IMÁGENES VIVAS E INTELIGENTES

Daniel Loaiza, Emiliano Causa, Gerardo Sanchez Olguin
Facultad de Bellas Artes - Universidad Nacional de la Plata

RESUMEN

Este trabajo aborda la aplicación de algoritmos genéticos en la creación de imágenes originales a partir del análisis y combinación de datos obtenidos de un set de imagenes inicial.

Se explicarán los conceptos de cromosoma y gen, genotipo y fenotipo, selección, crossover, y mutación. Se explicará cómo implementar estas ideas atendiendo al objetivo de crear una imagen original a partir de la descomposición en datos de imágenes existentes. Se discutirán los procesos y criterios que se usaron en este caso para descomponer a la imagen en datos manteniendo la noción perceptiva y comunicativa de la misma.

En en proceso de selección se pondrá especial atención a la selección asistida por un usuario en lugar de un algoritmo que automatice el proceso de selección.

El código completo del programa desarrollado se encuentra disponible para su descarga y uso desde *https://github.com/d7aniel/Algoritmos-Geneticos.*[1].

algoritmos genéticos,

pixel como dato

selección interactiva

1. *Para quienes descarguen el código notarán que existen algunas diferencias en cuanto a la cantidad de variables y organización del código en general, se intentará resolver cualquier tipo de duda al respecto en github.*

Algoritmo genético

Algoritmos generalmente usados para resolver problemas donde la cantidad de posibles respuestas es virtualmente infinita y encontrar la respuesta correcta mediante fuerza bruta (revisar todas las respuestas posibles) tomaría demasiado tiempo. Por ejemplo si queremos adivinar un número, la cantidad posible de respuestas es infinita ¿es cero, es diez, es cien?. Un algoritmo genético permite aproximarse al número correcto cambiando la respuesta y viendo como cada nueva respuesta se adapta a las condiciones. Por ejemplo si decimos cuarenta y nuestra respuesta está lejos y decimos cinco y nuestra respuesta está cerca sabemos que el número que queremos es menor a cuarenta(Shiffman, 2010).

El algoritmo genético funciona codificando información que representa a la respuesta posible, por ejemplo si queremos adivinar una frase, nuestro paquete de información podría ser una lista de palabras. Este paquete de información es llamado gen y va a ser heredado por una nueva criatura. Una criatura puede tener varios genes, al conjunto de genes vamos a llamar cromosoma. Un organismo puede tener uno o más cromosomas, con el objetivo de organizar la herencia de informacion maneras distintas.

La nueva criatura va a heredar la mitad de un paquete de información de una criatura y la mitad de otra mezclandolas con el objetivo de conseguir un gen superador, que se acerque más a la respuesta correcta, a este proceso llamamos **crossover**. Las dos criaturas que actúan como padres son escogidas en función de qué tan cercano es su gen al gen que corresponde a la respuesta correcta, este proceso es llamado **selección**. Por ejemplo si queremos adivinar la frase “Escribió un paper” y tenemos dos criaturas una con la lista “Escribo, Casa, Paper” y otro con la lista “Tengo un cachorro” la primera criatura tiene más posibilidades de ser escogida por que posee mayor cantidad de elementos en común con la frase a la que queremos llegar. Se llama aptitud a la variable que mide qué tan cerca se encuentra el elemento seleccionado al resultado que se quiere (Mitchell 1996).

Para expandir conceptos y conocimiento en relación a los algoritmos genéticos y su uso en el arte sugerimos como lectura el capítulo nueve de de el libro “Nature of code” de Daniel Shiffman y el artículo ”Los Algoritmos Genéticos y su Aplicación al Arte Generativo” de Emiliano Causa.

El Software

En la figura 1 se observa la interfaz del software, el mismo permite calificar una set de 30 imágenes correspondientes a imágenes de pinturas de dominio público, obtenidas del repositorio de acceso público y gratuito de NGA images, de la “National Gallery of Art, Washington”. Tras haber calificado las imágenes con cualquier criterio que el usuario haya usado, un algoritmo genético procede a mezclar las imágenes generando un nuevo set de treinta imágenes que son hijas de las imágenes previas, por lo cual han mezclado sus rasgos y empiezan a predominar los rasgos que el usuario encuentre más apropiados según el criterio que haya decidido tomar al momento de calificar las imágenes.

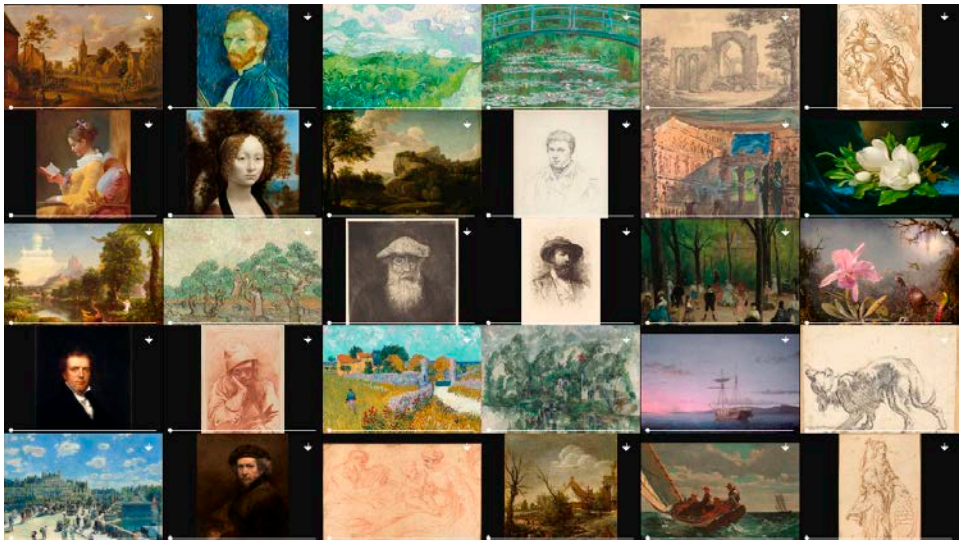


Figura 1

Interfaz inicial del software

Rasgos - Datos

Al querer fusionar dos imágenes para generar una imagen nueva, inmediatamente nos vemos frente a la problemática de qué tomar de una imagen y qué tomar de otra, es decir cuáles son los rasgos que vamos a extraer de una imagen y cuales son los rasgos que vamos a extraer de otra. La respuesta en función del objetivo del proceso, en este caso el objetivo es una creación estética, por lo tanto el criterio que deberíamos atender al separar una imagen en rasgos consiste en extraer los rasgos que sean más significativos en términos perceptivos y discursivos. Con esto queremos decir dos cosas primero que el rasgo tiene que aportar sentido a la obra en cuestión y segundo que al variar ligeramente ese parámetro el comportamiento de la imagen puede cambiar totalmente.

Para entender mejor esto vamos a atender dos ejemplo uno correcto y uno incorrecto de a qué nos referimos en este artículo con rasgos perceptivos y discursivos.

