UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Algoritmos generativos aplicados en sistemas interactivos

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

José Armando Rodriguez Ibarra

DIRECTOR DE TESIS

ING. Luis Sergio Valencia Castro

Ciudad Universitaria, Cd. Mx., 2017

Agradecimientos

Gracias a Dios y a mi Familia

Contenido

Introducción

El uso de los algoritmos generativos en aplicaciones interactivas representan una manera alternativa de manipulación e interpretación de la información visual usualmente desplegada en monitores, proyectores y demás dispositivos periféricos encargados de representar gráficamente los datos de salida de la computadora.

Se define informalmente a los algoritmos generativos como un medio por el cual se puede representar el comportamiento orgánico de la naturaleza en sistemas computacionales, por otro lado un sistema generativo encontrado en la naturaleza es aquel que no se puede construir mediante planos, materiales y herramientas, por consiguiente, un algoritmo generativo emula el comportamiento de la naturaleza mientras que un sistema generativo hace alusión directa al propio comportamiento. Un sistema generativo describe el comportamiento que presenta una flor o un árbol durante su ciclo de vida, mientras que por medio de un algoritmo generativo podemos simular este proceso lo más cercano a la realidad. Como es evidente, es imposible materializar artificialmente los procesos naturales o sistemas generativos, pero por medio de los algoritmos generativos se pueden simular; y en cuyo caso a manera de analogía, las semillas son las condiciones iniciales y la lógica. La característica principal de estos algoritmos es que son emergentes pero matemáticamente y lógicamente sustentados. Emular lo orgánico por medio de vías artificiales.

Actualmente los algoritmos generativos son utilizados en la generación de simulaciones y gráficos por computadora que posteriormente son implementados en productos filmográficos, videojuegos y en algunos casos instalaciones multimedia y obras artísticas. Otro rubro en dónde son implementados este tipo de algoritmos es en el diseño conceptual de estructuras arquitectónicas, las cuales son generadas y estudiadas para posteriormente decidir la viabilidad de desarrollo. Otro caso muy similar es en el diseño industrial que de igual forma genera formas a partir de algoritmos generativos; aprovechando sus cualidades para desarrollar productos únicos, creativos e innovadores.

El primer capítulo de este trabajo es un acercamiento a los conceptos básicos alrededor de los algoritmos generativos así como las formas con las que se puede obtener dicho comportamiento. Por otro lado se expondrán de igual manera los conceptos principales referentes a la interacción y los elementos que la componen.

En los capítulos dos y tres se profundizará en la explicación de algunas de las técnicas y herramientas más relevantes para generar el comportamiento generativo y el desarrollo de experiencias interactivas. El cuarto capítulo expone el proceso de trabajo empleado para el desarrollo de aplicaciones interactivas con un enfoque generativo y finalmente en el último capítulo de este trabajo se mostrarán las aplicaciones desarrolladas principalmente en lenguaje Processing así como los resultados gráficos obtenidos.

Alcance

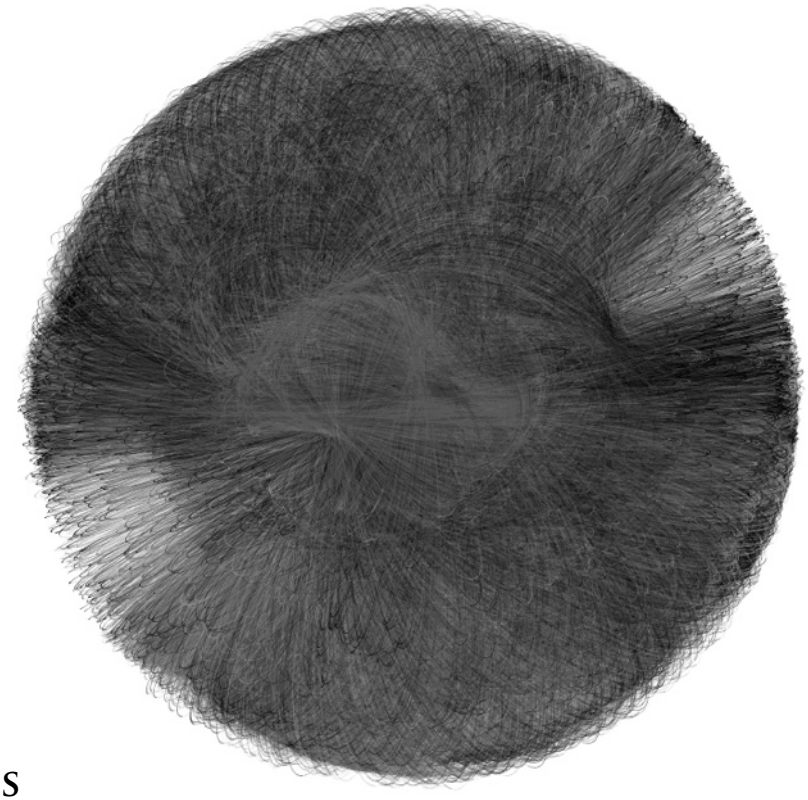
En este trabajo de tesis se usarán tecnologías existentes para apoyarse por el lado de la interacción, es decir las interacciones mas simples y elementales que se pueden conceptualizar con esta clase de sistemas, es con periféricos de entrada como el mouse y el teclado; y dispositivos de salida como el monitor. Aunque los proyectos actuales integran mas allá de los periféricos mencionados, es decir se implementan dispositivos como: acelerómetro, giroscopio, termostatos, fotosensores, pantallas táctiles, arreglos multipantalla, proyectores, controladores DMX, audio, por mencionar algunos.

Además se buscará hacer uso de tecnologías como el “Kinect” de Microsoft y proyecciones de video apoyada por técnicas multimedia como el Video Mapping las cuales brindan una experiencia para el usuario; así como el uso de Arduino para resolver obstáculos que se presentaron. Cabe destacar que la finalidad de mencionar estas tecnologías es que existe un amplio repositorio de recursos, documentación e información así como bibliotecas especializas para uso de estos dispositivos y técnicas. Por lo que se retomará esta información disponible para poder alcanzar los objetivos propuestos.

Objetivos

El objetivo en este trabajo es demostrar el uso de los algoritmos generativos en aplicaciones interactivas, las cuales representan una manera alternativa de manipulación e interpretación de la información visual. Se pretende implementar estos algoritmos en sistemas capaces de reaccionar a estímulos generado el usuario y se tenga por respuesta un comportamiento gráfico con el que se pueda interactuar en tiempo real. Se buscará que estos sistemas sean capaces de reaccionar en el menor tiempo posible, sin disminuir la calidad y fluidez de la visualización, con el fin de evitar romper el vínculo de interacción entre el usuario y el sistema.

Capítulo 1



Conceptos sobre los Algoritmos Generativos e Interacción

Este capítulo busca esclarecer los términos y definiciones básicas que se encuentran alrededor de los algoritmos generativos con enfoque a la interacción. Primeramente buscando una definición formal para los sistemas generativos y subsecuentemente los modelos empleados para obtener dicho comportamiento. De esta manera podremos ubicar un hilo conductor que nos permita dirigirnos hacia los elementos esenciales que componen a la interacción y posteriormente establecer un contexto en el cual convergen ambos campos.

1.1 Conceptos sobre algoritmos generativos

"Un modelo generativo desde el punto de vista de la probabilidad y estadística, es aquel que permite la generación aleatoria de valores a datos observables, generalmente proporcionando parámetros denominados ocultos. Es decir, dichos parámetros o variables ocultas son aquellas que no se observan directamente sino que son inferidas a través de un modelo matemático a partir de otras variables que sí son posibles de observar"[1].

Así un modelo generativo se puede utilizar por ejemplo, para simular valores de cualquier variable en el modelo. Ejemplos de modelos generativos incluyen:

- Modelo de mezcla Gaussiana.
- Modelos ocultos de Markov.
- Probabilidad de gramáticas de contexto libre.
- Máquinas de aprendizaje.
- Asignación Dirichlet latente.
- Máquina de Boltzmann restringida.

En el ámbito de la computación gráfica, los algoritmos generativos se convierten en un interesante objeto de estudio dado que converge con el campo de la geometría, dando así origen a la geometría algorítmica la cual permite resolver mediante algoritmos problemas geométricos. Algunos autores sobre todo en la arquitectura también denominan al punto de intersección entre la geometría algorítmica y la programación como Algoritmos Generativos. Por otro lado Celestino Soddu definió en 1992 a los modelos generativos como: "Procesos morfogenéticos que utilizan algoritmos estructurados como sistemas no lineales para obtener interminablemente resultados únicos e irrepetibles tal y como sucede en la naturaleza"[2].

En otras palabras un modelo generativo aplicado al cómputo gráfico, se debe conceptualizar como un proceso en el cual se realiza la generación aleatoria de valores los cuales fungen como los datos que tomará la o las variable ocultas, las cuales intervendrán en un proceso geométrico que busca emular la evolución de patrones y comportamientos únicos e irrepetibles encontrados en la naturaleza. Dichos valores aleatorios son generados mediante algoritmos genéticos, funciones pseudo aleatorias como la que fue desarrollada por el investigador estadounidense Kenneth H.

Perlin (Perlin Noise), APIs que recogen datos meteorológicos (realmente aleatorios); o el caso expuesto en esta tesis, generados a partir de la interacción de los usuarios.

1.1.1 Aleatorio o Pseudo-Aleatorio

Al menos todos los lenguajes de programación modernos contienen en su repertorio de funciones una equivalente a **rand()** de C, pero generar valores realmente aleatorios es una tarea teóricamente imposible para las las computadores actuales. "La aleatoriedad generada por computadora siempre es pseudo-aleatoria, usualmente se utiliza una ecuación matemática la cual incluye dentro de sus parámetros al reloj interno de la máquina, con la finalidad de asegurar que los resultados no sean constantes"[3].

La pseudo-aleatoriedad es un problema de gran envergadura en las ciencias de la computación, principalmente por que es involucrada en la encriptación de datos. Actualmente el instituto de Ciencias de la Computación de la Universidad de Dublin, posee el proyecto *random.org*, el cual aloja una API que devuelve "verdaderos" valores aleatorios basados en el ruido atmosférico.

1.1.2 Variabilidad natural

La función matemática Ruido Perlin, es una clase de ruido gradiente desarrollada por Ken H. Perlin en 1983. El desarrollo del Ruido de Perlin, ha permitido a expertos en el cómputo gráfico representar de una mejor manera la complejidad del comportamiento de la naturaleza en efectos visuales.

"Dicha función utiliza la interpolación entre un gran número de gradientes precalculados de vectores que construyen un valor que varía pseudo-aleatoriamente. El comportamiento es similar al del ruido blanco. Parte del resultado del trabajo de Ken Perlin, fue la implementación de dicho algoritmo para generar las texturas de un sin número de películas y producciones audiovisuales, algunas de ellas ganadoras de premios destacables en el ámbito de mejores efectos especiales y mejor logro técnico. El Ruido Perlin es ampliamente usado para simulaciones como el fuego, humo, nubes y en general todo tipo de fenómenos que requieran aleatoriedad sin perder continuidad"[4].

Finalmente, los Algoritmos Generativos permiten crear tecnología que simula el comportamiento evolutivo de la naturaleza aplicado en la exploración de nuevas formas y soluciones geométricas

que involucran al proceso creativo. Muchas de las aplicaciones actualmente desarrolladas en este ámbito, han logrado crear notables resultados en la generación de imágenes, video, sonido, modelos arquitectónicos, animación, por mencionar algunos. Estos últimos dan pie al siguiente concepto el cual involucra la experiencia de usuario con respecto al sistema con el cual interactúa.

1.2 Conceptos sobre interacción

Actualmente es difícil definir formalmente al conjunto de conceptos que forma el área de la interacción humano-computadora. En términos generales, se define como el estudio encargado de analizar el intercambio de información mediante software entre los usuarios y las computadoras. También es posible ligar esta definición con el diseño, evaluación e implementación de los aparatos tecnológicos interactivos. El objetivo es encontrar la manera mas eficiente de intercambiar información, minimizando errores; maximizando la experiencia de usuario y el uso intuitivo. Es decir, las tareas que involucran a personas y computadoras deben ser mayormente productivas.

1.2.1 Ingeniería de sistemas interactivos

La ingeniería de sistemas interactivos es considerada como un conjunto interdisciplinario en el cual converge la ingeniería en computación, diseño interactivo, desarrollo de software, etnografía, psicología, artes y otros factores de usabilidad involucrados. Tradicionalmente en la interacción humano-computadora se involucran tres áreas de conocimiento: diseño, experiencia de usuario e ingeniería. La ingeniería de sistemas interactivos está ligada directamente a está última área de conocimiento.

1.2.2 Interfaz

En el ámbito de la informática, la interfaz es conocida como el vínculo funcional entre sistemas, dispositivos y usuarios. La definición de interfaz puede ser perceptible desde tres diferentes enfoques: “Cómo instrumento, es decir, una extensión del cuerpo (mouse, teclado, audífonos, etc), como superficie, en otras palabras; para comunicar visualmente (formas, texturas, colores, instrucciones, etc) y como espacio (lugar de interacción)”[5].

Capítulo 2

Herramientas para generar aleatoriedad

Este capítulo expone los diferentes métodos con los cuales es posible obtener comportamientos generativos y un breve acercamiento las posibles aplicaciones que se pueden lograr empleando esta clase de sistemas.

