now, our method to calculate the Bezier curves looks like this:

```
var stepsize:Number = 1 / 30;
var N:Number = 4;

for (var u = 0; u <= 1; u += stepsize) {
 x = y = 0;
 for (var k = 0; k <= N; k++) {
  var blend:Number = (factorialTable[N] / (factorialTable[k] *
factorialTable[N - k])) * powerTable[u][k] *  powerTable[1 - u][N - k]
;
```

```
  x += this["controlPoint" + k]._x * blend;
  y += this["controlPoint" + k]._y * blend;
 }
 this.lineTo(x, y);
}
```

I also made a version with a lookup table for the Bezier blend values instead of power and factorial values. Here are the results.

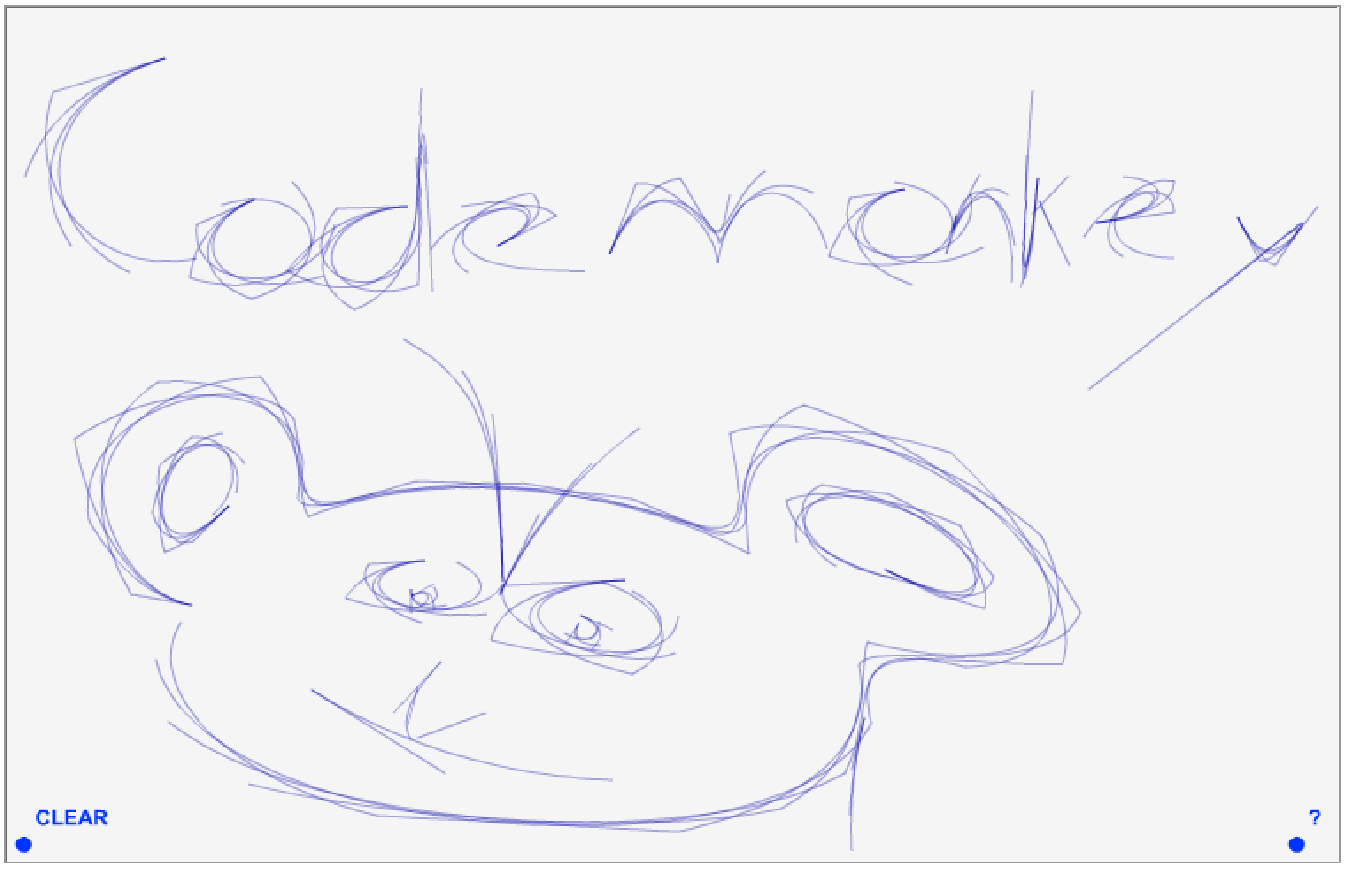
- power & factorial lookup table source
- Bezier blend value lookup table source

```
/wp-content/uploads/2016/02/JelloWave.swf, 600, 300
```

(download source)

Here are some more experimental Bezier examples:

- Bezier controlpoint demonstration (source)
- JelloWave.html (source)
- Globb_a.html (source)
- Globb_b.html (source)
- donut.html (source)
- Globbor.html (source)
- trailer.html (source)
- wheeler.html (source)
- sketcher.html (source)
- Bezier3D.html (source)



PDF generated by Kalin's PDF Creation Station