El valor rojo, verde y azul de los pixels de una imagen, por ejemplo, son valores que no constituyen rasgos perceptivos-discursivos para la imagen. Si alteramos por ejemplo un píxel de la imagen sin importar la ubicación de este es muy poco probable que la lectura que podamos hacer de la imagen se vea afectada. No podríamos decir que ese valor aporta sentido, es constitutivo de la imagen, es un rasgo que no cambia la percepción humana de la imagen drásticamente.

Por otra parte veamos el ejemplo de la paleta de color de una imagen, este es un rasgo que incorpora sentido a la imagen, dos fotografías con la misma estructura de imagen sujeto y escena, generan lecturas totalmente distintos si alteramos la iluminación, y por ende el color.

Atendiendo a estas características en los rasgos extraídos vamos a centrarnos en características que cumplan funciones discursivas, esto a sabiendas de que lo que aporta el valor discursivo son las decisiones compositivas del artista con respecto al *espacio ficticio* que desarrolla dentro de la imagen(Belinche, Ciafardo 2015), por esa razón escogemos tres

rasgos que son siempre decisiones importante en el proceso compositivo: paleta de color, clave tonal y formas o estructuras en la imagen.

Teniendo en mente los rasgos que necesitamos nos enfrentamos a una nueva problemática, sabemos que estas características existen en la imagen, pero ahora necesitamos extraerlas, procesar la imagen y convertirla en datos.

Nota para programación: *Se va a utilizar la palabra lista como sinónimo de arreglo con el objetivo de hacer la lectura más amigable a diferentes tipos de público.*

Paleta de color: Para extraer la paleta de color de una imagen vamos a definir una organización del color en la imagen y crear mapas de cómo posicionar los colores según una gama de importancia determinada por la frecuencia de uso de los colores.

Hay que aclarar a qué nos referimos con "color", cada uno de los pixeles de una imagen digital puede ser descompuesto en tres valores, uno para rojo otro para verde y uno para azul, pero podemos usar estos tres valores para determinar el color en valores de tonalidad y saturación, para nosotros la paleta de color va a estar compuesta justamente de las listas de las tonalidad y saturaciones según su uso en la imagen.

Primero tenemos que crear para cada imagen dos listas ordenadas según que cantidad de cada tonalidad y saturación se usa, para crear estas listas en primer lugar se asigna un nombre, en nuestro caso van a ser ***hue*** (tonalidad) y ***sat*** (saturación), luego vamos a crear dos arreglos que van a contener el histograma de tonalidad y saturación de la imagen. Los histogramas representan la distribución de frecuencia de cada una de las 255 posibilidades de tonalidad y color que tiene la imagen, pero no está ordenado, así que debemos ordenar los valores sin perder a que color corresponde cada cantidad.

Existe la posibilidad de que estas listas tengan valores repetidos y esto va a ocasionar problemas cuando queramos determinar el orden de uso de los colores. Por lo tanto tenemos que eliminar los duplicados pero sin eliminar los elementos y sin cambiar el orden que van a tener en un futuro estos elementos, la siguiente función recibe una lista y devuelve esa lista sin repeticiones y sin haber eliminado elementos:

```
float[] sinRepeticiones(float[] l) {
  ArrayList<Integer> rev = new ArrayList<Integer>();
  for (int i=0; i<l.length; i++) {
       if (l[i] == max(l)) {
            rev.add(i);
            break;
       }
  }
  int hack = 10000000;
  while (rev.size()>0 && hack >0) {
       for (int j=0; j<l.length; j++) {
```

```
            if (l[j] == l[rev.get(0)] && j != rev.get(0)) {
                l[j]--;
                rev.add(j);
            }
        }
        if (l[rev.get(0)]>0 && l[rev.get(0)]!=min(l) && rev.size()==1) {
            float masCercano = 1000000000;
            int indiceMasCercano = -1;
            for (int j=0; j<l.length; j++) {
                if (l[rev.get(0)]-l[j]<masCercano && l[rev.get(0)]>l[j]) {
                    masCercano = l[rev.get(0)]-l[j];
                    indiceMasCercano = j;
                }
            }
            rev.add(indiceMasCercano);
        }
        rev.remove(0);
        if (l[rev.get(0)]==0) {
            rev.clear();
        }
        hack--;
  }
  return l;
}
```

Posteriormente crearemos una versión ordenada de la lista a esta versión ordenada se le deben quitar los ceros. La siguiente función recibe un arreglo y devuelve el mismo elimina los ceros del arreglo:

```
float[] sinCeros(float[] arreglo) {
  int largo = 0;
  for (int i=0; i<arreglo.length; i++) {
        if (arreglo[i] != 0)
        largo++;
  }
  float [] arregloNuevo = new float[largo];
  for (int i=0, j=0; i<arreglo.length; i++) {
        if (arreglo[i] != 0) {
           arregloNuevo[j] = arreglo[i];
           j++;
        }
  }
  return arregloNuevo;
}
```

De esta forma tenemos dos listas(una tonalidad y otra saturación) que indican la cantidad de cada elemento y otras dos de dimensiones diferentes, por los ceros eliminados, que indican en qué posición está esa cantidad. Finalmente vamos para crear nuestras listas hue y sat usando las dimensiones de la listas ordenadas y asignamos a la posición uno el valor del color más predominante a la dos la del segundo y así sucesivamente. La siguiente función recibe la lista ordenada y desordenada y devuelve una lista ordenada que indica en qué posición se encuentra cada color según su uso.

```
int[] getArregloDeColoresOrdenado(float[] desordenado, float[]ordenado) {
  int[] arreglo = new int[ordenado.length];
  for (int c = 0; c < ordenado.length; c++) {
        for (int i = 0; i < 256; i++) {
            if (desordenado[i] == ordenado[c]) {
                arreglo[c] = i;
                break;
            }
        }
  }
  return arreglo;
}
```

Para recapitular veremos el código de todo el proceso para crear las listas hue y sat:

```
//-- asignamos nombres
int[] hue;
int[] sat;
void calcularListas(PImage img) {
  //-- creamos histogramas
  float[] histHue = new float[256];
  float[] histSat = new float[256];
  for (int i = 0; i < img.width; i++) {
        for (int j = 0; j < img.height; j++) {
        int hue = int(hue(img.get(i, j)));
        histHue[hue]++;
        int saturation = int(saturation(img.get(i, j)));
        histSat[saturation]++;
        }
  }
  //-- eliminamos las repeticiones sin eliminar los elementos
  histHue = sinRepeticiones(histHue);
  histSat = sinRepeticiones(histSat);
```

```
  //-- creamos una version ordenada de las listas
  //-- la funcion sort() devuelve una version ordenada sin alterar la
original
  float[] hueOrdenados = sort(histHue);
  float[] satOrdenados = sort(histSat);
  //-- eliminamos los valores que tienen 0 en las listas ordenadas
  hueOrdenados = sinCeros(hueOrdenados);
  satOrdenados = sinCeros(satOrdenados);
  //-- finalmente asignamos los valores de nuestras listas
  hue = getArregloDeColoresOrdenado(histHue, hueOrdenados);
  sat = getArregloDeColoresOrdenado(histSat, satOrdenados);
}
```