2.1 Distribución normal de los números aleatorios

Existen fenómenos encontrados en la vida cotidiana, por ejemplo la estatura de los habitantes de la Ciudad de México, la cual no está distribuida de una manera uniforme, es decir hay una menor cantidad de personas que son mas altas o mas bajas que el promedio de estatura. En otras palabras, los valores que se encuentras al rededor del promedio son aquellos a los que nos referimos como la distribución normal o distribución de Gauss.

Cómo resultado de esto se genera una curva en forma de campana. Dicha curva sigue el comportamiento de una función matemática la cual define la probabilidad de un valor dado en función de la media y la desviación estándar. Se tiene una desviación estándar baja cuando los valores están altamente concentrados alrededor del promedio, por otro lado se tiene una desviación estándar alta cuando dicho valores se encuentran diseminados al rededor del promedio.

Para producir un número aleatorio con distribución normal en el lenguaje de programación Processing del cual se describirá posteriormente y con el que se realizó la mayor parte del desarrollo en este trabajo, se expresa de la siguiente manera:

```
void draw()
  {
   float num = (float) generator.nextGaussian();
  }
```

2.2 Perlin Noise

Perlin Noise tienen una apariencia mucho mas orgánica por que produce una secuencia de números pseudo-aleatorios naturalmente ordenados. Las siguientes figuras muestran el comportamiento de un algoritmo, el cual dibuja un espiral. El espiral es formado por medio de líneas rectas cuyos puntos extremos están relacionados con un radio con origen en el centro de la imagen. La magnitud de dicho radio es afectado en la primera imagen gracias a una implementación del Ruido Perlin (figura 2.1), mientras que en el segundo caso únicamente el radio es afectado por simples números aleatorios (figura 2.2). Cabe mencionar que ambas imágenes fueron generadas a partir de Processing, el cual cuenta con la función de Ruido de Perlin implementada: la función **noise( )**. Esta función toma entre uno y tres valores, dependiendo de la cantidad de dimensiones en las que se implementará[6].

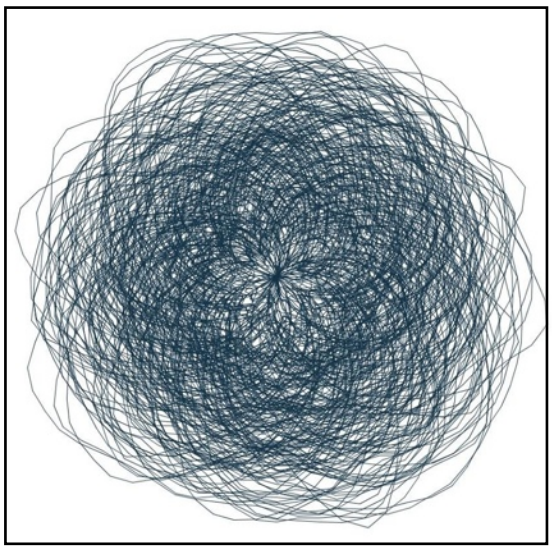


Figura 2.1

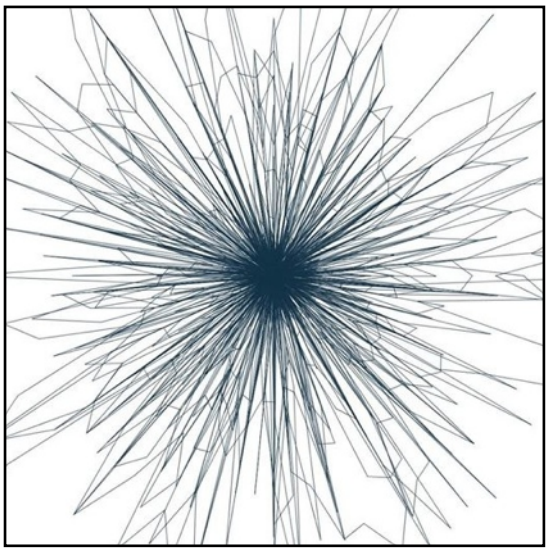


Figura 2.2

El fragmento del siguiente código muestra el funcionamiento del algoritmo previamente mencionado implementando la función noise.

```
float radioNoise = random(11000);
for (float ang = 0; ang <= 90000; ang += 5) {

  //El segundo parámetro indica la extensión del espiral

  float rad = radians(ang);

  //La variable radioNoise será evaluada por la función Noise (Perlin Noise)
  //Es aumentada cada loop para que sea evaluada en un punto distinto en cada iteración

  float newrad = radio + (noise(radioNoise) * 280) - 100 ;
  //280 representa el radio máximo del espiral y -100 su valor mínimo
  x = centX + (newrad * cos(rad));
  y = centY + (newrad * sin(rad));
  if (lastx > -999) {

    //La línea es dibujada

    line(x, y, lastx, lasty);
  }
  lastx = x;
  lasty = y;
  radioNoise += 0.05;
}
```

Por otro lado el siguiente fragmento muestra la implementación con simples números aleatorios solo con ligeras modificaciones al fragmento de código anterior.

```
float radioNoise = random(11);
for (float ang = 0; ang <= 9000; ang += 5) {
  float rad = radians(ang);
  float newrad = radio + (random(radioNoise)  *  4)  -  100 ;
```

Como salida de la función **perlin( )** siempre se devuelven valores entre 0 y 1. Implementando esta función en una sola dimensión en una secuencia lineal a través del tiempo se obtienen valores como los siguientes:

Tiempo	Valor
0	0
1	0
2	0
3	0
4	0

El control de la tasa de variación en la curva se puede controlar mediante una variable que se incrementa con cada ciclo, es decir un incremento menor en la variable en cada ciclo genera cambios menores en la variación de la curva y viceversa. El siguiente algoritmo expresa la variabilidad de la curva mediante el control de dicha variable.

Figura 2.3

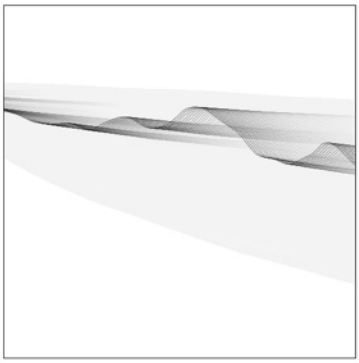


Figura 2.4

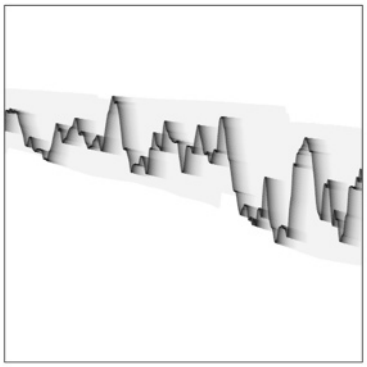


Figura 2.5

El siguiente fragmento de código expresa el comportamiento de la función noise para lograr diferentes efectos controlando el incremento del valor de la variable en la que es evaluada dicha función. Se podrá observar que el comportamiento con incremento de 0.005 cada ciclo corresponde al de la figura 2.4.

```
for (int x = 60; x < width-60; x++) {
  thisNoise += 0.005;
  yrange+=noise(thisNoise);
  float y = ystart + (noise(thisNoise) * yrange);
  stroke(_moveX, 80);
  if (lastx != -1.0) {
    line(lastx, lasty, x, y);
  }
  lastx = x;
  lasty = y;
}
```

Con un incremento inferior de 0.0005 se tiene el comportamiento expresado en la imagen 2.3 y finalmente con el mayor incremento de 0.05 se obtuvo el resultado correspondiente a la imagen 2.5. Implementar **noise( )** en dos y tres dimensiones, el principio es el mismo solo que en estos casos el valor de salida no varia únicamente en un sólo eje, lo hace también con valores laterales y diagonales.

2.3 Agentes Autónomos

Cuando se hace referencia al término agente autónomo, generalmente se define como una entidad capaz de tomar sus propias decisiones acerca de cómo actuar en su entorno, sin la influencia o intervención de un agente humano, un líder o un plan maestro previamente establecido. En el contexto de un entorno generativo cuando se dice que "toma decisiones", quiere decir que presenta un comportamiento (gráfico) el cual puede representar movimiento, forma, geometría y color. Daniel Shiffman, cuyo trabajo ha sido retomado por numerosos autores, propone que lo agentes poseen tres componentes clave los cuales son[7]:

-Un agente tiene capacidad limitada de percibir su ambiente. Es decir, deberá almacenar referencias de otros agentes para que de esta manera pueda percibir su entorno.

-Un agente autónomo procesa la información de su ambiente y calcula una acción. Como ya se había mencionado, puede ser un comportamiento gráfico.

-Un agente no tiene un líder al cual seguir. Esta tercera clave dependerá directamente del tipo de sistema que se plantea.

Gran parte de los trabajos que convergen con esta rama y el cómputo gráfico han sido basados en los algoritmos desarrollados por el científico Craig Reynolds. Su trabajo ha permitido crear sistemas capaces de reaccionar cada una de sus partes individualmente pero que a su vez presenten un comportamiento colectivo complejo e inteligente. El ejemplo mas famoso es el modelo "Reynolds´s boids" el cual permite simular el fenómeno natural que ocurre en los enjambres, bancos de peces, parvadas de aves, entre otros.

A continuación se presenta una alteración de un ejemplo encontrado en el libro: *"Generative Art"* de Matt Pearson, en este ejemplo se retoman los conceptos de ruido y agentes los cuales analizan si existen las determinadas condiciones para realizar acciones. Es decir, si detectan el acercamiento de cualquier otro agente en el entorno, entonces se genera una figura. Cada agente tiene movimiento, comportamiento y características propias, por ejemplo la velocidad y dirección del movimiento es regida por valores asignados independientemente a cada agente, pero a su vez cambian constantemente debido a la intervención de valores ruido que alteran de igual manera cada uno de esos valores. Así mismo se ha implementado interacción básica por medio del mouse, el cual al pulsar el "click" derecho aumenta el número de agentes en el sistema. También si el mouse es desplazado lateralmente sobre la pantalla, el efecto visual es cambiado paulatinamente. En la figura 2.6 se observa el comportamiento visual del sistema. Cabe destacar que la interacción es en tiempo real, para una mejor apreciación diríjase a: http://www.openprocessing.org/sketch/380671.

```
/*Se declara el número de agentes que se agregan al
sistema cada vez que el usuario pulsa el mouse (_num)*/

int _num = 5;
Circle[] _circleArr={};
//Variables noise son iniciadas

//... el código completo se puede retomar  en la liga proporcionada

  void draw() {

    //... el código completo se puede retomar  en la liga proporcionada

    //se revisa el estado del cada agente para determinar colisiones

    for (int i=0; i<_circleArr.length; i++){
      Circle thisCirc = _circleArr[i];
      thisCirc.updateMe();
```

```
    }
  }
```

//La función updateMe() modifica la posición de cada uno de los elementos del sistema, se detectan colisiones con los bordes de la ventana y con otros elementos del sistema.

```
 void mouseReleased() {
  drawCircles();

  //por cada acción de click derecho en el mouse, se crean y agregan nuevos
  elementos al sistema (de acuerdo a la variable  _num).

  println(_circleArr.length);
  }

 void drawCircles() {
  for (int i=0; i<_num; i++) {

      //se crean los nuevos agentes u objetos cada vez que se pulse el mouse

      Circle thisCirc = new Circle();
     _circleArr = (Circle[])append(_circleArr, thisCirc);

     //concatena el nuevo elemento al arreglo de objetos

    }
  }

class Circle {

//... el código completo se puede retomar  en la liga proporcionada

 Circle () {

  //declaración de las características de cada agente nuevo

  x = random(width);                  //posición aleatoria
  y = random(height);
  radius = random(100) + 10;          //tamaño de radio aleatorio
  xmove = random(10) - 5;             //velocidad X y Y aleatoria
  ymove = random(10) - 5;
 }
```

```
void updateMe() {

    //... el código completo se puede retomar  en la liga proporcionada

   //para evitar que salgan los agentes de la ventana se evalúa su posición y se
   reposiciona en caso de "tocar" un borde.

 if (x > (width+radius)) { x = 0 - radius; }
 if (x < (0-radius)) { x = width+radius; }
 if (y > (height+radius)) { y = 0 - radius; }
 if (y < (0-radius)) { y = height+radius; }

 for (int i=0; i<_circleArr.length; i++) {
  Circle otherCirc = _circeArr[i];

  //para cada agente excepto el mismo evalúa la distancia entre ambos

  if (otherCirc != this) {
   float dis = dist(x, y, otherCirc.x, otherCirc.y);

   //verifica si hay traslape entre lo círculos

   float overlap = dis - radius - otherCirc.radius;
   if (overlap < 0) {
      float midx, midy;

      //calcula el punto medio entre ambos círculos (si hay colisión)

      midx = (x + otherCirc.x)/2;
      midy = (y + otherCirc.y)/2;
      noFill();

      //el valor de traslape se convierte en el radio de un nuevo círculo que será
      dibujado

      overlap *= -1;

      //Color de línea de acuerdo a posición del mouse

      stroke(mouseX,20);

      //Se dibuja un círculo desde el punto medio hasta el centro del circulo

      ellipse(midx, midy, overlap, overlap);

      //color de trazo

      stroke(random(180),random(20),random(20),10);
```

```
            //se dibuja una línea que conecta el centro de los círculos que han colisionado

            line(x, y,otherCirc.x , otherCirc.y);
          }
        }
      }
    }
  }
```

