GENETIC PROGRAMMING: MONA LISA FAQ

December 9, 2008 Roger Johansson Domain Specific Languages, Evolution 114 comments

This is a follow up to: http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/

Mona Lisa FAQ

Here are some answers to the most frequently asked questions:

Q) What is the use for this? this is nothing but a fun toy

A) Exactly… It was a 3 hours fun hack for my friends and readers…

Search

BLOG STATS

- 1,453,944 Views

TOP POSTS

- Why mapping DTOs to Entities using AutoMapper and EntityFramework is horrible

Q) What is your population size?

A) TWO , Parent and Child is competing against eachother each generation.

Q) Is this Genetic Programming?, I think it is a GA or even a hill climbing algorithm.

A) I will claim that this is a GP due to the fact that the application clones and mutates an executable Abstract Syntax Tree (AST).

Even if the population is small, there is still competition between the parent and the child, the best fit of the two will survive.

Sample:

```
1  Drawing
2
3      DrawPolygon (
4         Brush(10,30,50,30),
5         Point(554,93) , Point(67
6      )
7
8      DrawPolygon (
9         Brush(50,1,43,66),
10        Point(66,112) , Point(12
11     )
12
13     DrawPolygon (
14         Brush(32,100,22,87),
15         //dynamic number of poi
16         Point(423,342) , Point(
17     )
18
19 end
```

Q) What is your fitness function?

A) The fitness function is a pixel by pixel compare where the fitness for each pixel is summed and compared to the parent.

Pseudo Sample:

```
double fitness = 0
for y = 0 to width
    for x = 0 to height
          color c1 = GetPixel(s
          color c2 = GetPixel(g

          //get delta per color
          double deltaRed = c1.
          double deltaGreen = c
          double deltaBlue = c1

          // measure the distan
          double pixelFitness =



          //add the pixel fitne
          fitness += pixelFitne
    end
end

return fitness
```

Q) How long did the generation take?

A) About 3 hours on my laptop.

Q) Where can I get the source code?

A) Here: http://rogeralsing.com/2008/12/11/genetic-programming-mona-lisa-source-code-and-binaries/

Adding more soon….

Related

Genetic Programming: Evolution of Mona Lisa
In ".NET"

Genetic Gallery
In "Evolution"

Genetic Programming: Mona Lisa Source Code and Binaries
In ".NET"

« Genetic Programming: Evolution of Mona Lisa

Genetic Programming: Mona Lisa Source Code and Binaries »

114 COMMENTS

myriam says:

December 9, 2008 at 5:22 pm

Could I have a pic with just the polygons (outline) in the final stage? I want to see side-by-side the final picture and the final polygons. my address is mabramson@gmail.com.

are you evolving just the colors or also the geometry?

Reply

Chewie007 says:

December 9, 2008 at 5:27 pm

Will the polished version allow for users to select a picture file other than Mona Lisa?

Reply

Dan says:

December 9, 2008 at 5:31 pm

Roger, thanks for the clarifications. The only other question I would have would be how did you mutate the drawing? What probabilities did you use?

Reply

Jered Floyd says:

December 9, 2008 at 5:33 pm

Fascinating. This reminds me a lot of fractal compression using IFSs from “back in the day”. It shares the attribute of being resolution independent; a nice property for image compression.

What sort of effective compression ratio do you see at the default image resolution? That is, if you use a standard compression routine (e.g. ZIP) to compress your presumably non-optimal DNA language, what is the output file size compared to the source?

Reply

wulwife says:

December 9, 2008 at 5:53 pm

very cool…i'm impressed

Reply

Pingback: Genetic Programming: Evolution of Mona Lisa « Roger Alsing Weblog

Rhys Ulerich says:

December 9, 2008 at 6:20 pm

Will you please give an idea of the effective compression rate you're seeing?

Reply

Just curious says:

December 9, 2008 at 6:35 pm

Thanks for the follow-up, dude. Much

appreciated.

Reply

Mike says:

December 9, 2008 at 6:52 pm

I don't know if there'd be a benefit but it'd be interesting fun to play with replacing pixelFitness() with a Delta E colour difference measure, i.e. http://www.colorwiki.com/wiki/Delta_E:_The_Color_Difference.