Ahora que tenemos dos listas (tonalidad y saturación) ordenadas para crear los colores que usa la imagen, necesitamos almacenar el lugar en el espacio que le corresponden a dichos colores para lo cual creamos un mapa que contenga la distribución de la tonalidad y otro para la saturación en la imagen, estos mapas deben contener valores normalizados donde el uno es el más utilizado y el 0 es el menos utilizado.

```
float[] mapaTonalidad =new float[ancho*alto];
float[] mapaSaturacion =new float[ancho*alto];
for (int i = 0; i < img.width; i++) {
  for (int j = 0; j < img.height; j++) {
        int indice = j*g.width+i;
        int tono = int(hue(img.get(i, j)));
        int saturacion = int(saturation(img.get(i, j)));
        mapaTonalidad[indice] = masCercado(hue, tono);
        mapaSaturacion[indice] = masCercado(sat, saturacion);
  }
}
```

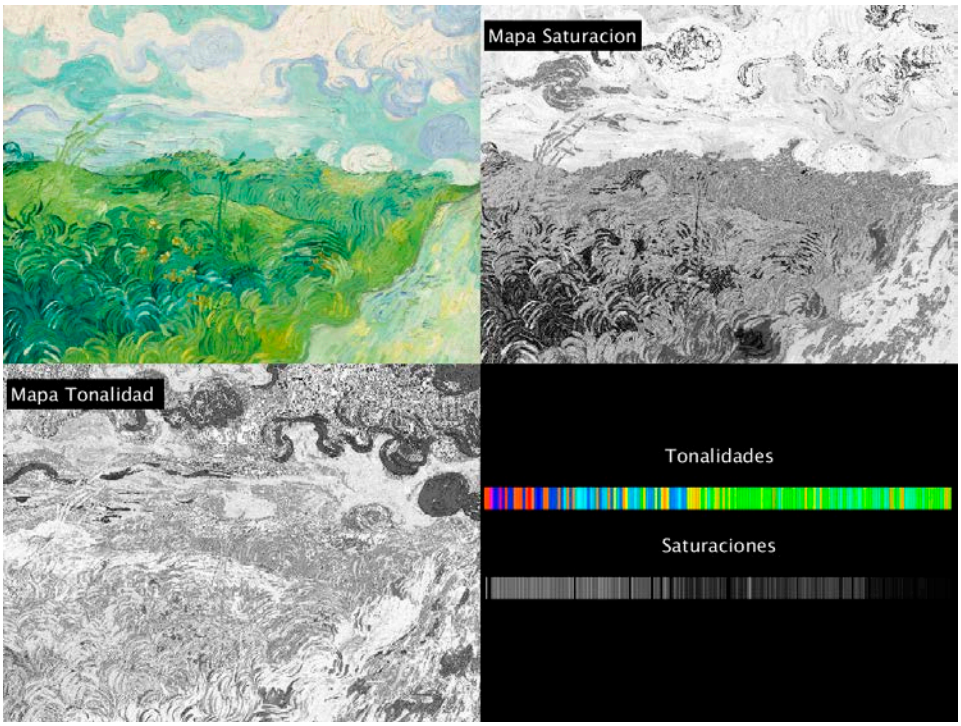
La función ***masCercano*** compara los valores de la imagen con todos los valores de la lista hasta encontrar el valor más cercano, el índice de ese color será la posición de uso de ese color, esa posición se divide para la cantidad de posiciones y así se obtiene un valor normalizado para el mapa.

De este proceso nuestro cromosoma debe tener cuatro genes, un mapa de distribución de tonalidad,uno para la saturación, una lista de colores posible en orden ascendente desde el menos usado al más usado y uno lista con las mismas características para la saturación. En el siguiente ejemplos la pintura es reconstruida construida combinando la información en sus mapas y la lista de colores disponibles.

Clave tonal: Ahora con la clave tonal vamos a proceder de una forma similar, sin embargo hay una importante diferencia, vamos a considerar que en una imagen la clave tonal está dada por la concepción con respecto al funcionamiento de la luz que tenemos como seres humanos por nuestra experiencia diaria, aun cuando en una imagen no existe la idea de

Figura 2

Ilustración los mapas de tonalidad y saturación acompañados de las listas correspondientes



luz o iluminación podemos determinar un caso hipotético de cómo está siendo iluminada la pieza, por esta razón lo que nos importa extraer al referirnos de la clave tonal es la composición de la luz en la imagen. Para lo cual vamos a tomar el píxel más iluminado de la imagen y el menos iluminado, si estos puntos se corresponden a los valores de blanco y negro sabemos que la clave tonal es una clave mayor, caso contrario es una clave menor.

Segundo vamos a determinar un valor promedio de toda la iluminación en la imagen, básicamente haciendo un promedio del brillo de todos los pixels. Si el punto promedio está a la mitad del rango de brillo es una clave media, de lo contrario será alta o baja según hacia dónde se aleje la clave.

Si el valor máximo ronda el 255 (blanco) y el mínimo ronda el 0 (negro), la clave será mayor, de otra forma será menor. Después debemos encontrar un punto promedio de los todos valores de la imagen. si el punto intermedio está alrededor de 127 será una clave media mientras si se aleja hacia el 255 o el 0 se comportara como una clave alta o baja según corresponda. Para obtener los datos necesarios:

```
float brilloMax = -1;
float brilloMin = 100000;
float brilloPromedio = 0;
for (int i = 0; i < img.width; i++) {
  for (int j = 0; j < img.height; j++) {
        float brillo = brightness(img.get(i, j));
        brilloPromedio+=brillo/255.0;
        brilloMax = brilloMax<brillo?brillo:brilloMax;
        brilloMin = brilloMin>brillo?brillo:brilloMin;
```

```
  }
}
brilloPromedio/=(img.width*img.height);
brilloPromedio*=255.0;
```

Finalmente debemos crear un mapa para conservar la posición en el rango de iluminación obtenido para cada pixel de la imagen, sin embargo este mapa va a tomar como punto máximo no el punto máximo de iluminación posible, sino el punto máximo de iluminación que exista en la imagen, de igual forma con el mínimo y el valor intermedio va a ser el promedio, estos como vemos son los valores que obtuvimos previamente.