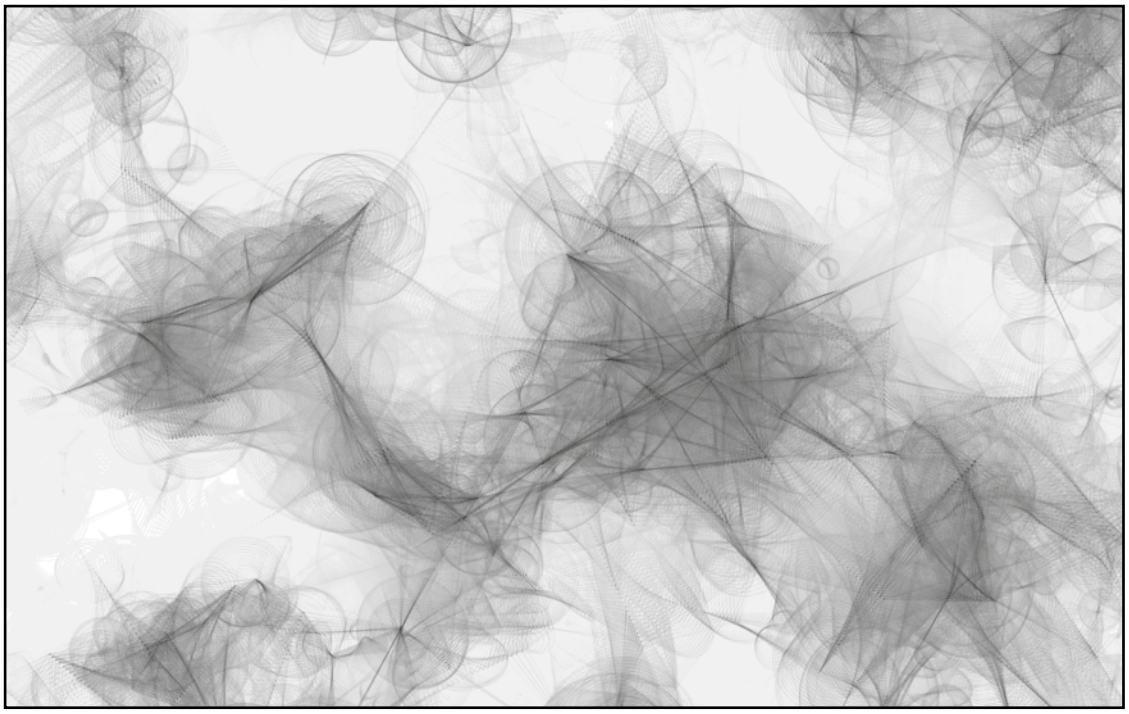


Figura 2.6

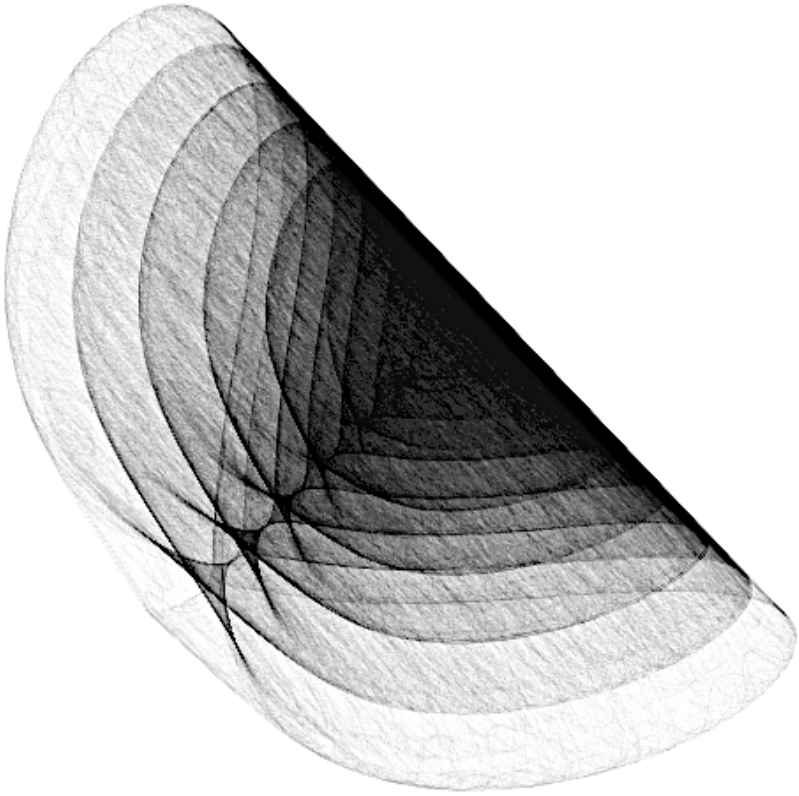
2.4 Inestabilidad Numérica

En al ámbito matemático del análisis numérico, la estabilidad matemática se define como una propiedad de los algoritmos numéricos, la cual describe la propagación de los errores en los datos de entrada a través del algoritmo. Cuando un método numérico es estable, los errores ocasionados por aproximaciones son mitigados o atenuados durante el proceso de cómputo. En contraste, cuando se tiene un método inestable hasta el error mas mínimo se magnifica rápidamente. Esta clase de métodos generan información basura que en la mayoría de los casos se considera inútil durante el proceso.

Esta propiedad es utilizada en algunos algoritmos generativos, dado su carácter "impredecible". Es común encontrar variables declaradas con un tipo "erróneo", por ejemplo declarar variables tipo "float" en lugar de "double" cuando estas incrementan rápidamente, eventualmente los valores de estas variables se vuelven inexactos y el comportamiento del algoritmo llega a degenerarse exponencialmente. En muchos casos se aprovecha de este sencillo recurso para lograr efectos visuales y de audio interesantes, en estos casos la inestabilidad pueda ser deseable. A pesar de llegar a encontrar autores[8] que mencionan esta práctica, no se destaca que para estos fines se presenta una problemática, ya que los resultados de inestabilidad dependen directamente de la aritmética de punto flotante de cada CPU. Además como los resultados varían de manera exponencial, difícilmente es posible implementarlo en experiencias en las que intervengan la interacción y la animación en tiempo real.

Capítulo 3

Herramientas de Hardware para el apoyo en la interacción



Este capítulo expone los posibles recursos para establecer la transmisión de mensajes entre el usuario y el sistema, permitiendo un intercambio de acciones que den lugar a un sistema interactivo intuitivo. Además se mencionan las principales herramientas empleadas por la industria para crear experiencias interactivas.

3.1 Transmisión de Mensajes

La interacción se genera a partir de mensajes, es decir, la información enviada del usuario al sistema y viceversa. Como se mencionó anteriormente, estos mensajes pueden traducirse como texto, voz, sonido, color; un comportamiento visual o mecánico, incluso se puede tratar de un ademán o movimiento físicos. Claramente se puede distinguir que dependiendo el tipo de aplicación, se deberá elegir correctamente el medio y formas de comunicación.

Uno de los retos mas grandes en la creación de aplicaciones interactivas, es lograr comprender como el sistema entiende los mensajes provenientes del usuario y como el usuario entiende los mensajes enviados por el sistema. Aquellas aplicaciones que son consideradas con alto grado de interacción, permiten realizar una amplia variedad de acciones entre el sistema y el usuario, ya que es muy fácil que en algún punto del intercambio de mensajes, la información se torne difusa o difícil de interpretar para el sistema y en consecuencia la experiencia de usuario se vea comprometida.

Como ya se ha mencionado, en el intercambio de mensajes entre el sistema y usuario deberá existir un puente facilitador de comunicación llamado interfaz. Dichos elementos necesitarán de un lenguaje para poderse comunicar, en otras palabras y a modo de ejemplo; cuando en una computadora se desea agregar un archivo a una carpeta existente, usualmente en el monitor se “arrastra” un ícono que representa dicho archivo hacia otro ícono que representa a su vez la carpeta destino. Lo importante a comprender es que las acciones que realiza el usuario deberán de ser las mismas que el sistema conoce y por tanto se interpretarán de la forma y manera esperada. De igual forma sucede cuando se escribe el código en un lenguaje de programación, para lo cual se requiere que ambas partes conozcan el significado de los símbolos y declaraciones así como el orden que deben adoptar.

Existen diferentes formas de categorizar los tipos de información, a continuación se muestra una clasificación tomada de: “Programming Interactivity” de Joshua Noble.

“Manipulación física: Fueron las primeras en aparecer en la industria electrónica, a esta se engloban sistemas que involucran presionar botones físicos, deslizadores, perillas, etc. Usualmente se requiere un monitor que despliegue las múltiples fuentes de información dadas mientras se manipulan los controles, por ejemplo un osciloscópio.

Utilización de código como entrada: Este es el claro ejemplo cuando floreció la era de las computadoras, la clásica interacción se traducía como una terminal donde el usuario introducía comandos que posteriormente eran ejecutados y finalmente se presentaban los resultados en un monitor en forma de texto.

Manipulación del mouse: Este es el método mas común de interactuar con una computadora actualmente y la interfaz para la cual muchas aplicaciones han sido diseñadas. Algunas de las técnicas implementadas como lenguaje han sido: *drag-and-drop, double-click y click-and hold*.

Detección de presencia y ubicación: El uso del concepto de presencia y ausencia del usuario o participante es sumamente simple pero al mismo tiempo es una manera altamente intuitiva de interactuar. Utiliza recursos como la detección de masa, movimiento, luz, cambios de temperatura y en algunos casos sonido. La reacción es simple, actuar como interruptor, iniciar o terminar el proceso. Como consecuencia nos podemos conducir a fascinantes modos de interacción que usan tecnología llamada *computer vision*, la cual nos permite procesar imágenes generadas por una cámara, es decir, se analizan los pixeles de dicha imagen; lo cual nos permite detectar la ubicación de objetos; detección de formas, profundidad, rostros humanos, contornos, patrones, entre otras aplicaciones.

Interfaces hápticas y multitouch: La esencia de esta forma de interacción esta basada en emplear a gestos que son utilizados por los usuarios en la vida cotidiana, por ejemplo: usar los dedos para contraer o expandir, para rotar un objeto o señalar para seleccionar. Este tipo de gestos se han convertido en un lenguaje que podemos llevar mas allá del lenguaje natural.

Ademanes: El análisis de ademanes es un interesante modelo de interacción por que fácilmente se asocia con señas, escritura y movimiento del cuerpo. Este tipo de interacción no está guiada por el uso de un teclado o mouse, dado que estos para ciertas tareas resultan poco intuitivos. Este modelo es implementado en dispositivos con interfaces táctiles, plumas o superficies utilizadas para aplicaciones de diseño y dibujo, tecnologías diseñadas para personas con necesidades especiales y aplicaciones para niños.

Reconocimiento de voz: Hace referencia a la programación de una computadora para reconocer ciertas palabras o frases para posteriormente ejecutar tareas basadas en esos comandos. Dichos

comandos pueden ser simples como la activación por voz hasta responder a preguntas concretas realizadas por el usuario, basándose en información encontrada en internet. En términos simples, para una computadora las palabras o comandos son reconocidos como patrones de sonido que son comparados a su vez con un diccionario de patrones para determinar a que comando corresponde."[9]

Esta lista pretende representar una fracción de los posibles modelos de interacción, actualmente se están innovando y creando nuevas formas de interacción acordes a las nuevas tecnologías, tendencias de entretenimiento, necesidades de la población y educación por mencionar algunas.

3.2 Herramientas de Software para el apoyo de la Interacción

A continuación se presenta una muestra de diferentes herramientas y lenguajes empleados en la industria del cómputo gráfico, entretenimiento, interacción, entre otras. No se pretende enunciar todas las herramientas que se encuentran disponibles hoy en día, si no hacer mención de aquellas que han sido referidas comúnmente por autores, expertos, artistas e ingenieros.

3.2.1 Processing

Processing fue uno de los primeros proyectos de código abierto creados específicamente para la práctica y desarrollo de aplicaciones interactivas. Su descarga, uso y modificación es completamente gratuita. El proyecto fue iniciado por Casey Reas y Ben Fry en el MIT bajo la tutela de John Meada, aunque actualmente un grupo de ingenieros y desarrolladores mantienen esta herramienta en constante actualización. Mediante processing es posible desarrollar aplicaciones en javascript (p5.js); aplicaciones para Android, en línea y para la mayoría de los sistemas operativos. Finalmente la razón por la que esta herramienta es tan flexible; es por que Processing genera aplicaciones Java; lo cual significa que corre en una máquina virtual, en este caso una Máquina Virtual de Java (JVM).

3.2.2 OpenFrameworks

Openframeworks es una herramienta que integra bibliotecas ampliamente empleadas en la industria, por ejemplo: OpenGL, GLEW, GLUT, rtAudio, PortAudio, OpenAL, FreeType, Quicktime, OpenCV, Assimp, por mencionar algunas. Las aplicaciones escritas por medio de Openframeworks son altamente compatibles con diferentes sistemas operativos, permitiendo desarrollar para

Windows, OSX, Linux, iOS y Android en diferentes IDEs (XCode, Code::Blocks, Visual Studio y Eclipse). Al igual que Processing, OpenFrameworks es distribuido bajo licencia del MIT, lo cual permite su libre uso para fines comerciales y no comerciales. Actualmente esta herramienta es utilizada en proyectos cuya interacción involucre a dispositivos con IOS como un elemento del sistema.

