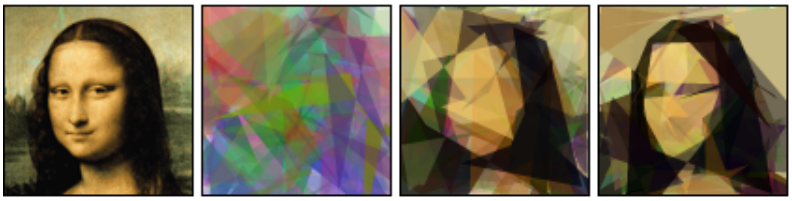
Introducción a la Programación Multinúcleo
Evolución genética de imágenes

Daniel Barreto - #04-36723 <daniel@gia.usb.ve>
Ernesto Level - #05-38402 <ealevel@gmail.com>

9 de diciembre de 2009



1. Introducción

El presente trabajo expone la utilización de un algorítmo genetico para "evolucionar" la mejor aproximación hacia una imagen usando una cantidad finita de polígonos sobrepuestos y semi-transparentes, y estudiar el comportamiento del algorítmo en la arquitectura **Cell Broadband Engine** de IBM.

1.1. Evolucionando imágenes

La "evolución de una imagen", en el contexto de este proyecto, consiste en conseguir configuraciones de un número estático y finito de polígonos (dado como un parámetro del algorítmo) para aproximarse a una represetación de una imágen dada a través de técnicas de programación evolutiva.

Una configuración de poligonos está dada como una lista de polígonos semi-transparentes sobrepuestos entre ellos, siendo dibujados en el orden en el que salen en la lista.

Por su parte, los polígonos estan representados simplemente como una secuencia de vertices, un color de base (en formato **RGB**) y un nivel de transparencia `alpha`.

De esta forma, la estrategia evolutiva del algorítmo es usar las nombradas configuraciones como individuos a evolucionar, tratando de mejorar su función de *fitness*, la cual es explicada más adelante en la sección 3.7

2. Marco teórico

2.1. Algorítmos genéticos

Los algorítmos genéticos son parte de una línea de estudio en la *inteligencia artificial*, son llamados asi porque "se inspiran en la evolución biológica y su base genético-molecular. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una Selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados."[3].

Una representación generalizada de un algoritmo genético se encuentra más detallada en el algoritmo 1.

2.2. Investigaciones previsa y aproximaciones al problema

Hasta el momento se han realizado dos estudios para buscar la solución de la evolución de imagenes. Estas han sido realizadas por Roger Alsing[1], pionero en la idea de conseguir aproximaciones a una imagen usando polígonos, y Jacob Seidelin[2].

1. **Roger Alsing:** Utiliza una población de sólo un individuo, que representa una cantidad de polígonos variable, empieza con un único polígono y luego va agregando mas. Realiza mutaciones con poca probabilidad para ir variando las posiciones y los colores de los polígonos que representa el individuo. Parecido a *Hill Climbing*

2. **Jacob Seidelin:** Utiliza una población variable, pero de más de un individuo. Los individuos con mejor fitness se cruzan y se generan nuevas poblaciones. Aproximación netamente genética.

2.3. Estructuras y representaciones genotipos

Para la implementación realizada, la cual es explicada en la sección 3, se utilizó una lista de números *flotantes* entre $[0, 1]$ (ambos inclusive) para representar el ADN de un individuo. Esta

lista de flotante contiene, o representa, a su vez una secuencia de polígonos, que son los elementos claves de cada *ADN*.

Un polígono está dado como: un color (r, g, b), un nivel de transparencia u opacidad y una secuencia de puntos (x, y) que representan sus vertices. Luego la lista de flotantes que representa a un polígono es de la siguiente forma:

```
r g b a v1x v1y v2x v2y ... vNx vNy
```

Dónde cada item es un número flotante. Luego podemos calcular que el tamaño de la representación de un polígono es: $4 + 2 * numero\ de\ vertices$, y de la misma forma, podemos ver que el tamaño de la lista de flotantes de un *ADN* es: $(4 + 2 * numero\ de\ vertices) * numero\ de\ poligonos$

3. Implementación

Como ya se ha dicho anteriormente, para la resolución de este problema se utilizó un algoritmo genético, éste parte de la creación de una población de individuos generados aleatoriamente.