La clave naturalmente cambia de una imagen a otra por lo tanto hay que normalizar los valores, pero esta vez vamos a hacerlo de -1 a 1, siendo 0 el punto promedio, -1 el mínimo y 1 el máximo. Y estos serán los valores para nuestro mapa de clave tonal.

```
float[] mapaClaveTonal =new float[ancho*alto];
for (int i = 0; i < img.width; i++) {
  for (int j = 0; j < img.height; j++) {
        int indice = j*g.width+i;
        int brillo = int(brightness(img.get(i, j)));
        mapaClaveTonal[indice] = brilloSegunPuntos(brillo);
  }
}
```

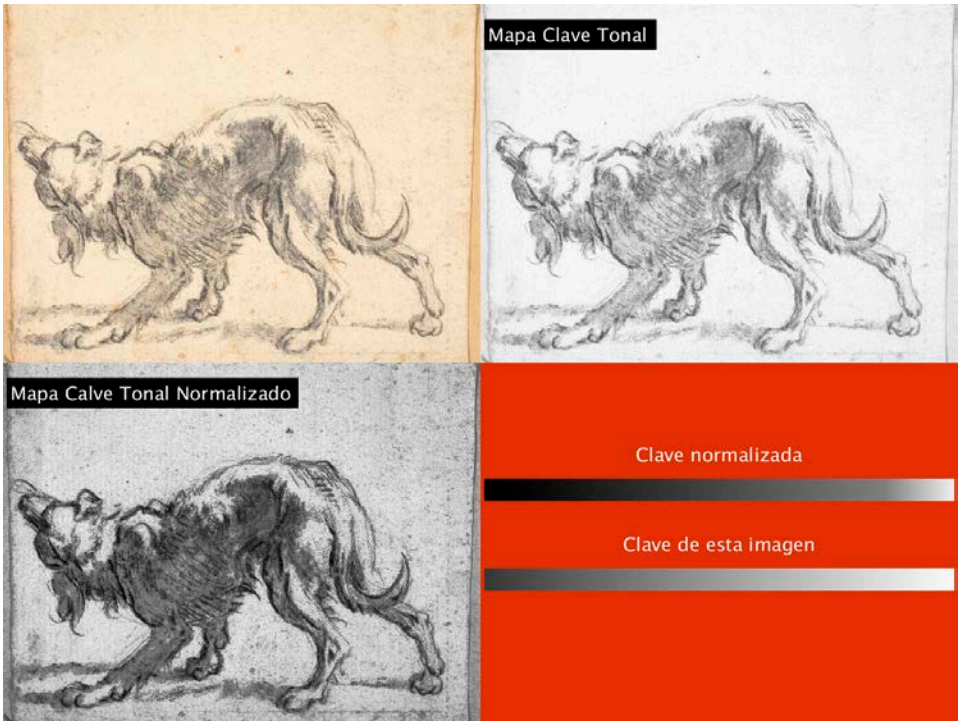
La funcion ***brilloSegunPuntos*** recibe un valor entre *brilloMin* y *brilloMax* y devuelve ese valor mapeado entre 1 y -1 tomando como punto medio el *brilloPromedio*.

```
float brilloSegunPuntos(float brillo) {
  float b = 0;
  if(brillo>briPromedio){
        //-- si el brillo es mayor al promedio
        //-- devolver un valor entre brilloPromedio y brilloMax mapeado de
0 a 1
        b = map(brillo, brilloPromedio, brilloMax, 0, 1);
  }else if(brillo<briPromedio){
        //-- si el brillo es menor al promedio
        //-- devolver un valor entre brilloPromedio y brilloMin mapeado de
0 a -1
        map(brillo, brilloPromedio, brilloMin, 0, -1);
  }
  //-- si brillo es igual al promedio devuelve 0
  return b;
}
```

El mapa de clave tonal al estar normalizado va a adquirir cualquier clave

Figura 3

Ilustración de la clave tonal de la imagen



tonal dependiendo de el máximo, mínimo y promedio de brillo que se le asigne. Pero conservará su distribución en el espacio.

Hemos agregado a nuestro cromosoma dos nuevos genes, un mapa de la distribución de la clave tonal y otro de el rango y punto intermedio del brillo que se emplaza en el mapa.

Estructuras presentes en la imagen: Al hablar de las estructuras nos referimos a aquello cuyo perímetro podemos delimitar como un configuración dentro de la más grande totalidad de la imagen. Por ejemplo el dibujo de un gato sobre el paisaje de un jardín, podemos extraer al gato como una configuración. Es fundamental comprender que la estructura no es un objeto trasladado de una realidad a un soporte artístico, sino más bien una abstracción de "algo" a la que se a moldeado para hacerla perceptible y comunicable (Belinche, Ciafardo 2015). Es precisamente el rasgo de ser algo comunicable la razón por la cual la forma resulta atractiva para ser uno de los parámetros a los cuales el algoritmo debería prestar especial atención.

Para extraer las estructuras tenemos primero que delimitar los contornos de estas formas, existen varios algoritmos que permiten hacer esto, la mayoría toma como criterio el brillo de los pixeles para diferenciarlos unos de otros y encontrar bordes. En este caso en concreto vamos a usar el algoritmo de "edge detection" de la librería de procesamiento de imagen OpenCV. El algoritmo devuelve una serie de contornos presentes en la imagen, los cuales usamos para delimitar las estructuras que existen.

```
OpenCV  opencv = new OpenCV(this, 640, 480);
//-- cambiar estos valores cambian el tamaño de las estructuras resultantes
tamBloque = 150;
```

```
constante = 3;
ArrayList<Contour> contornos;
void filtrar(PImage img) {
  opencv.loadImage(img);
  opencv.gray();
  opencv.contrast(contrast);
  if (tamBloque%2 == 0) tamBloque++;
  if (tamBloque < 3) tamBloque = 3;
  opencv.adaptiveThreshold(tamBloque, constante);
  opencv.invert();
  opencv.dilate();
  opencv.erode();
  opencv.blur(1);
  contornos = opencv.findContours(true, true);

}
```

Tomamos las tres estructuras más predominantes de la imagen y formamos con ellas otras tres imágenes de las mismas dimensiones que la imagen original pero haciendo invisibles a todos los pixels que no corresponden a la estructura dentro del contorno.

```
PImage[] getMascaras() {
  PImage[] mascaras = new PImage[3];
  PGraphics g;
  for (int i=0; i<3; i++) {
        Contour contour = contornos.get(i%contours.size());
        g = createGraphics(opencv.width, opencv.height);
        g.beginDraw();
        g.background(0);
        g.fill(255);
        g.noStroke();
        g.beginShape();
        for (PVector point : contour.getPoints()) {
           g.vertex(point.x, point.y);
        }
        g.endShape();
        for (int j=(i+1); j<3; j++) {
           g.fill(0);
           Contour contourDiff = contours.get(j%contours.size());
           g.beginShape();
           for (PVector point : contourDiff.getPoints()) {
              g.vertex(point.x, point.y);
           }
```