3.2.3 Cinder

Cinder es una biblioteca diseñada para que el lenguaje C++ adquiera capacidades avanzadas de visualización y en muchos sentidos es comparable a OpenFrameworks. La principal diferencia entre ambos es que Cinder utiliza bibliotecas mas apegadas al enfoque para el cual fue creado, desarrollo de aplicaciones móviles, por tanto presenta un mejor desempeño.

3.2.4 Pure Data (Pd)

Pd es un lenguaje de programación gráfico diseñado para crear experiencias visuales y sonoras en tiempo real. Pd fue concebido por Miller Puckette, como una alternativa de código abierto a "MAX", otra herramienta similar pero con costos de licenciamiento y con opción de procesar video en tiempo real. Esta herramienta es ampliamente usada en aplicaciones que involucran ejecuciones en vivo. Otras herramientas similares a Pd son: MAX la cual está enfocada a la síntesis de audio y SuperColider.

3.2.5 Vvvv

Vvvv o también conocido como 4v es un software enfocado al diseño de prototipos de sistemas que involucran procesamiento de video, conexiones con dispositivos físicos, desarrollo de aplicaciones y medios interactivos. Es un lenguaje de programación visual similar a MAX y Pure Data, pero con la diferencia de que la interfaz de 4v está menos enfocada a experiencias sonoras. Dicha herramienta es posible descargarla de manera gratuita aunque este tipo de licenciamiento está limitado al desarrollo de aplicaciones con enfoque no comercial.

3.2.6 Otras Herramientas Relevantes

Por otro lado es posible mencionar aquellas herramientas que no precisamente están enfocadas en el desarrollo de experiencias generativas o interactivas, mas bien son de propósito general pero

resultan de gran utilidad cuando se desea integrar con otros medios. Por ejemplo el lenguaje C, SQL, PHP, ensamblador, Wiring (lenguaje enfocado a la programación de microcontroladores), Perl, por mencionar algunos.

3.3 Herramientas de Hardware para el apoyo de la interacción

Al día de hoy existen un sin número de alternativas de hardware con las que es posible desarrollar aplicaciones interactivas, algunas de ellas parten desde los componentes electrónicos, por ejemplo: sensores de temperatura, giroscopio, acelerómetro, encoder; dispositivos como: potenciómetros, cámaras, monitores, proyectores, motores, hasta herramientas complejas como Kinect, RFID y Leap Motion. Lo más importante es determinar que herramientas son las más apropiadas de acuerdo al objetivo y enfoque de un aplicación interactiva.

3.3.1 Kinect de Microsoft

Es un dispositivo diseñado por Microsoft que alberga una cámara RGB, un sensor de profundidad basado en un proyector infrarrojo, un micrófono que permite detectar la dirección de la fuente de sonido y un procesador de imagen encargado de la captura de todo el cuerpo en 3D, con capacidades de reconocimiento facial y de voz. Este dispositivo ha sido altamente empleado en diferentes aplicaciones mas allá de los videojuegos, ya que por sus características técnicas permite ser como un escáner, como sensor para manipular mecanismos o incluso ámbitos experimentales. Gracias a los esfuerzos de la comunidad de usuarios, se han desarrollado bibliotecas que permiten manipular este dispositivo fuera de las plataformas desarrolladas por Microsoft. Por ejemplo en Processing y OpenFrameworks se cuentan con bibliotecas como OpenNI, ofxKinect y KinectPV2 que han permitido llevar este dispositivo a *espacios públicos de recreación o incluso galerías de arte*[10]. El Funcionamiento de dichas bibliotecas se basa en obtener las imágenes de los sensores así como la información arrojada por el procesador de profundidad así como del rastreo de extremidades, posteriormente se accede a la información RAW de cada una de la imágenes obtenidas finalmente esta información puede ser transportada a un espacio virtual en 3D para su visualización, análisis de rostros, obtener contornos de imágenes o cualquier aplicación que se desee implementar. A continuación se presenta un diagrama que ilustra como trabaja la biblioteca KinectPV2 dirigida para Processing.

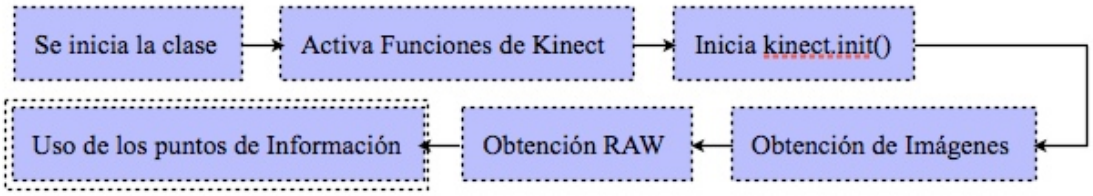


Figura 3.1

3.3.2 Leap Motion

Leap Motion es un dispositivo periférico el cual está diseñado para ser instalado en una computadora de escritorio o laptop, aunque recientemente ha sido utilizado como complemento a los lentes de realidad virtual. Internamente se encuentra compuesto por dos cámaras infrarrojas y tres LEDs infrarrojos, los cuales en conjunto permiten analizar información en un área semiesférica con un radio aproximado de distancia de 1 metro. Dicho conjunto de componentes permiten analizar hasta 200 cuadros por segundo de información gráfica, posteriormente dicha información es analizada mediante un software desarrollado por Leap Motion el cual ejecuta complejas funciones matemáticas, las cuales no han sido reveladas por el fabricante. Lo poco que se ha sabido sobre su funcionamiento es que puede sintetizar información en 3D a partir de la comparación de cuadros en 2D generados por dos cámaras simultáneamente logrando una exactitud promedio de 0.7 milímetros[11].

Actualmente al igual que el Kinect de Microsoft, es posible explotar sus capacidades mediante otras herramientas, por ejemplo la biblioteca leapmotion p5 para processing o ofxLeapMotion para OpenFrameworks.

3.3.3 Video Mapping

Propiamente esta herramienta está basada en el uso de uno o varios proyectores, los cuales tienen la función de desplegar imágenes sobre superficies usualmente edificios, esculturas o espacios urbanos. Desde el punto de vista interactivo es posible emplear esta técnica para lleva la experiencia de usuario a espacios aún no explorados. Para lograr que una aplicación interactiva desarrollada en ambientes como Processing y OpenFrameworks es necesario implementar el protocolo TCPSyphon el cual permite establecer conexión entre “hosts” idénticos que procesan información gráfica. Esta utilidad permite que la información fluya a través de TCP/IP para realizar una red trasparente. Por otro lado representa una alternativa de código abierto al protocolo AirPlay desarrollado por Apple el cual permite transportar video, audio, compartir pantalla y fotografías entre dispositivos de manera inalámbrica empleando los protocolos UDP y RTSP[12].

Actualmente las bibliotecas para implementar este protocolo son: ofxSyphon para OpenFrameworks y Syphon para Processing, las cuales internamente habilitan el canal de comunicación únicamente en fragmento de código que representen la visualización de los elemento gráficos a transmitir. Finalmente esta es una herramienta que permite realizar interacción en tiempo real implementando una salida mas allá del monitor y una proyección plana.

3.3.4 Otras Herramientas Relevantes

Existe una amplia cantidad de herramientas de hardware, algunas de ellas pueden ser utilizadas para fabricar prototipos por ejemplo la tarjeta Arduino, la cuál a pesar de sus limitadas capacidades es posible implementarla para generar rápidas soluciones a sistemas de control. Por otro lado una solución definitiva puede ser implementada mediante el uso y programación de microcontroladores, los cuales debido a su amplia variedad de características se adecuan a las exigencias de cada aplicación. Otras opciones importantes son: Raspberry Pi y la tarjeta Galileo, de Intel.

Actualmente existen soluciones móviles para el desarrollo de aplicaciones gráficas avanzadas, tal es el caso de las tarjetas Jetson series TX y TK, las cuales *"representan una plataforma de computo gráfico para el procesamiento acelerado por las GPUs en los sistemas móviles embebidos. Su alta capacidad de cálculo y bajo consumo energético permiten desarrollar aplicaciones dirigidas hacia el deep learning y visión computarizada"*[13]. Esta herramienta ha sido utilizada para la creación de robots autónomos y drones.

Finalmente existen periféricos que también son altamente empleados en el desarrollo de dichas aplicaciones, por ejemplo "wearables". Uno de ellos el Myo Armband, el cual es un dispositivo de reconocimiento de gestos desarrollado y fabricado por Thalmic Labs. Esta herramienta con forma de un brazalete permite controlar otros dispositivos de manera inalámbrica por medio de gestos generados por el movimiento del brazo. Funciona mediante un conjunto de sensores electromiográficos (EMG) que muestrean la actividad eléctrica del antebrazo, posee un giroscópio, acelerómetro y un magnetómetro que en conjunto permiten reconocer dichos gestos. Este dispositivo es empleado para aplicaciones con el mismo enfoque que Leap Motion, por lo que en la industria pueden llegar a ser confundidos.

Capí-tulo 4

Metodología

Este capítulo explica una metodología adecuada para el desarrollo de aplicaciones interactivas con enfoque generativo. Actualmente es necesario recurrir a la metodología SCRUM, la cual ha sido pensada para el desarrollo de videojuegos, pero es relevante por que involucra el trabajo de diferentes disciplinas por ejemplo: las artes y el diseño.

4.1 Proceso de Trabajo

El proceso de trabajo en la creación de sistemas interactivos generalmente se concibe como la combinación de los siguientes procesos: concepción, investigación, diseño, desarrollo, pruebas y entrega[14]. Cabe destacar que el desarrollo de sistemas generativos converge con distintas disciplinas, por lo que el desarrollo de software y hardware se considera como un módulo dentro del proyecto[15]. De las metodologías estudiadas durante el proceso de formación académico se retomarán características de la metodología SCRUM, la cual últimamente se le han atribuido características para el desarrollo de videojuegos. Aunque el desarrollo de aplicaciones interactivas converge tangencialmente con el ámbito de los videojuegos, esta metodología logra integrar el proceso creativo con una metodología sólida para el desarrollo de software[16]. Por otra parte es necesario adaptar una metodología no solo para integrar el proceso creativo con el desarrollo de software, también se involucran otras áreas del conocimiento como la psicología, diseño industrial, arquitectura y artes, los cuales poseen sus propias metodologías que en términos generales son muy ajenas a las del software. Por lo cual, en términos generales se ha optado por la metodología de tipo cascada, la cual aunque limitada y conservadora, es flexible, permitiendo adaptarse fácilmente al flujo de trabajo de distintas disciplinas.

Finalmente se puede definir que la metodología empleada será en forma general; cascada, retomando conceptos y procesos de SCRUM en el contexto de desarrollo de videojuegos. Algunos elementos retomados de esta metodología es el uso de sprints como el corazón del proceso de desarrollo, documentación de procesos, creación de entregables y creación de prototipos (en el caso de SCRUM con enfoque a los videojuegos se plantea la creación de prototipos los cuales son: pre-prototipo, prototipo, demo, alpha y beta, los cuales también son aplicables para el desarrollo de aplicaciones interactivas y en este caso con un enfoque a los algoritmos generativos).

En términos generales bajo el esquema de cascada antes mencionado las principales actividades para el desarrollo de aplicaciones interactivas de acuerdo con Joshua Noble, autor de libros y artículos enfocados a la interacción son[17]:

Concepción: Este proceso consiste en esbozar las ideas o requerimientos producto de entrevistarse con el cliente. En esta parte del proyecto se definirán a detalle las características del sistema,

capacidades, las sensaciones que transmitirá el sistema hacia el usuarios y objetivos de la aplicación.

Investigación: En esta etapa, es cuando es definida la tecnología que será implementada para desarrollar el proyecto. En esta etapa intervienen otras disciplinas, por ejemplo: Psicología y diseño industrial, las cuales a su vez intervienen en un proceso de investigación para poder contextualizar correctamente el espacio, el tipo de interacción, la instalación y el tipo de interacción con las sensaciones que se transmitirán al usuario.

Diseño: Desde el punto de vista del software, esta etapa del desarrollo va ligada a la traducción de los requisitos funcionales y no funcionales en una representación de software[18]. Como se podrá notar, a diferencia de una metodología puramente enfocada al software, ésta es la tercera etapa en la que no solo se realiza el plasmado de requerimientos, también intervienen los resultados de las investigaciones realizadas por las disciplinas antes mencionadas. Algunas de la pautas clave en esta fase del proyecto son explicadas por John Maeda, Fundador el Grupo de Computación y Estética del Media Lab del MIT, en su obra mas destacada: *"Leyes de la Simplicidad (2006)"*, la cual proporciona una visión de la complejidad de las tecnologías y la combinación entre la forma y el código de la comunicación visual[19].