RGB colourspace isn't good for measuring colour differences – so there might be faster convergence and a better visual result with Delta E 1976, Delta E 1994 or Delta E 2000. Though the potential trade off is the computational expense of Delta E calcs and RGB to LAB colourspace conversion.

Very neat btw.

Reply

kdborg says:

December 9, 2008 at 6:56 pm

For a performance boost, I'd take out the sqrt() in the fitness function and just leave the multiplications:

```
double pixelFitness = deltaRed * deltaRed
+ deltaGreen * deltaGreen
+ deltaBlue * deltaBlue
```

While the sqrt() would give the proper value. Without the sqrt() still gives a valid indicator of fitness.

Reply

Steve says:

December 9, 2008 at 6:58 pm

I'm going to echo the comments of somebody else in the other thread… the program needs to also output in SVG.

Reply

Daniel Pope says:

December 9, 2008 at 7:03 pm

You don't need to do that Sqrt().

Also, you've got a . rather than a – in the blue channel.

Reply

jae says:

December 9, 2008 at 7:23 pm

is that a typo in line 10 for the fitness func?

did you mean a “-” operator instead?

Reply

Roger Alsing says:

December 9, 2008 at 7:29 pm

@kdborg

You are right about the SQRT.

I have removed this from the application and updated the pseudo sample.

Thanks

//Roger

Reply

Steve says:

December 9, 2008 at 7:47 pm

@Daniel ~ the fitness function would probably work without the Sqrt(), but its presence is still mathematically significant because we’re looking at the picture as a whole.

With the Sqrt(), the match will get worse in a

linear fashion when a region of the picture gets worse. Without it, the match will get worse quadratically.

The question boils down to whether, for example, a single pixel that's off by 20 colour units (measured in 3D space as it was in the function) is better or worse than two pixels that are off by 10 colour units each (according to the function presently, they're equivalent, 20 vs. 20; take away the Sqrt and the one pixel is much worse, 400 vs. 200). If the one bad pixel is worse than the two less bad ones, the Sqrt() slows down the convergence.

You'd probably get the fewest iterations if you determined the ideal exponent (instead of 1/2) to describe what the relationship between one "really bad" pixel and two "less bad" ones was, and used that exponent instead. Then you'd waste more time calculating the power, though, so it's probably not worth it.

Reply

Josh says:

December 9, 2008 at 7:54 pm

You're smart.

Reply

Gilles says:

December 9, 2008 at 7:57 pm

This is a great compression methode! Did you think about making it in 3D to compress à movie?

Reply

catblade says:

December 9, 2008 at 8:46 pm

Have you considered trying HSI instead of RGB for each pixel? This might reduce the noise more quickly as it would simplify it from the RGB version and the distance would be a bit more precise.

Reply

xteraco says:

December 9, 2008 at 9:09 pm

With no source, and no app to play with, I'm not sure I believe this. :)

Have you guys seen the car demo?
http://www.wreck.devisland.net/ga/

Reply

John W. Ratcliff says:

December 9, 2008 at 9:11 pm

If you are willing to share the source code to your app, regardless of how 'clean looking' it is, I can rework it so it will probably run many orders of magnitude faster as that is kind of my specialty.

I would enjoy messing with the algorithm.

John

Reply

blaze says:

December 9, 2008 at 9:14 pm

Have you thought about adding total polygon complexity to the fitness function?

Reply

shamus says:

December 9, 2008 at 9:26 pm

To make further speed ups beyond the removal of sqrt() you may find that sum of absolute difference is an equivalent, and faster still fitness function. Where:

pixelFitness=abs(deltaRed) + abs(deltaGreen) + abs(deltaBlue);

Reply

Roger Alsing says:

December 9, 2008 at 9:44 pm

The performance problem is not really in the comparer but in the renderer.

The renderer is using .NET GDI+ which is a software based renderer which is fairly slow.

So thats where I'm wasting CPU atm.

Reply

Pat Scandalis says:

December 9, 2008 at 10:31 pm

This is fantastic. In thinking of this as an asymetric compression algorithm, the basis function is a semi-transparent colored polygon and the encoding algorithm (transform) is the GP itself. As an Audio DSP engineer I have to wonder if audio compression could be achieved with a new basis function (not A(n)*sin(…)) and a GP could be used to encode.