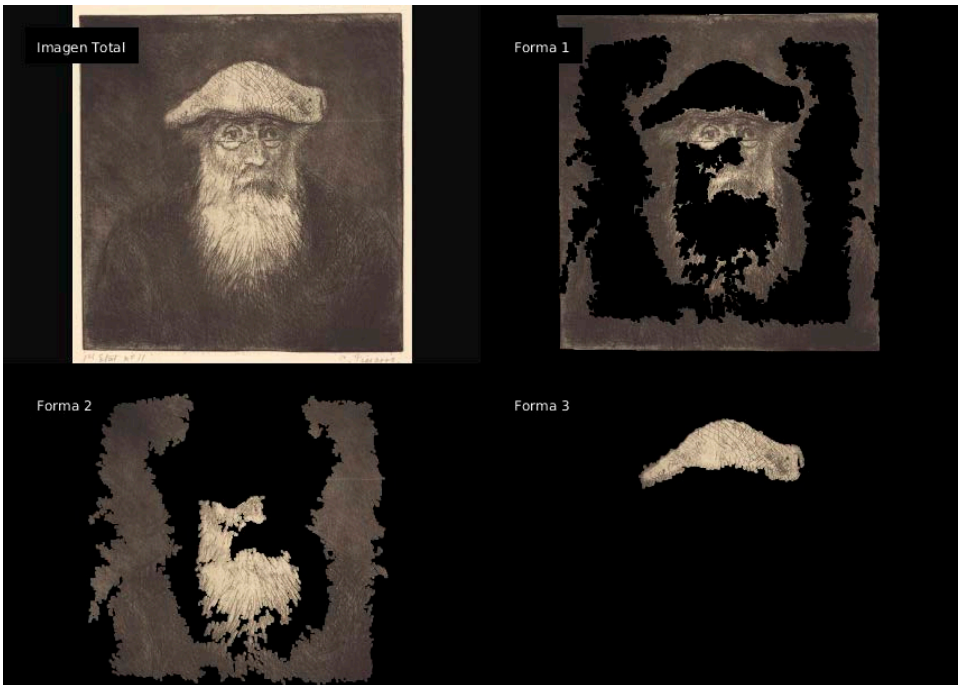
```
            g.endShape();
        }
        g.endDraw();
        mascaras[i] = g;
    }
    return mascaras;
}
```

Como vemos el tipo de dato donde guardamos la información de la transparencia es PImage, que es un tipo exclusivo de processing, para tener un código más fácil de transportar de una plataforma a otra vamos a cambiar el tipo de dato y usar un arreglo de tipo flotante como lo hicimos con los otros mapas.

```
float[] mapaAlphaMascara;
for (int i = 0; i < img.width; i++) {
  for (int j = 0; j < img.height; j++) {
        float alpha = alpha(img.get(i, j));
        mapaAlphaMascara[indice] = alpha;
  }
}
```

Figura 4

Estructuras más predominantes de la imagen



Después a cada una de estas imagenes las vamos a dividir en los mismos seis parámetros que usamos anteriormente para la paleta y la clave tonal. Para tener una mejor organización cada una de estas imágenes va a corresponderse con un cromosoma, de esta forma todo organismo va a contar con un fondo y tres formas que se superponen a ese fondo, todo organizado en cuatro cromosomas.

La información genética de nuestras imágenes: Finalmente tras haber extraído toda la información que se ha detallado, las estructuras que detectamos en la imagen van a corresponderse a un cromosoma cada una y la imagen original va a ser otro cromosoma, para un total de cuatro. Cada cromosoma va a tener los siguientes siete genes:

```
lista de tonalidad en orden de uso,
lista de saturaciones en orden de uso,
puntos de mayor, menor y promedio de iluminación (clave tonal)
mapa de la distribución de tonalidad
mapa de la distribución de la saturación
mapa de la distribución de la clave tonal
mapa de la transparencia de pixeles
```

Si bien tenemos toda la información organizada con una nomenclatura que facilita su entendimiento, esta organización complejiza las cosas al momento de mezclar los genes. Razón por la cual vamos a optar por crear para cada cromosoma un paquete que contenga todos los genes, este paquete será del tipo ArrayList que permite tener diferentes tipos de datos para cada elemento.

```
Arraylist genes =  new ArrayList();
genes.add(hue);
genes.add(sat);
//-- bri es un arreglo de tres posiciones
//-- que tiene el brillo minimo, promedio y máximo.
genes.add(bri);
genes.add(mapaTonalidad);
genes.add(mapaSaturacion);
genes.add(mapaClaveTonal);
genes.add(mapaAlphaMascara);
```

Genotipo y Fenotipo

La noción de genotipo hace referencia al conjunto de genes que están dentro de cada célula de un organismo, estos genes pueden afectar o no al mismo. Cuando lo hacen van a tomar forma como un rasgo físico o mental de ese organismo, eso es el fenotipo, podemos entonces definir al genotipo como el conjunto de *genes* de un organismo y el fenotipo como el conjunto de *rasgos* de un organismo (Mitchell 1996).

Es importante tener en cuenta estos dos conceptos, ya que permiten repensar nuevos comportamientos en los sistemas. En nuestro ejemplo podemos decir que el fenotipo es la imagen en sí y el genotipo es el cromosoma que creamos, en este programa no tenemos ningún rasgo que dependa de condiciones externas, pero es fundamental entender que el cromosoma y

la imagen no tienen porqué tener el mismo resultado siempre, podría haber un gen que dependiendo de ciertas circunstancias altere o no a la imagen.

Antes de entrar en el proceso del algoritmo genético vamos a repasar un poco la estructura de nuestro genotipo y hablar de la población. Cada una de nuestras imágenes va a ser conceptualizada como un organismo que posee cuatro cromosomas, cada uno con siete genes. Para poder mezclarlos entre sí estos deben ser parte de una grupo, Por lo tanto crearemos algo para contener y aplicar funciones específicas a este grupo. Este objeto se llamará población y en esencia va a ser un objeto que tiene un arreglo de Organismos dentro de sí, por facilidad semántica también vamos a llamar a este arreglo población.

Algoritmo Genético

Finalmente se va a mostrar el algoritmo genético como tal, la versión más simple del mismo está compuesto de tres etapas que vamos a analizar: selección, crossover y mutación.

Selección

Este proceso de selección consiste en tomar dos organismos que van a ser los padres de una nueva criatura, para seleccionarlos sus cromosomas tienen que haber sido puestos a prueba contra el "mundo" en el que estos organismos viven, entre más apto es el fenotipo de ese o esos cromosomas, mayor probabilidades tiene el mismo para ser escogido para la reproducción (Shiffman 2012).

Rápidamente notaremos que aparece una nueva variable, la aptitud de un cromosoma, usualmente en un algoritmo genético, para evaluar dicha aptitud vamos a hacer que nuestro organismo realice una tarea, intente resolver un problema o algo similar y evaluaremos los resultados que obtuvo, entre más se aproxime al resultado deseado mejor será su aptitud y mayores probabilidades tendrá de reproducirse.

En nuestro caso lo que estamos buscando es una construcción estética original y conceptualizar este objetivo como un problema en programación resulta muy complejo, quizá imposible. Sin embargo existe la posibilidad de que el usuario califique las producciones del algoritmo a partir de su criterio estético y la aptitud de cada organismo se modifique en función de ese criterio, a esto llamaremos selección interactiva (Sims 1991).

La forma de interacción que hemos escogido para el programa es una grilla con las imágenes, bajo cada imagen hay un slider que permite calificar a cada una según el criterio estético de la persona que interactúa. La aptitud de ese organismo es determinada en función de dicha calificación.

Ahora usamos la aptitud de cada organismo para determinar la probabilidad de que el mismo sea escogido el momento de reproducirse. Aquí nos enfrentamos a una problemática clásica, tomar objetos al azar tomando en cuenta diferentes probabilidades para los objetos. Existen

Figura 5

GUI del software

varias formas de enfrentar esta problemática. Sin embargo algunas requieren tener una piscina gigantesca de objetos, otras necesitan calcular la probabilidad de cada objeto de tal modo que sumen un total de cien por ciento. El método que proponemos es tanto sencillo como eficiente, tomamos un índice al azar y preguntamos si la aptitud del organismo al que le pertenece es mayor que una aptitud al azar, evidentemente notamos como entre mayor sea la aptitud mayor probabilidad de que la condición se cumpla.