A partir de esta población se genera una nueva, esto se hace tomando los mejores individuos para realizar en ellos los *operadores genéticos* de recombinación y mutación. Esta nueva población generada toma el lugar de la población anterior, y en caso que el mejor individuo de esa población sea el mejor individuo encontrado hasta el momento, se dibuja. Este último de paso se hace un número no determinado de veces, dependiendo de que tan bien estén saliendo los individuos de las nuevas generaciones. Mientras más veces se realice, mejores individuos se van consiguiendo y mejor la solución generada se parece más a la imagen original.

3.1. Variables aleatorias del algoritmo

El algoritmo implementado contiene **6** variables aleatorias, las cuales son explicadas a continuación:

1. **Tamaño de la imagen:** El tamaño en pixeles de la imagen, para este proyecto se usan solo imágenes cuadradas, por lo tanto el tamaño del ancho y del alto son exactamente iguales
2. **Número de polígonos:** Cantidad de polígonos con los cuales se va a intentar conseguir la mejor representación posible de la imagen dada. Cada individuo será una representación de esta cantidad de polígonos.

3. **Número de vertices por polígono.**

4. **Tamaño de la población:** Tamaño de la población a evolucionar.

5. **Tasa de mutación:** Probabilidad de un individuo en mutar para la siguiente generación.

6. **Cantidad de hijos por padre.**

7. **Cantidad de padres que serán cruzados.**

Siendo las últimas cuatro netamente del algoritmo genético.

Es importante notar que las variables **6** y **7** multiplicadas deben dar el mismo número que el establecido en la variable **4**.

3.2. Configuración utilizada

Tomando en cuenta las variables anteriores, para el desarrollo de nuestro proyecto y los resultados mostrados en la sección 5 usamos la siguiente configuración:

- **Tamaño de la imagen:** 128
- **Número de polígonos:** 48
- **Número de vertices por polígono:** 6
- **Tamaño de la población:** 36
- **Tasa de mutación:** 2 %
- **Cantidad de hijos por padre:** 6
- **Cantidad de padres que serán cruzados:** 6

3.3. Descripción del algoritmo usado

A continuación se presenta la base del algoritmo utilizado para la resolución del problema, considerando la configuración explicada en la sección anterior.

El algoritmo 1 muestra el algoritmo genético utilizado para la resolver el problema.

Algoritmo 1 Algoritmo Genético

Crear una población inicial con individuos generados aleatoriamente.
for i **in** $1..n$ **do**
  Generar una nueva población a partir de la población anterior.
  Reemplazar la población actual con la nueva población.
  if El mejor individuo de la población actual es el mejor individuo **then**
    Dibujar mejor individuo de la población actual.
  end if
end for

3.4. Patrones de selección

Para la generación de la nueva población se decidió tomar los mejores 6 individuos de la población como los padres, y por cada uno de ellos tomar 6 otros individuos escogidos aleatoriamente y cruzarlos, dando como resultado una población de 36 individuos.

3.5. Cruce de individuos

El nuevo individuo a crear será una mezcla de los polígonos que conforman los ADN de ambos padres, elegidos uniformemente aleatorios. Esto se hace con un factor de probabilidad del 50 % entre tomar el cromosoma del primer padre o tomar el del segundo padre.

3.6. Mutación del individuo

El nuevo individuo generado pasa un proceso de mutación, en donde con una probabilidad del 2 % se decide si se muta uno de los cromosomas del individuo o no, esto se hace para todos sus cromosomas.

En caso que se desee mutar el cromosoma, se tomará el cromosoma y podrá mutar hasta un 10 %; esto significa que su valor podrá variar hasta un máximo de 10 % por encima o debajo de su valor. Así:

$$cromosoma = cromosoma + random * 10\,\% * 2 - 10\,\%$$

Considerando que el *random* genera número entre 0 y 1, se tiene que el máximo número resultante sería $cromosoma + 0,1$ y el mínimo $cromosoma - 0,1$.

3.7. Función de Fitness

La función de fitness utilizada simplemente calcula pixel a pixel la diferencia de colores entre la imagen original y la imagen generada; esto permite distinguir a todos los individuos generados y tomar el mejor, es decir, quien tenga mayor fitness.

$$F_{fitness} = \sum_{pixel} \delta_r + \delta_g + \delta_b$$

donde δ_i indica la diferencia absoluta entre el pixel de la imagen original y la imagen generada, para $i = \{r, g, b\}$ indicando respectivamente los colores: rojo, verde y azul.