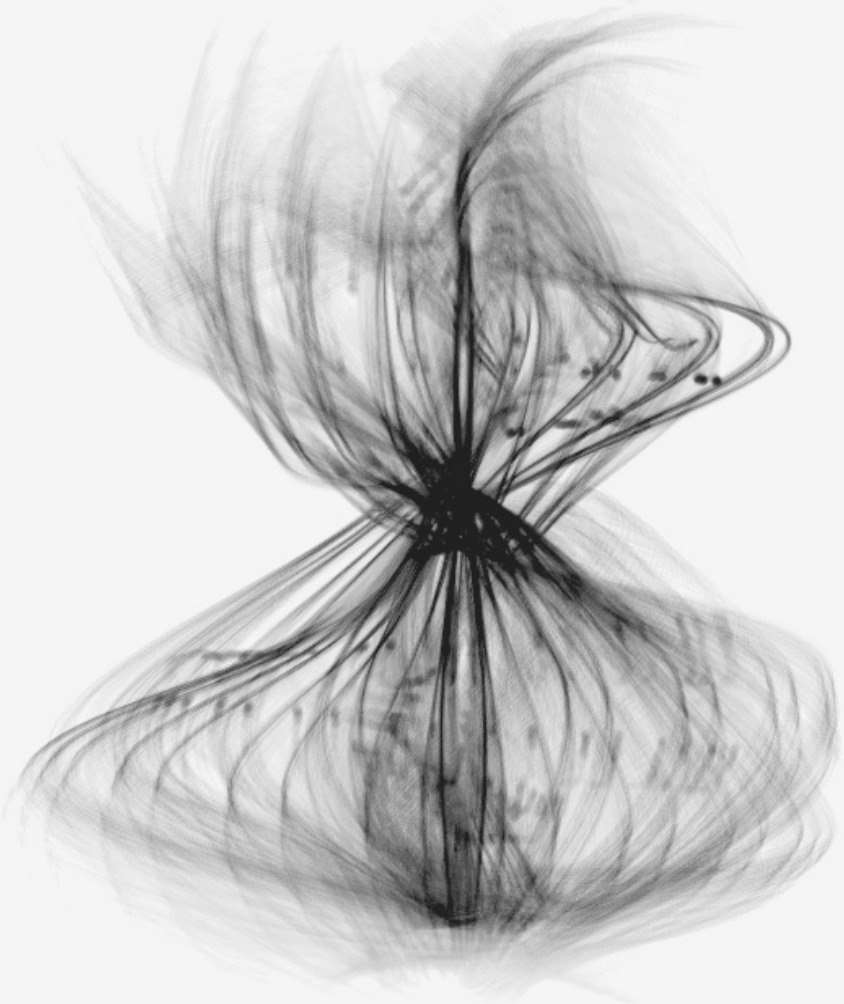
Construcción: En esta etapa del proceso, son desarrollados e integrados los módulos que constituyen el proyecto: software, hardware, instalación, espacio y diseño, para su posterior ensamblaje.

Pruebas: La etapa de pruebas se constituye como el conjunto de actividades encargadas de proporcionar información objetiva sobre la calidad del producto a la parte interesada, es decir el cliente.

Entrega: Se define como el conjunto de actividades necesarias para llevar a cabo las entregas de software y documentación englobadas en el proyecto. Dichas actividades incluyen la entrega formal del producto, revisión final y validación de la misma[20].

Capítulo 5

Aplicaciones e implementaciones



Este capítulo muestra las diferentes aplicaciones desarrolladas así como los resultados gráficos obtenidos. Como se podrá observar la mayor parte de las aplicaciones fueron desarrolladas con el lenguaje de programación Processing el cual es ampliamente usado en la industria actualmente.

El desarrollo de aplicaciones de esta naturaleza ha crecido debido al surgimiento emergente del marketing experiencial y transmedia.

5.1 Aplicaciones Gráficas de los Algoritmos Generativos

En este capítulo se abordarán distintas aplicaciones desarrolladas empleando técnicas y herramientas expuestas en los capítulos previos. La siguiente aplicación fue expuesta en el Museo Universitario de Arte Moderno (MUAC) con el propósito del cierre de actividades referentes a la exposición *Pseudomatismos* del artista e ingeniero químico Rafael Lozano-Hemmer. Dicha aplicación junto con otras desarrolladas por equipo multidisciplinarios formaron parte de una muestra llamada "Nuevas Herramientas para el Proceso Creativo". La aplicación desarrollada en el lenguaje Processing produce superficies generativas, es decir a partir de un molde o modelo base dado por fórmulas de superficies (primeramente se desarrolló con fórmulas correspondientes a superficies conocidas, por ejemplo: toroide, elipsoide y esfera, posteriormente se implementaron fórmulas que dieron como resultado figuras complejas), posteriormente los valores correspondientes a la trayectoria de dibujo de los trazos de la imagen son alterados por medio del proceso generativo. Finalmente se podrán apreciar imágenes que presentan una textura en común característica, a pesar de las diferentes superficies que representan. Dicha textura pretende alejarse de la perfección del dibujo trazado de manera artificial, mas bien imita la forma en la que una persona lo haría con papel y un lápiz, con errores y bordes irregulares, pero sin llegar al punto de desplegar figuras amorfas y carentes de sentido producto de un errado proceso aleatorio en cuyo único fin descansaría lo "impredecible". Todas las aplicaciones desarrolladas en este trabajo fueron diseñadas para sistema operativo OSX versión 10.8.5 ó superior.

5.1.1 Formas a través de las ecuaciones parámetricas de superficies

```
//Variables Globales//
/*_num hace referencia al número de objetos independientes con los que será
  dibujada la superficie (300). La forma en como se definieron las variables
  ha sido retomada de diferentes autores, pero principalmente de Matt Pearson
  y su libro Generative Art del cual fue retomado el algoritmo para dibujar un
  circulo y la declaración y uso de objetos mediante arreglos.*/


int _num = 300;
int e,r,t;
Circle[] pArr = new Circle[_num];

void setup() {
```

```
  //Se indica el tamaño de la ventana con la propiedad de render en 3D
  size(1200, 700, P3D);
  smooth();

  //Ajuste de velocidad de refresco de pantalla a 60 cuadros por segundo

  frameRate(60);
  background(230);

  //Posicionamiento del inicio de trazo así como configuración de la perspectiva
  //de la cámara (no esta situada completamente al centro de la pantalla).

  float startx =  (width/4)*3;
  float starty =  (height/4)*3;
  float startz =0;
  camera(60, 180, 80, (width/2), (height/2), 0,  1.0, 1.5, 1.0);

  //Son inicializados los objetos mediante un arreglo y colocados en una misma posición

  for (int i=0;i<_num;i++) {
    pArr[i] = new Circle(i);
    pArr[i].init(startx, starty, startz);
  }
}

void draw() {

  // Cada ciclo son modificados los valores de posición de cada objeto

  for (int i=0;i<_num;i++) {
    pArr[i].update();
  }
}

//*********************************************INTERACCIÓN
void keyReleased() {

  //Si la tecla "S" o "s" es oprimida, se guardará una captura de la imagen generada

  if (key == 's' || key == 'S') saveFrame("_##.png");
  }


void mousePressed() {

  // Si el mouse es presionado, el programa se reinicia
  setup();
}
```

```
//*******************************************OBJETOS

class Circle {
  int id;                          //ID de objeto
  float angnoise, radiusnoise;    //Tanto el radio como el ángulo son afectados por el efecto
                                   //generativo, deformando su comportamiento
  float angle = 0;                 //ángulo de inicio de figura 1
  float angle2 = 0;                //ángulo de inicio de figura 2
  float radius;                    //radio de la figura (para ambas)
  float centreX;                   //Centro en el espacio X,Y,Z
  float centreY;
  float centreZ;
  float strokeCol;                 //Color de trazo
  float angleincr;                 //Incremento del ángulo
  float lastX = 9999;              //Para evitar trazos "ruido" en el primer ciclo
  float lastX2 = 9999;             //Se declaran valores altos fuera de alcance
  float lastY,lastY2,lastZ, lastZ2;          //Son declaradas variables de respaldo de la
  float lastOuterX, lastOuterY, lastOuterZ; //última posición y generar trazos (líneas)

  Circle (int num) {

    //A cada objeto le es asignado un ID

    id = num;
  }

  void init(float ex, float why, float ze) {

    //El inicio del trazo será denominado el centro del trazo

    centreX = ex;
    centreY = why;
    centreZ = ze;

    //Se tendrá una valor de radio inicial independiente asignado de manera aleatoria

    radiusnoise = random(1);

    //El color de cada trazo es determinado en la escala de grises de forma aleatoria

    strokeCol = random(255);

    //Ángulo de inicio es asignado de forma aleatoria

    angle = random(180);
```

```
  //El incremento es mínimo de un grado por ciclo mas una fracción aleatoria

  angleincr = random(1) + 1;
}

void update() {

  //Cada Ciclo el ruido del radio es incrementado

  radiusnoise += 0.02;

  /*El radio es afecta por el siguiente proceso generativo:
   El valor que contantemente esincrementado, se evalua con la funcion noise,
   la cual arroja valores entre 0 y 1, multiplicado por un valor proporcional
   a un cuarto del ancho de la ventana, multiplicado por un factor de proporción
   (5), el cual se obtuvo para obtener un tamaño deseable en la figura. Finalmente
   es añadido nuevamente el valor del ruido de radio evaluado en la función ruido
   para obtener pequeños bordes irregulares a la forma.
   */

  radius = (5*(noise(radiusnoise) * (width/4)))+noise(radiusnoise);

  //El ángulo se incrementa

  angle += angleincr;

  //Son evaluados en valores entre -PI y PI, ciertos casos los resultados son
  //significativamente diferentes al modificar dicho intervalo.

  float rad2 = (radians(angle)-(PI));
  float rad = rad2;

    /*
    //Estas fórmulas corresponden a figuras en 2D, que se implementaron en la primera
    //etapa del desarrollo.
    float x2 = centreX + ((radius * cos(rad))/(1+pow(sin(rad),2)));
    float y2 = centreY + ((radius * sin(rad)*cos(rad))/(1+pow(sin(rad),2)));// infinito
    float y1 = centreX -280 - ((radius/1.5 * cos(rad))*(1-cos(rad)))*1.2;
    float x1 = centreY + ((radius/1.7 * sin(rad)*(1-cos(rad))));// cardioide
    */
    /*

    //El código comentado corresponde a figuras que han sido exploradas y se han mantenido
    //como registro debido a su importancia (3D).
    float x1 = centreX + cos(2*rad)*cos(rad2) + (radius/2)*cos(2*rad)*(1.5+sin(3*rad)/2);
    float y1 = centreY +sin(2*rad)*cos(rad2) + (radius/2) * sin(2*rad)*(1.5+sin(3*rad)/2);
    float z1 = sin(rad2)+200*cos(3*rad);
    */
```

```
    /*
    float x1 = centreX + radius * sin(3*rad) / (2+cos(rad2));
    float y1 = centreY + radius * (sin(rad) + 2*sin(2*rad)) /(2 + cos(rad2+3.1416*2/3));
    float z1 =  radius/2 * (cos(rad) - 2*cos(2*rad))*(2+cos(rad2)) * (2 + cos(rad2+3.1416*2/3)) / 4;

    float x2 = centreX + (radius * sin(rad) * cos(rad) * cos(rad2));
    float y2 = centreY + (radius * sin(rad) * sin(rad2) *cos(rad2));
    float z2 = radius * (cos(rad)* cos(2*rad));
    */

    float x1 = centreX + radius * sin(3*rad) / (2+cos(rad2));
    float y1 = centreY + radius * (sin(rad) + 2*sin(2*rad)) /(2 + cos(rad2+3.1416*2/3));
    float z1 =  radius/2 * (cos(rad) - 2*cos(2*rad))*(2+cos(rad2)) * (2 + cos(rad2+3.1416*2/3)) / 4;

    float x2 = centreX + (radius * sin(rad) * cos(rad2)*cos(radius*0.01)*0.6);
    float y2 = centreY + (radius * sin(rad) * sin(rad2)*cos(radius*0.01)*0.6);
    float z2 = radius * cos(rad)*sin(radius*0.001)*0.6;

   if (lastX != 9999) {

    //Se asigna grosor y color al trazo (los valores de color se multiplicaron por "basura"
    //intencionalmete).

    strokeWeight(1);
    stroke(strokeCol*r, strokeCol*e,strokeCol*t,10);

    //Es posible eliminar el comentario en la siguiente línea para desplegar 2 imágenes
    //simultáneamente

    /*line(x1, y1,z1, lastX, lastY,lastZ);*/

    //trazo recto entre la posición actual y anterior

    line(x2, y2,z2, lastX2, lastY2,lastZ2);
    }

   //Las siguiente líneas ayudan a almacenar la posición anterior y poder realizar trazos

   lastX2 = x2;
   lastY2 = y2;
   lastZ2= z2;
   lastX = x1;
   lastY = y1;
   lastZ = z1;
  }
}
```

A continuación se muestran los resultados obtenidos modificando el intervalo de valores en que las ecuaciones son evaluadas. Evidentemente se da origen a distintas formas o geometría dependiendo de la naturaleza de cada ecuación.