La siguiente función devuelve un Organismo basado en su aptitud desde un arreglo llamado población.

```
Organismo getEltern() {
  int hack = 0;
  Organismo eltern = null;
  while (true && hack<10000) {
        int indice = floor(random(poblacion.length));
        eltern = poblacion[indice];
        float r = random(aptitudDelMejor);
        if (r<eltern.aptitud) {
        return eltern;
        }
        hack++;
  }
  return eltern;
}
```

Crossover

Tras haber seleccionado los padres de una criatura viene el proceso de crossover, el cual consiste en tomar genes de los padres y mezclarlos en una nueva criatura. Es importante recalcar que en el proceso de crossover no existe una modificación de los genes, los genes se mezclan exactamente igual a como fueron extraídos, las imágenes resultantes tiene los mismos genes que las originales pero en órdenes y distribuciones distintas. No se inventa información sino que se usa la existente para concebir nuevos resultados.

Para la mezcla de dos criaturas lo que debemos hacer es tomar cada uno de los cromosomas y combinar sus genes tomando al azar algunos genes de una criatura y algunos de la otra, para esto vamos a crear una función que reciba el arraylist con los genes de cada cromosoma y devuelve un arraylist con los genes mezclados.

```
ArrayList mezclar(ArrayList g1, ArrayList g2) {
  ArrayList ng = new ArrayList();
  for (int i=0; i<g1.size(); i++) {
        if (forzado[i] == 0) {
            ng.add(g1.get(i));
        } else {
            ng.add(g2.get(i));
        }
  }
  return ng;

}
```

Tenemos que repetir esto para cada cromosoma, esto se realizará dentro del constructor del organismo, en la siguiente tabla podemos ver la implementacion del metodo para realizar el crossover:

```
//--- seleccionamos dos organismos según su aptitud
Organismo A = getEltern();
Organismo B = getEltern();
//-- creamos una nueva criatura

Organismo criatura = new Organismo(A, B);
```

Mutación

La mutación permite que cuando el algoritmo haya agotado todas las posibilidades que sus genes le permiten, si aun así no ha encontrado una solución esperada entonces crea nuevos genes que pueden acercarse mejor al resultado deseado.

En nuestra organización, de 30 organismos con 4 cromosomas cada uno y cada cromosoma con 6 genes que se intercambian entre organismos tenemos una cantidad de probabilidades cuyo número escrito tiene 36 cifras. Son muchas probabilidades, si a esto le sumamos que nuestro proceso de selección está hecho por un ser humano, los tiempos de cada época pueden ser varios minutos. Jamás se completan las épocas suficientes para que la mutación sea realmente necesaria.

Dicho esto, nos parece importante al menos dar una explicación sobre cómo se aplica de forma tradicional la mutación a algoritmos genéticos y en qué situaciones y bajo qué criterios sería interesante la aplicación de este proceso.

El más simple de los procesos de mutación consiste en cambiar totalmente al azar algunos bits de un gen (Mitchell 1996). Este proceso se hará con una probabilidad determinada de antemano, la probabilidad de mutación suele ser baja ya que entre más alta es, más ineficiente es el algoritmo, sin embargo en sistemas altamente caóticos, es decir con muchos elementos, encontrar los elementos adecuados requiere de niveles de mutación ligeramente más altos ya que esta es la única forma de incorporar a la piscina de selección los genes que no estaban en juego y así encontrar la mejor solución.

En casos para generar imágenes bajo los parámetros que trabajamos, resultaría interesante variar sobre las listas que determinan la paleta de color y la clave tonal. Tambien se podrian crear formas generativamente. Los mapas de color y clave tonal son dados por la estructura de las pinturas originales y en este desarrollo fue un elemento que se quiso mantener desde un principio, esta es la única razón por la cual no se modifican los mapas para la mutación, sin embargo es una posibilidad y un posible desarrollo futuro.

Resultados

Finalmente tras este proceso obtenemos la mezcla de organismos, Aquí algunos ejemplos:

En las figuras 6a-d observamos como de las dos imágenes iniciales (las dos en la parte superior de la ilustración) generamos una tercera imagen hecha a partir de la mezcla de los parámetros que hemos cubierto previamente,

Nota: A los lados de la imagen se pueden ver qué genes tomó de cada uno de los cromosomas de las imágenes correspondientes.

Los ejemplo anteriores se obtiene tan solo después de una generación, es decir tiene genes pertenecientes a dos imágenes iniciales, sin embargo podemos continuar mezclando genes incorporando rasgos de varias imágenes base y generando composiciones mucho más complejas que divergen en mayor grado de las imágenes base.