Reply

ishmal says:

December 9, 2008 at 10:35 pm

I wonder what is the total node count (all of the corners)? This might be a good general mechanism to vectorize a bitmap. More CPU -> better compression for a given rendering quality.

Reply

ToniT says:

December 9, 2008 at 10:44 pm

About the first sample code:

How are the values for the polygons/brushes set ? Are these just random values ? And after each iteration they change randomly a bit to see if the new version is better ? And what exactly is “a bit” ? I think randomly change everything would be too much.

Why is the last polygon made up of dynamic points (>3) while the first two polygons are ones which just 3 points ?

Reply

A.W says:

December 9, 2008 at 10:47 pm

This is neat, but its *Not* Genetic programming. The individuals of the population are not executed and are not programs. you call it a executable abstract tree but I would contend that its a simple data structure that has no power to do computation on its own.

Reply

Roger Alsing says:

December 9, 2008 at 10:58 pm

>>you call it a executable abstract tree but I would contend that its a simple data structure that has no power to do computation on its own.

Would you say there is a difference if the root of the AST returned an INT or a Bitmap ?

What makes the INT version more of a program than an AST that returns a Bitmap?

It's not like an "Add" node in a computational tree is more complex than a "DrawPolygon" node.

Call it Genetic declarative programming if you like ;-)

At what stage do you consider a program to

be a program?

—

string x = “hello world”

print x

—

Is that a program?
It does not compute anything and is merely rendering the data stored in X onto the screen.

Reply

Eric Haines says:

December 9, 2008 at 11:07 pm

Nice! I’d love to see what the individual polygons looked like at the end.

Reply

John W. Ratcliff says:

December 9, 2008 at 11:12 pm

Yeah, the first thing I was going to do was implement it using Direct3D hardware and custom shaders.

That would speed up rendering dramatically.

John

Reply

foodpoison says:

December 9, 2008 at 11:13 pm

very cool, but you should get better results if you always kills off the parents but have more children to choose from; because keeping the parents you can quite often get stuck in a place where the parents are “good enough” but all their 1-mutation-step away children are not better, i.e. “stuck in a valley” when people talk about hill climbing; whereas if you always kills off the parents you won’t get this problem.

Reply

RPK says:

December 9, 2008 at 11:22 pm

What if, instead of comparing pixels against the Mona Lisa to determine fitness, you allowed random internet users to vote on the “beauty” of each successive generation (say on a scale of -1 to 1) where users would compare the new generation to the old generation and vote on the improvement (or lack thereof). The goal of the program would be to obtain a consistently high rating. Given enough time, would we end up with a

beautiful piece of abstract art or a muddled mess? Or what if the human evaluators were asked to judge not the beauty of the painting but whether each successive generation looked more or less like particular figure, such as a face. Would we be able to "crowd-source" our way toward something recognizable?

Reply

Yiding says:

December 9, 2008 at 11:32 pm

this looks more like a local optimization problem than genetic programming.

Reply

Christian Buchner says:

December 9, 2008 at 11:37 pm

Can't resist to implement this in one of OpenGL + CUDA or DirectX + CUDA. Consider a speed up of 1000 compared to a software render.

It might even become feasible to apply this to movie clips, taking the polygons from the previous frame as a basis to start mutating for the current frame. Encode the movement

of the polygon edges (and any removed/additional ones) and you've got a resolution-independent video codec.

Source code, please ;)

Reply

Mark IJbema says:

December 10, 2008 at 1:10 am

I think it's more simulated annealing than genetic programming, if anything, but it isn't really SA, since it is easy to get stuck in a local optimum. I'd suspect that actually using simulated annealing (sometimes accepting a child that is worse than the parent) will give better result. Check wikipedia or something for a description. I'm really interested to see using it makes the picture better.