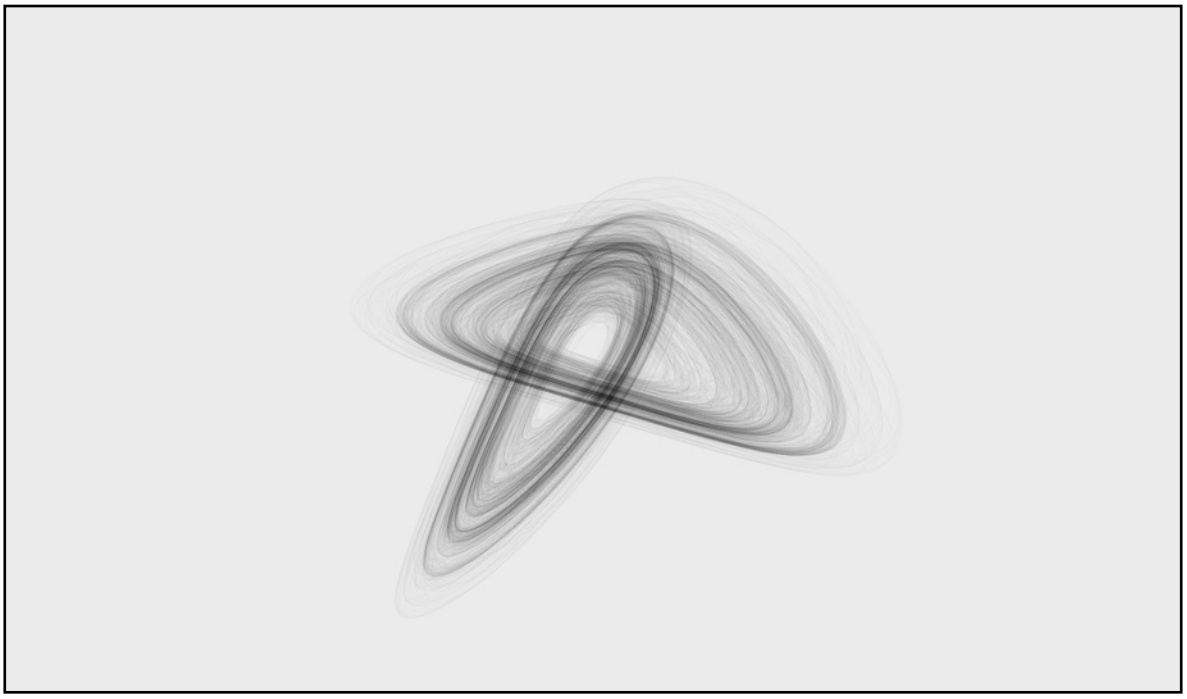


Figura 5.1

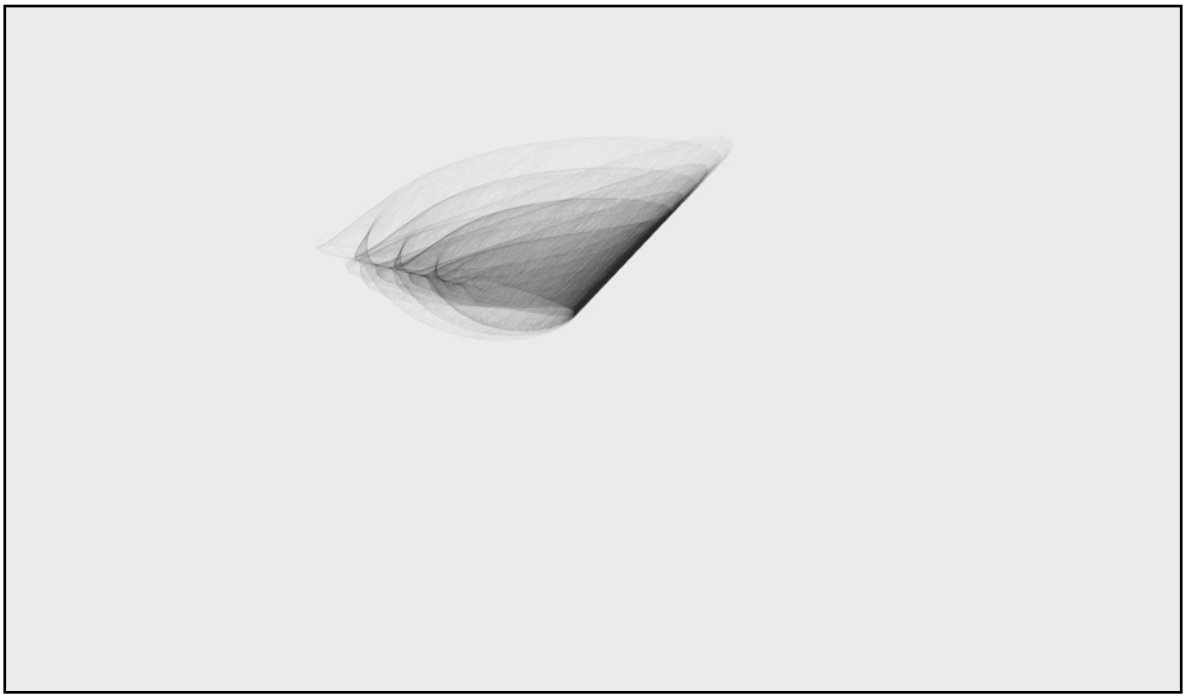


Figura 5.2

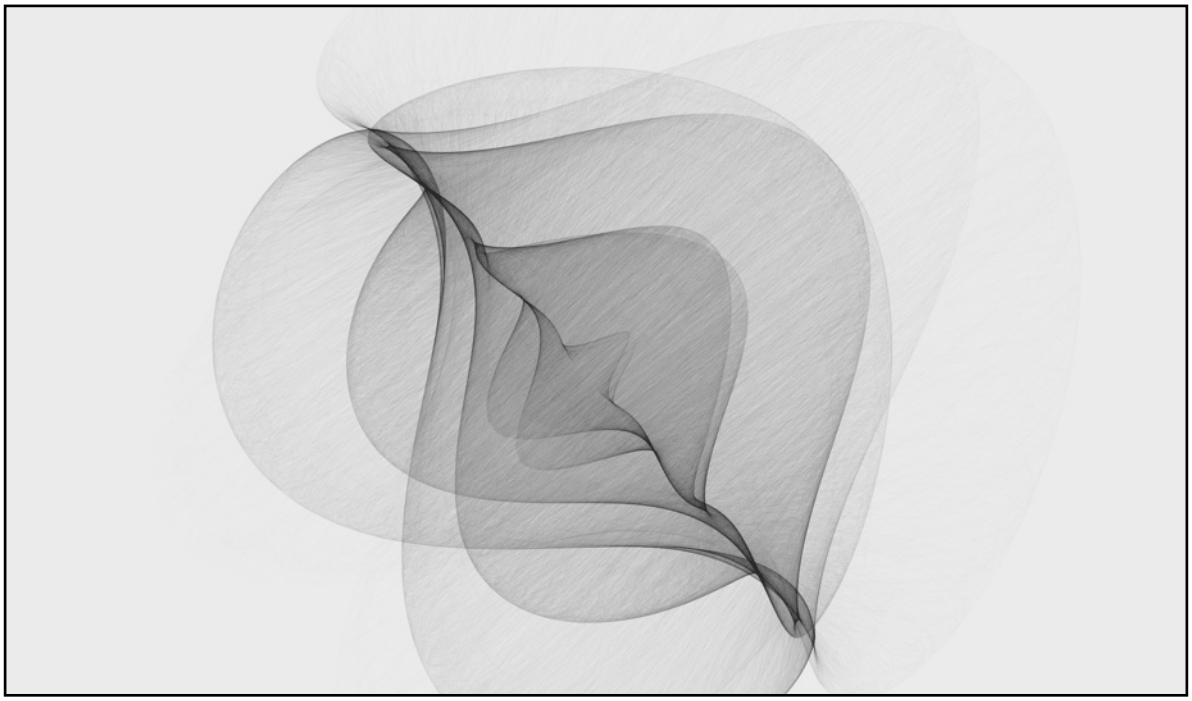


Figura 5.3

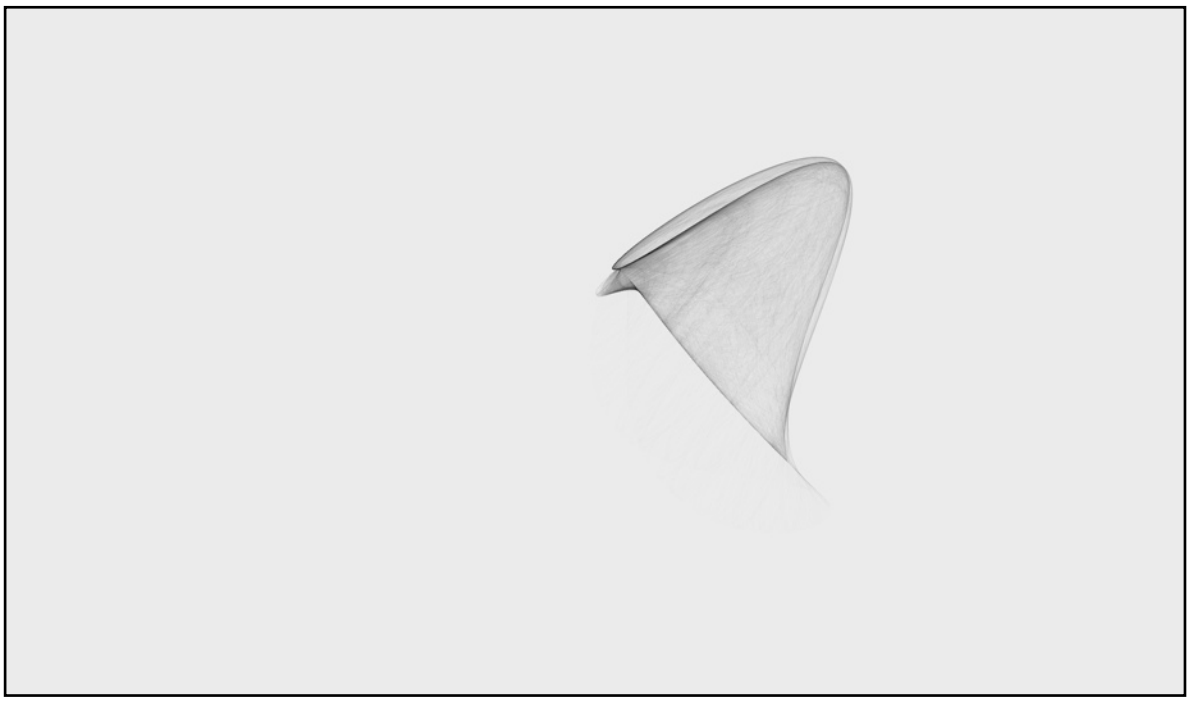


Figura 5.4

Las figuras anteriores (5.1, 5.2, 5.3 y 5.4) pertenecen a una serie de imágenes que puede ser consultada completamente a través de la siguiente dirección: https://armyrdz.wordpress.com/2016/09/13/introduccion-al-polimorfismo/. En dicha dirección son publicadas imágenes obtenidas a través de la exploración de distintos algoritmos generativos.

Los resultados obtenidos en la aplicación anterior, en la cual a través del tiempo son modificadas la trayectorias de los trazos que dibujan una sola imagen, es decir se trata de un sistema generativo estático que únicamente dibuja un cuadro "frame" y en el que el usuario decide en que momento realizar una captura de imagen (presionando la tecla "S" o "s") o reiniciar el programa (pulsando "click" derecho sobre cualquier punto de la ventana).

5.1.2 Partículas y ecuaciones paramétricas en 2D

A diferencia de la aplicación anterior, se tienen un sistema que cambia con cada refresco de pantalla, permitiendo producir una especie de animación generativa; aunque para definirlo formalmente, se trata una animación procedural o algorítmica la cual nos permite realizar animaciones en tiempo real y acciones que modifiquen el comportamiento de ésta, al contrario de las animaciones predefinidas o pre-realizadas[21].

La textura al igual que la aplicación anterior, pretende imitar el comportamiento de un lápiz de grafito al ser arrastrado sobre el papel, así como los bordes irregulares en las figuras. Este efecto es logrado utilizando intencionalmente el grosor de los trazos inferiores a un pixel, esto como consecuencia es interpretado por la computadora como trazos discontinuos. Los trazos discontinuos llegan a carecer de utilidad si son desplegados pocas veces por cuadro, en otras palabras es necesario "saturar" la imagen de éstos para que se sobrepongan unos con otros y de esta manera logren dibujar formas con ligeros detalles difusos.