Figura 6a

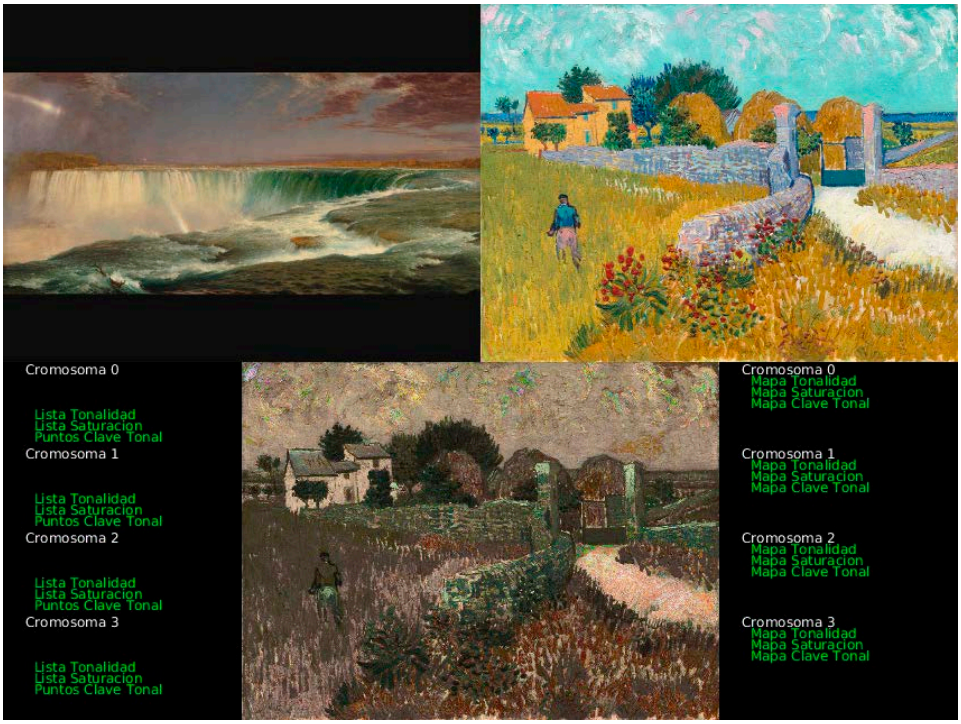


Figura 6b

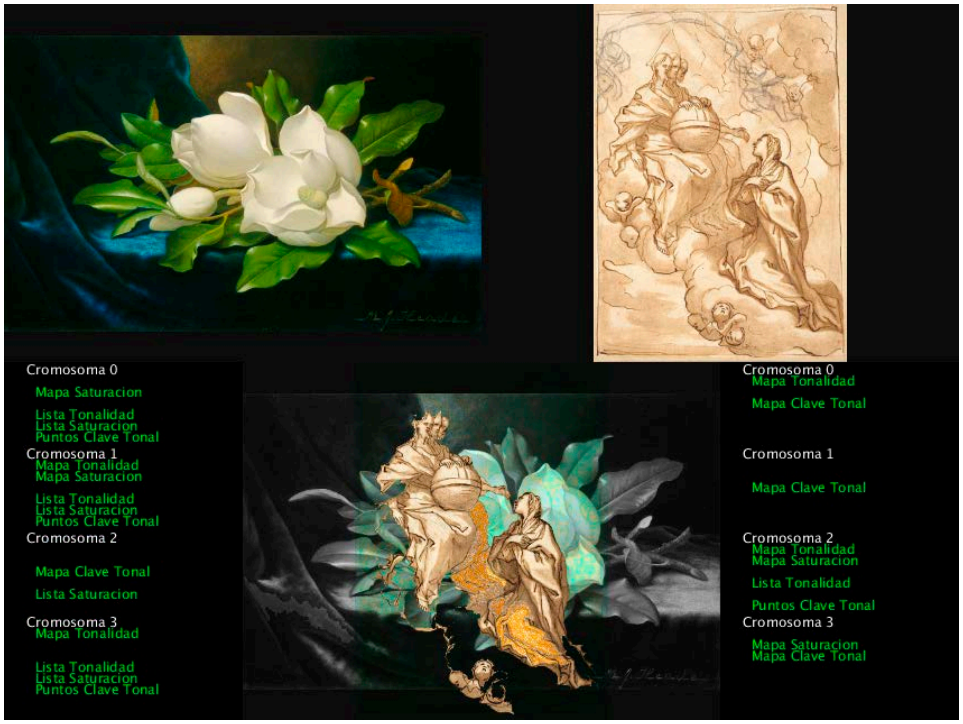


Figura 6c

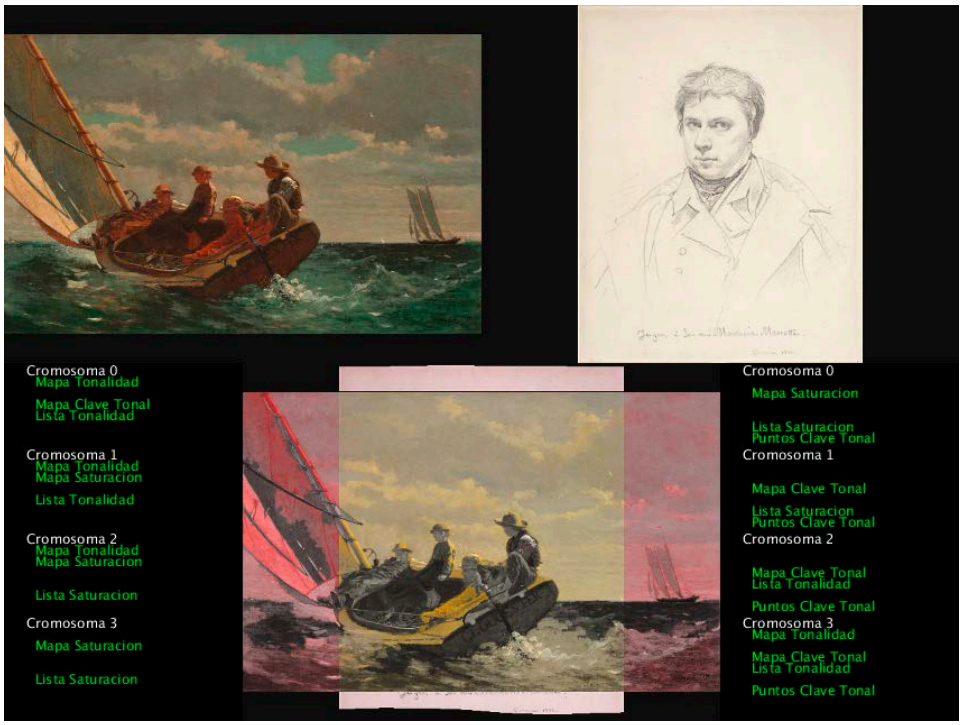


Figura 6d

Figura 7a

Figura 7b

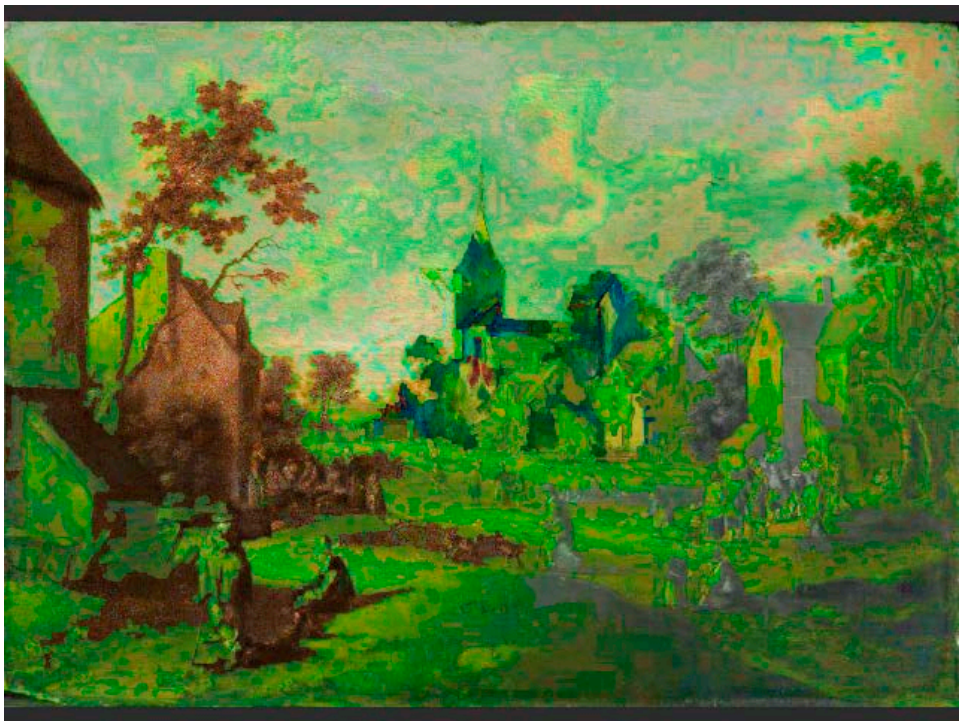


Figura 7c

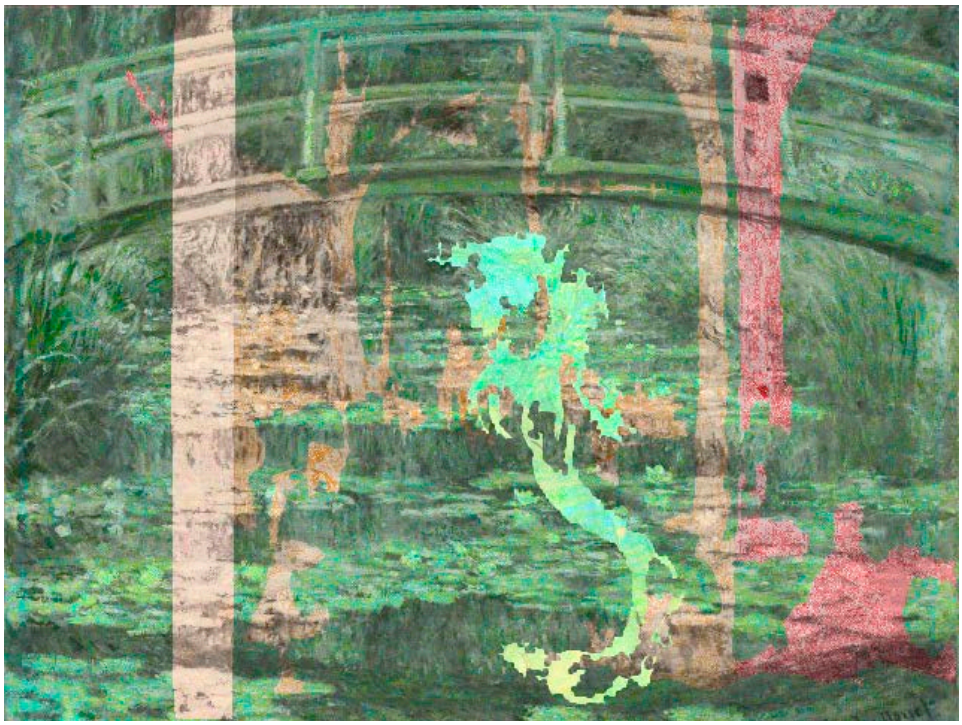
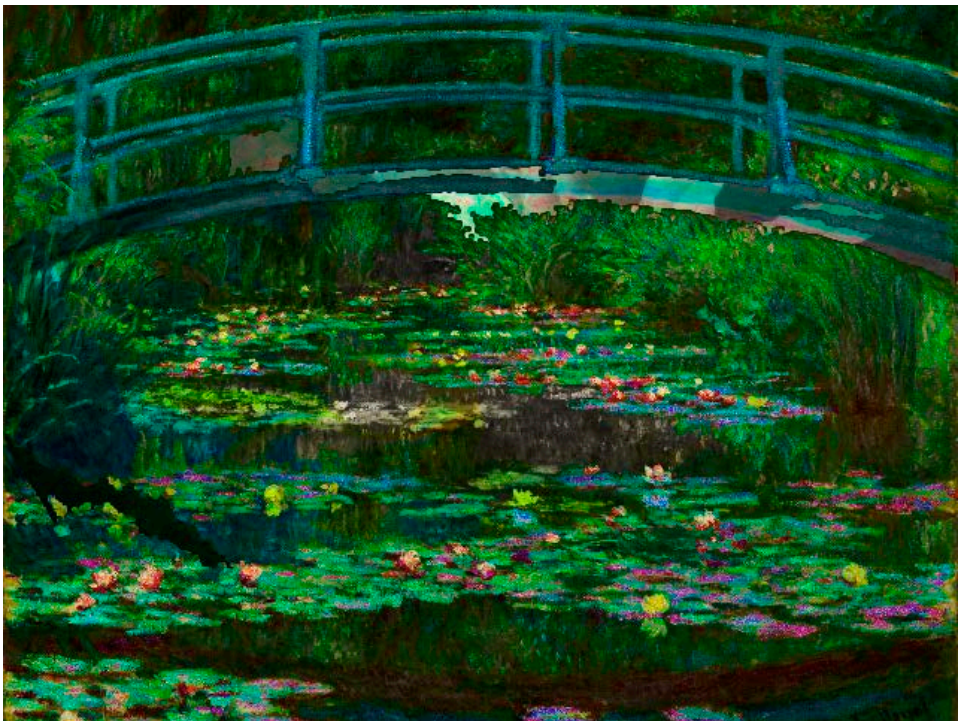


Figura 7d

Figura 7e

Figura 7f



Conclusiones y cómo seguir

Sobre los algoritmos genéticos en el arte: Los algoritmos genéticos son parte de la familia de algoritmos de inteligencia artificial, por lo tanto su potencialidad tanto pedagógica como productiva en el arte y otras área es muy grande y a un costo muy bajo, hablando del hardware y conocimiento necesario para iniciarse, los requerimientos para empezar a ensayar ideas propias es mínimo, notablemente inferior a otras técnicas de inteligencia artificial como las redes neuronales.

Cabe destacar que cuando nos referimos al arte nos referimos a todo tipo de manifestaciones artísticas, en este caso particular hemos realizado aplicaciones en imágenes digitales, pero podría realizarse en sonido o en producciones audiovisuales.

Sobre el uso de los rasgos de una imagen como datos: tener como objeto inicial de trabajo imágenes implica una problemática totalmente distinta al abordar el trabajo, es casi lo contrario a lo que se espera de un proceso de composición de imagen pues el proceso implica descomponer la "cosa" en los segmentos que van a ser reconstituidos en algo nuevo y es sumamente importante que estos segmentos cumplan funciones perceptivas y comunicativas, de lo contrario lo único que quedaría es el azar. Por esta razón al trabajar con información en arte es muy importante pensar al dato como una materialidad, al final lo que un algoritmo genético comparte son datos.