Reply

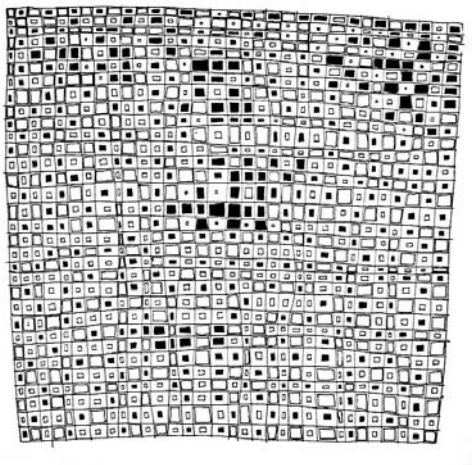
a bill says:

December 10, 2008 at 2:58 am

Just wanted to drop a note that this is a great project. Amazing Really!!!

Although its sightly unrelated to your coding, I had to share a project of mine. Using the

same image (Mona Lisa), I reduced color and pixel dimension, then transcoded that into a drawing fill pattern and replicated the image by hand. Here was my result:



Reply

Ben FrantzDale says:

December 10, 2008 at 3:00 am

Very cool. Could you explain what the random-update function is? How are the “genes” mutated? Your other page says you “mutate it slightly”. I assume your configuration space involves n polygons each with some number of vertices and with a color. Are you just adding Gaussian random numbers to those continuous variables, or what?

It looks like this is a random down-hill walk through state space. Have you tried something like simulated annealing instead?

http://en.wikipedia.org/wiki/Simulated_annealing It seems like that might get you a more-optimal solution.

Alternately, there are many other optimization algorithms you might use that might work better/faster. One that I haven't tried but that looks interesting for cases in which an analytical derivative does not exist is this: http://www.applied-mathematics.net/optimization/rosenbrock.html

Happy optimization.

Reply

Lynch82 says:

December 10, 2008 at 3:32 am

Hi Roger,

Firstly, this is very interesting stuff. I think the debate about whether it is really genetic programing, or evolution, or what not, is a bit silly… the point is: it is interest, and it does apply the basic premise of genetic programing and genetic algorithms.

When I saw this yesterday, I was inspired to tinker with it myself. I am in the process of making a C# app that does similar, but not identical thing, trying to match any picture. It

is a great thought exercise.

With regards to speed: I two am using GDI, because I am more interested in the concept then making it super pretty, but I am not rendering each generation to the screen, I am rendering them to non-visible pictures. This seems to be quite high performance, is that what you are also doing?

Reply

Rob Agar says:

December 10, 2008 at 3:34 am

Very, very cool! As a programmer I particularly like your reason for not releasing the code yet :)

Reply

Ray says:

December 10, 2008 at 4:18 am

Cool. Had the same idea decades ago, back on Commodore 64, but that machine wasn't capable for such a things. I was sure it would be too slow even on modern machines (hours per frame IS slow) and / or the compression ratio wouldn't be really impressive at acceptable quality. And some academic

people (I'm just a game programmer / demoscener) already axed the algorithm, so there's no reason to toy with it.

Anyway, looks impressive, I wonder how fast it can be after optimizations, and what is it's compression ratio. It would be interesting to test it on real life photos (lena.bmp) with only triangles (more of them) in different colorspace. Unfortunately, even if it's compression ratio is really good, I'm afraid it wouldn't be useful in video compression.

Reply

James W says:

December 10, 2008 at 5:15 am

This is really quite amazing. I'm curious to see how well it gets results with other photos – the "enigmatic" quality of the smile is particularly well represented here.

The chief difficulties of the GP is the selection of the fitness function. Another would be the creation of a large enough population to find the best fit. Given the (almost) zero cost per child, what would be the most efficient number of children to spawn before selection? Would there be benefit to merging the top two (three, four…n) fittest in someway to introduce the changes rather

than random mutations? Could there be other qualities that could benefit the selection? For example, have one trait determined by edge detection to create a line drawing, have another trait to select by colour fidelity. The more traits, the better the return.

Reply

Xomex says:

December 10, 2008 at 5:17 am

How many polygons and how long to create the guy who can paint Mona Lisa? :-)

Reply

Nik says:

December 10, 2008 at 6:07 am

@RPK – google for Electric Sheep – similar concept.

Reply

Pierre says:

December 10, 2008 at 6:55 am

This is brillant! I was so inspired by your post that I implemented my own version of the