La siguiente aplicación está basada en la forma en que el autor Daniel Shiffman ha desarrollado muchas aplicaciones en su texto "*Nature of Code*", los resultados expuestos a continuación distan de los expuestos en su trabajo y únicamente se ha retomado la manera en que manipula un sistema de partículas. Dicha referencia se ha mencionado ya que a través de dicho texto se explica como funciona un sistema de partículas y como manipularlas en el espacio de una forma sencilla y fácilmente aplicable a otros algoritmos[22]. Cabe señalar que dicho autor ha enfocado gran parte de su trabajo en el desarrollo del proyecto Processing así como la difusión y enseñanza de dicha herramienta.

```
/*Se Declara un objeto del tipo Wave, es posible implementar
mas de un sistema de partículas gracias a la clase*/

Wave wave0;

void setup() {
  background(255);

  //Al implementar P2D, es usado el modo de render por OPENGL
  //lo cual permite un framerate mayor al ser implementado

  size(1000,700,P2D);

  // Se inicia el sistema de partículas en el centro de la
  //pantalla, largo, amplitud y periodo.

  wave0 = new Wave(new PVector(width/2,height/2),25500,300,50);
}

void draw() {
  background(255);

  //método que calcula la posición de la partículas

  wave0.calculate();

  //método que despliega la imagen

  wave0.display();
 // saveFrame("circles-######.png");
}

//Clase partícula

class Particle {

  /*Las variables hacen referencia al posición de la partícula y
  a una variable baile corresponde al "ruido" o perturbación en la
  trayectoria de posición de cada partícula, es llamada baile
  por que a través de cambios minúsculos, se obtienen resultados
  posiblemente irrepetibles.*/

  PVector location;
  float baile;

  Particle() {
    location = new PVector();
    baile=random(10000000);
```

```
  }

  void setLocation(float x, float y) {

    //Se asignan posiciones a las particulas

    location.x = x;
    location.y = y;
  }

  void baileSuma()
   {

    //La variable baile será evaluada en la función noise(), por lo que la tasa
    //de cambio debe ser pequeña

    baile=baile+0.00003;
   }
}

/*La clase Wave es la encargada de "dar" forma a las trayectorias
así como del carácter generativo en la aplicación. Wave es interpretado
como un solo trazo, es decir la imagen en realidad está compuesta por un
gran trazo (como dibujar sin despegar el lápiz).*/

class Wave {
  int xspacing = 2; //Inicialmente es interpretado como el espaciamiento horizontal
                    //entre las partículas, pero posteriormente representa el grado de
                    //saturación de los trazos en la imagen.

  int w;              //Tamaño del trazo o longitud de Wave.
  PVector ahora;    //Estos vectores es una manera alternativa de registrar las posiciones
  PVector antes;    //actuales y pasadas de cada partícula.

  PVector origin;           //Para indicar el punto de inicio del trazo (Wave).
  float theta = 10000.0;    //Inicia el ángulo en 10000 dado que a  partir de un valor grande
                            //los cambios en la forma son significativos.

  float amplitude;          //Amplitud de trazo
  float period;             //periodo del trazo (también es una senoidal).
  float dx;                 //Valor de incremento en X, calculado en función del periodo y
                            //espaciamiento.

  //Se inicia un sistema de partículas

  Particle[] particles;
```

```
 // Se inicia el sistema de partículas en el centro de la pantalla, largo, amplitud y periodo.

 Wave(PVector o, int w_, float a, float p) {
  antes = new PVector();
  ahora = new PVector();

  //Inicia el inicio del trazo en el centro de la ventana

  origin = o.get();
  w = w_;
  period = p;
  amplitude = a;

  //Cálculo de dx en función del periodo y espaciamiento

  dx = (TWO_PI / period) * xspacing;

  //La cantidad de partículas iniciadas es determinada por la longitud total de la onda o
  //trazo y el espaciamiento en X.

  particles = new Particle[int(w/xspacing)];
  for (int i = 0; i < particles.length; i++) {
      particles[i] = new Particle();
  }
 }


 void calculate() {

  //Al incrementar el valor de theta se puede interpretar como la velocidad angular.

    theta += 0.000000002;
```

/*En esta sección se explica el comportamiento generativo en la función:
El valor de x es aumentado lentamente conforme a la variable theta, que en cuyo caso es una especie de velocidad angular, la dx cuyo valor es muy pequeño y aumenta lentamente conforme a la suma de los valores cos(theta)+cos(x) que a su vez afectan minoritariamente el valor de x. Las variables son interdependientes, excepto theta.

Se eligió esta forma de manipular los valores de las variables para alterar los resultados obtenidos al emplear únicamente la función noise. Como resultado se tiene una variable X con un comportamiento no lineal, mientras que la amplitud de la onda si es afectada por el comportamiento de la función noise() que en cuyo caso toma el valor de la variable "baile" correspondiente a cada partícula del sistema y es aumentado 0.00003 unidades sobre sí mismo cada ciclo. El valor final que altera la amplitud es multiplicado por el factor de 10 para obtener variaciones "orgánicas" de entre 0 y 10 unidades.

Finalmente la variable x adentro de las funciones trigonométricas (no confundir con las coordenadas de puntos de origen) es multiplicada por distintos factores que pueden ser alterados y dan pie a la creación de diferentes formas a la imagen final. Como se puede observar la siguiente línea corresponde a la ecuación paramétrica de una función con una forma específica:

origin.x+(cos(x)*(amplitude+(noise(particles[i].baile)*10))),origin.y+(sin(x*6)*(amplitude+(noise(particles[i].baile)*10)))

Alterando el factor de multiplicación del sin(x*6) por sin(x), se obtiene una figura predominantemente circular:

origin.x+(cos(x)*(amplitude+(noise(particles[i].baile)*10))),origin.y+(sin(x)*(amplitude+(noise(particles[i].baile)*10)))

O alterando el factor de multiplicación del sin(x) por sin(x*2), se obtiene una figura predominantemente con geometría de infinito:

origin.x+(cos(x)*(amplitude+(noise(particles[i].baile)*10))),origin.y+(sin(x*2)*(amplitude+(noise(particles[i].baile)*10)))

Es posible emplear estas funciones para crear un sin fin de formas y geometría. */

```
    float x=theta;
    for (int i = 0; i < particles.length; i++) {

    //las posición es calculada y establecida para cada partícula del sistema

  particles[i].setLocation(origin.x+(cos(x)*(amplitude+(noise(particles[i].baile)*10))),origin.y
+(sin(x*6)*(amplitude+(noise(particles[i].baile)*10))));
        particles[i].baileSuma();
        x+=dx*20000;
        dx+=cos(theta)+cos(x);
      }
   }

 void display() {

   //El grosor debe ser inferior a un pixel para obtener el efecto de grafito.

   strokeWeight(0.5);
   stroke(0,20);
```

/*La posición de cada partícula representa un punto que junto a la posición de la partícula anterior, forma un trazo de línea recta (Se evita calcular con la primera partícula para no crear trazos "basura").*/

```
    for (int i = 0; i < particles.length; i++) {
       ahora=particles[i].location;
          if(i>0)
           {
            line(ahora.x,ahora.y,antes.x,antes.y);
           }
       antes=particles[i].location;
     }
   }
}
```

/*FIN*/

Los resultados que muestra el algoritmo son los siguientes: Las figuras 5.5 y 5.6 muestran el comportamiento con la siguiente ecuación.

origin.x+(cos(x)*(amplitude+(noise(particles[i].baile)*10))),origin.y+(sin(x*6)*(amplitude+(noise(particles[i].baile)*10)))

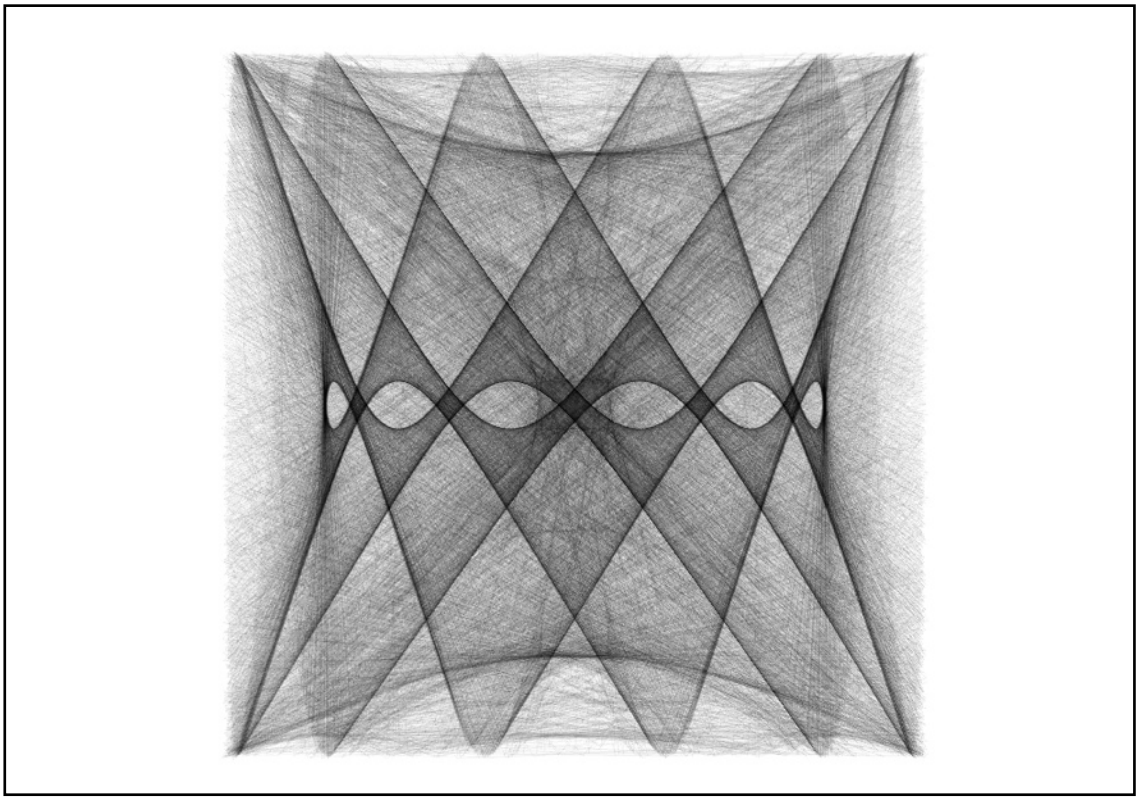


Figura 5.5

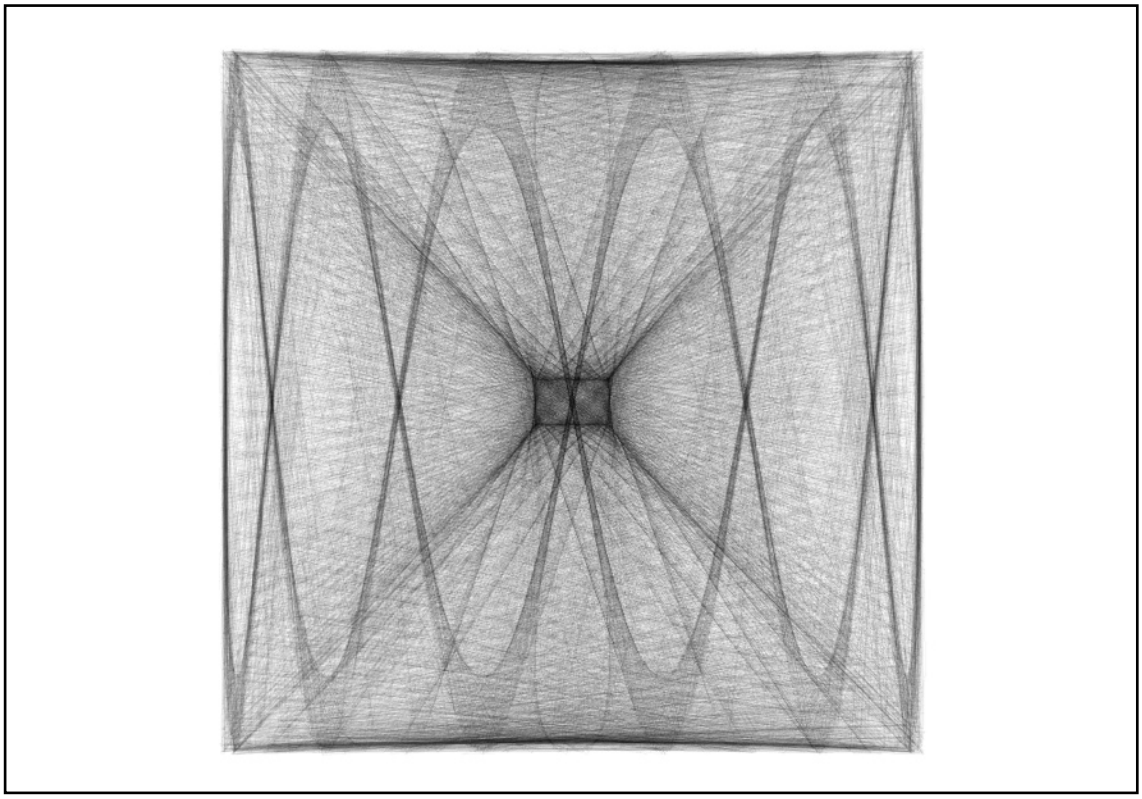


Figura 5.6

El algoritmo es capaz de mostrar una imagen totalmente diferente cada cuadro, es decir dependiendo de las capacidades de hardware de la computadora es el límite de imágenes creadas por segundo. Las figuras 5.7 y 5.8 muestran el comportamiento con la siguiente ecuación:

origin.x+(cos(x)*(amplitude+(noise(particles[i].baile)*10))),origin.y+(sin(x*2)*(amplitude+(noise(particles[i].baile)*10)))