Sobre este trabajo y su continuación: mi mayor expectativa con este trabajo es que sirva de iniciación en el camino para alguien hacia su primer algoritmo de inteligencia artificial o su primer trabajo descomponiendo una imagen en datos maleables desde la percepción y el gusto. Naturalmente estos dos temas son mucho más amplios que lo que se ha cubierto en este trabajo y son temáticas que espero continuar tratar en el futuro, por ejemplo la problemática de crear un código mucho más eficiente que permita hacer crear el mismo resultado pero por ejemplo en una reproducción de video en tiempo real, las redes neuronales han demostrado en los últimos años un crecimiento inmenso sobre todo en términos de velocidad y eficiencia.

Bibliografía

- Mitchell, Melanie (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press. ISBN 9780585030944.
- Causa, Emiliano (2011) Los Algoritmos Genéticos y su Aplicación al Arte Generativo.
- Shiffman,Daniel (2012) The nature of code
- https://natureofcode.com/book/chapter-9-the-evolution-of-code/
- Mariel Ciafardo, Daniel Belinche (2015) El espacio y el arte.
- Sean, Luke (2013) Essentials of Metaheuristics.

• Sims, Karl (1991) “Artificial Evolution for Computer Graphics”, Computer Graphics, Vol.25

Imágenes

Las imágenes usadas tanto para el desarrollo del programa como para este trabajo fueron todas descargadas del repositorio de acceso público y gratuito de NGA images, de la “National Gallery of Art, Washington”.