4. Paralelización

4.1. SPU

El cómputo más fuerte se realiza en el cálculo del fitness de un individuo. Sin embargo, el SPU no poseía las librerías necesarias para realizar esto, y aunque se buscó y trató de compilar los archivos fuentes de la librería necesario no nos fue posible la paralelización de esta función.

Sin embargo, se paralelizó el calculo de los nuevos individuos. Como una población de individuos es de tamaño 36, tomando 6 primeros padres y otros 6, se decidió paralelizar esta generación generando uno de cada 6 individuos por ver a cada SPU.

Es decir, al tomar el primer padre y se debían generar 6 otros padres aleatorios, el calculo del cruce de estos padres lo hacían los SPU. Y se repetía para los siguientes cinco primeros padres restantes.

4.2. PPU

El *PPU* se encarga de llevar el manejo principal de la ejecución. Se encarga de distribuir la carga entre los SPU, luego de cada generación se encarga del cálculo del fitness, al tener la población se encarga de tomar el mejor individuo y dibujarlo.

El peso en memoria de un individuo viene dado principalmente por el peso del *ADN* que representa, el cual puede ser estáticamente calculado de la siguiente forma:

$$Peso_{poligono} = Peso_{colores} + Peso_{vertices} = 4 + 2 * 6 = 16 \text{ floats}$$

$$Peso_{ADN} = NUM\_POLIGONOS * Peso_{poligono} = 48 * 16 = 768 \text{ floats}$$

El peso de un color ocupa 4 bytes (rojo, verde, azul, alfa). El peso de un vértice pesa 12, ya que se utilizan 6 vértices, donde cada uno indica las coordenadas X e Y del punto.

4.3. *Loop Unrolling*

Cálculo de fitness contiene dos ciclos anidados que iteran 128 veces cada uno (por el tamaño de la imagen: 128x128 pixeles), siendo el total de iteraciones $128 * 128 = 16384$.

Aprovechando las operaciones *Vector/SIMD* se redujo el total de iteraciones a $128 * (128/4) = 4096$. Logrando reducir hasta un **75 %** el número de iteraciones.

5. Resutados

5.1. Descripción de los experimentos

Se corrió el algoritmo de forma secuencial y paralelizado utilizando la configuración establecida en la sección 3.2 y limitándolo a 350 iteraciones.

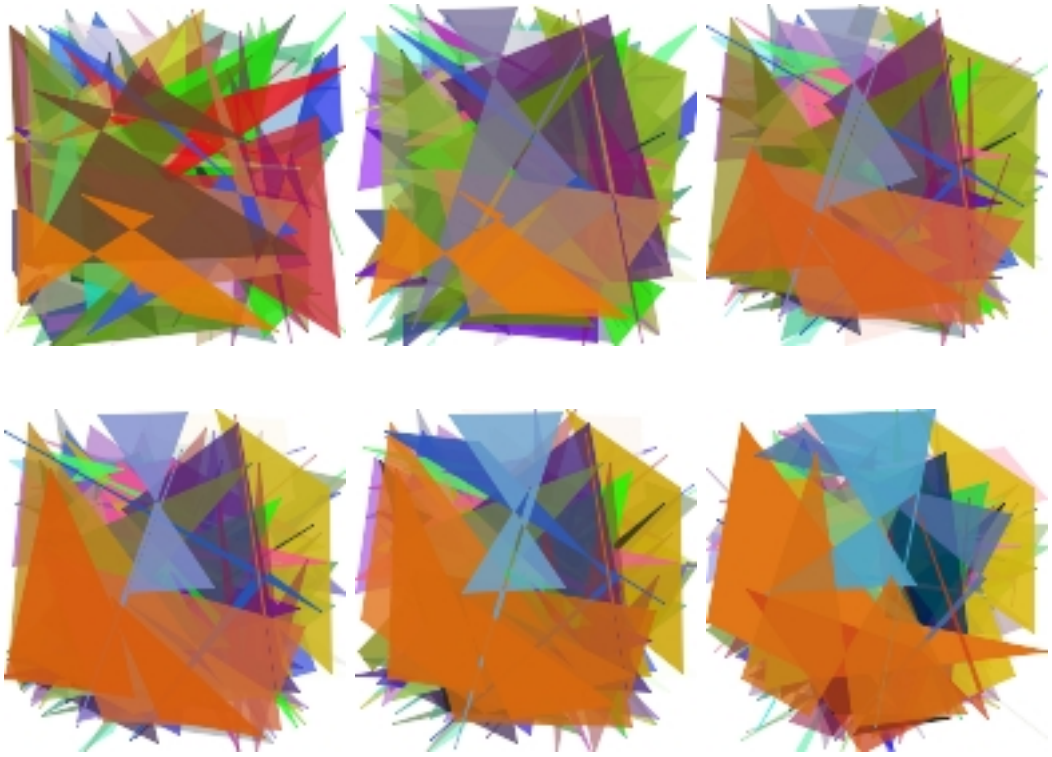
La imagen utilizada como báse para tratar de aproximar fué:

El resultado de ambos experimentos se muestra a continuación:

5.2. Corrida secuencial

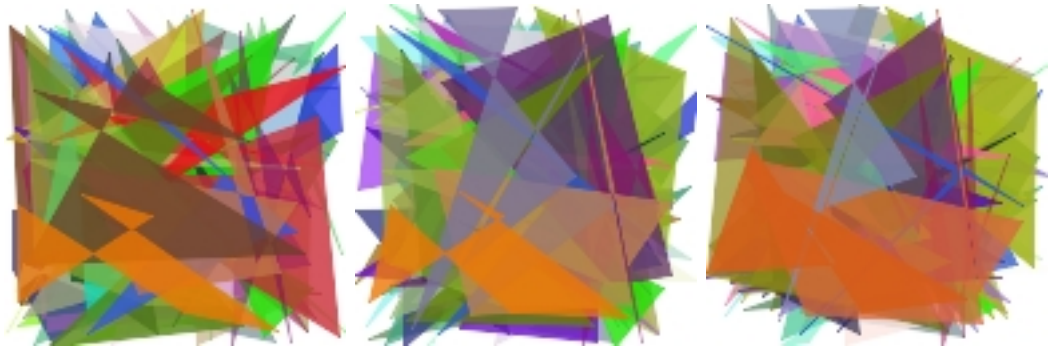
- **Tiempo de CPU:** 116min 20seg.
- **Cantidad de imágenes generadas:** 188 imágenes.

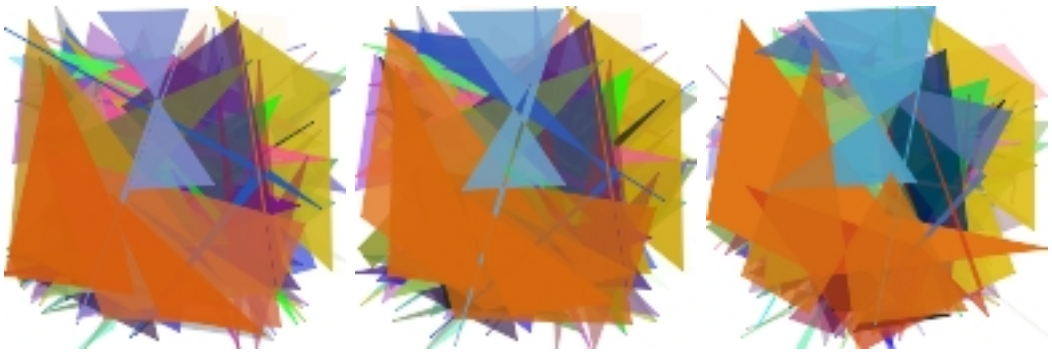
- **Máximo fitness alcanzado:** 89.850616 %



5.3. Corrida paralelizada

- **Tiempo de CPU:** 116min 20seg.
- **Cantidad de imágenes generadas:** 188 imágenes.
- **Máximo fitness alcanzado:** 89.850616 %





6. Análisis de resultados y conclusiones

Referencias

[1] Roger Alsing. Genetic programming: Evolution of mona lisa — roger alsing weblog. http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/.

[2] Jacob Seidelin. Genetic algorithms, mona lisa and javascript + canvas - nihilogic. http://blog.nihilogic.dk/2009/01/genetic-mona-lisa.html.

[3] Wikipedia. Algoritmo genético - wikipedia la enciclopedia libre. http://es.wikipedia.org/wiki/Algoritmo_genético.