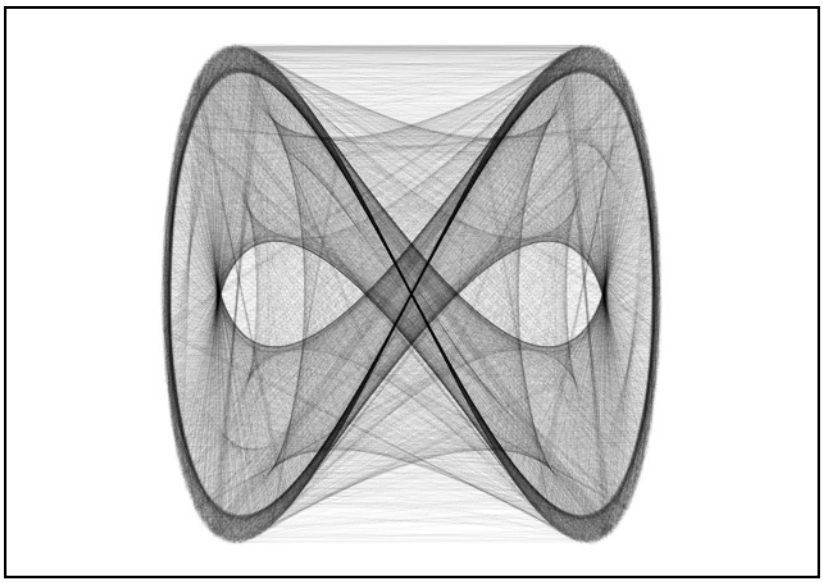


Figura 5.7

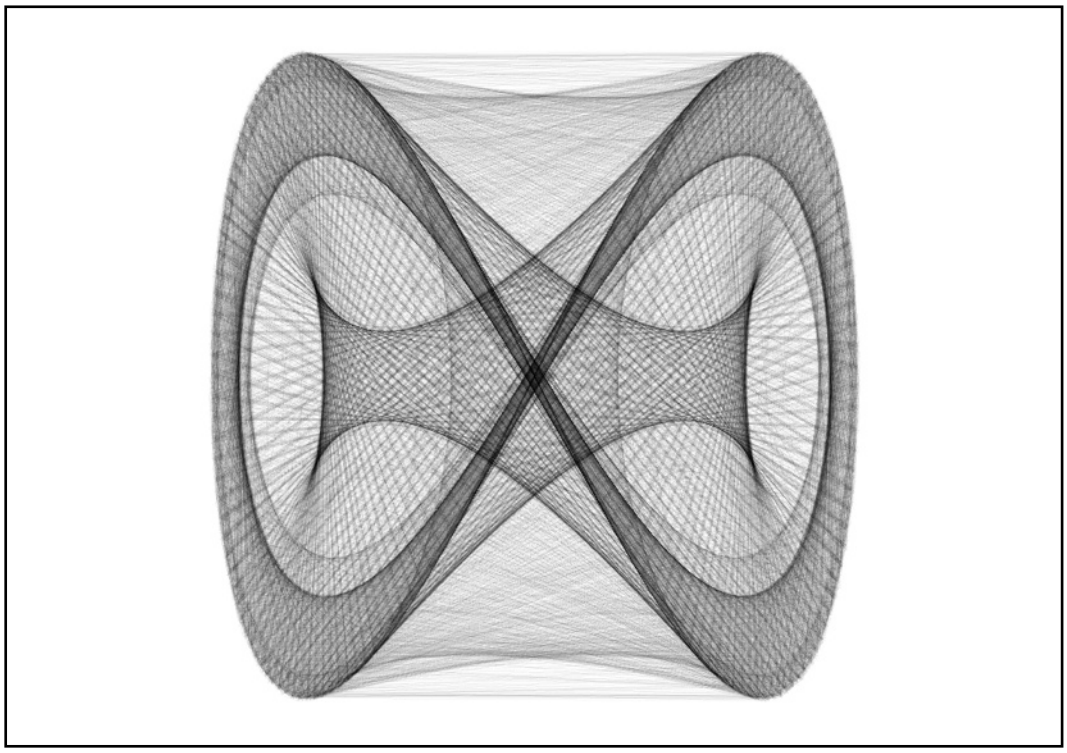


Figura 5.8

Las figuras 5.9, 5.10 y 5.11 muestran el comportamiento con diferentes ecuaciones con factores de multiplicación (1,30), (1,1) y (3,2) respectivamente.

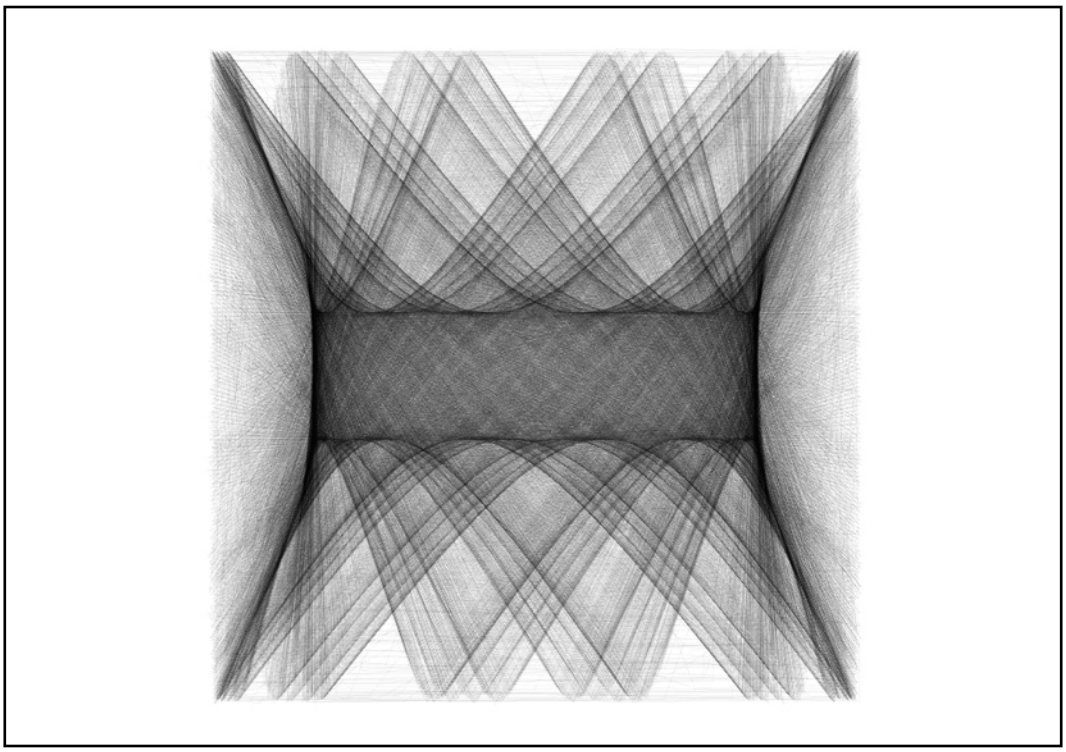


Figura 5.9

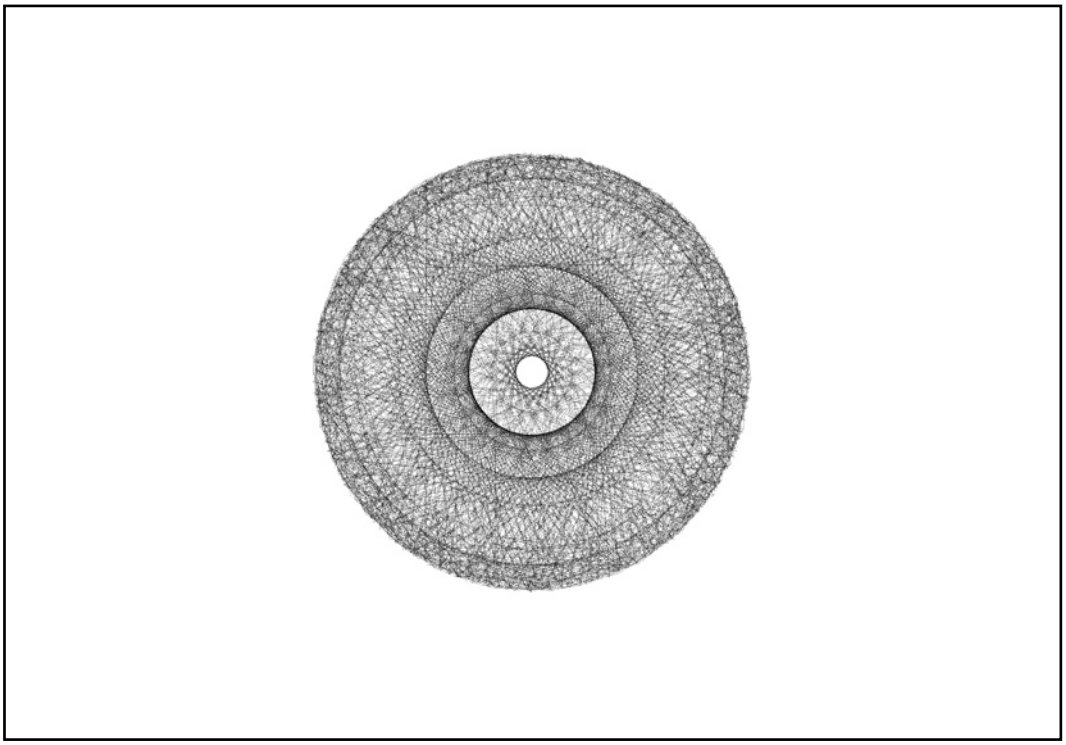


Figura 5.10

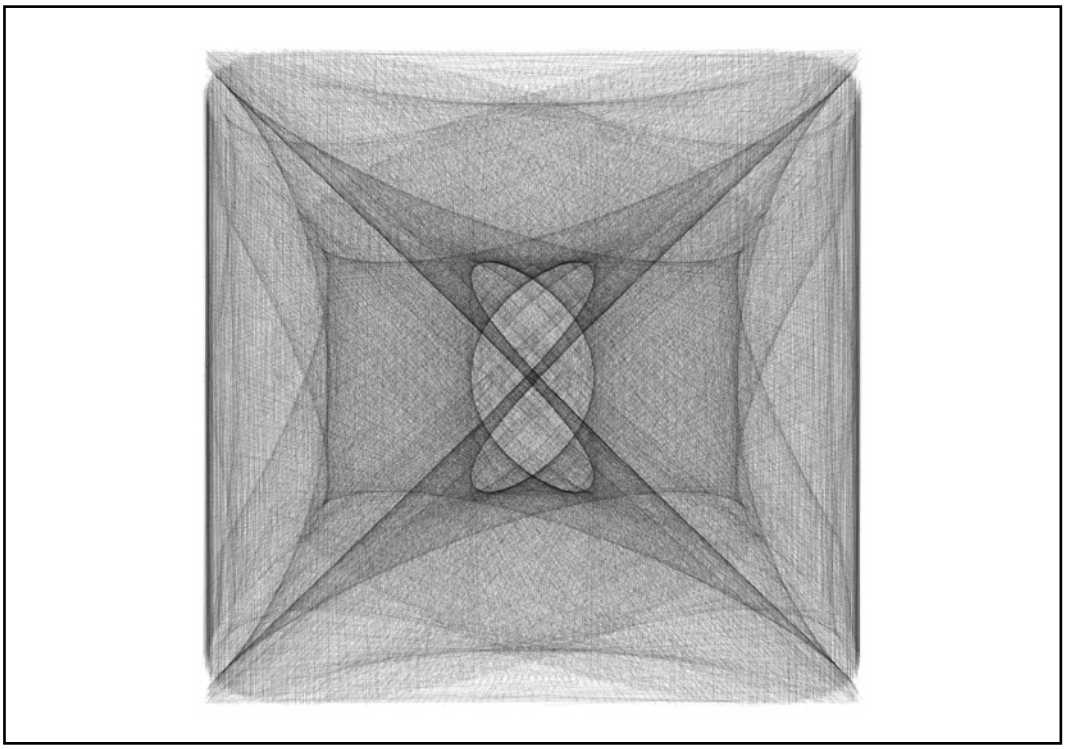


Figura 5.11

Finalmente es posible observar el comportamiento de las animaciones así como experimentos con otras ecuaciones en las siguientes direcciónes de internet: https://armyrdz.wordpress.com/2016/07/08/procedural-animations/ y https://armyrdz.wordpress.com/2016/07/08/vaiduag/.

5.1.3 Partículas y ecuaciones paramétricas en 2D, animaciones

La siguiente aplicación demuestra que pequeñas variaciones en el algoritmo, pueden modificar ampliamente el comportamiento de éste, así como los efectos visuales producidos. Es posible crear a partir del algoritmo generativo explicado en la aplicación anterior, una basta cantidad de variaciones.

Ahora se trata de un enfoque distinto, en la aplicación anterior se demostró que el algoritmo es capaz de crear formas a partir de números o valores "pseudo-aleatorios" pero generados mediante un sistema generativo. Se definió un "trazo" u onda la cual toma dichos valores y dibuja una figura que con cada ciclo en el programa, es modificada drásticamente creando una animación la cual puede llegar a ser carente de sentido bajo ciertas condiciones (influye el grado de las ecuaciones empleadas así como el factor de multiplicación elegido, se demostró que con las ecuaciones con factores de multiplicación enteros y de valor inferior a 5, es posible percibir una animación entendible, después de este umbral el sistema continúa generando figuras pero no una animación, ya que las imágenes desplegadas en cada cuadro del proceso difieren en gran medida unas de otras).

Ahora el enfoque de la aplicación es crear una animación entendible a pesar de modificar ciertas variables. Esto se ha conseguido implementando dos conceptos: El primero, considerar al sistema de partículas como un lazo u onda dirigido a través de un espiral el cual rota sobre su punto medio de origen. Como se conoce, para representar un espiral es necesario describir por medio de una ecuación, la cual describe puntos de una circunferencia que con cada iteración el radio aumenta una pequeña fracción. Empleando lo anterior, se da pie el siguiente concepto: el radio de cada punto que conforma el espiral es alterado por medio del algoritmo explicado en la aplicación anterior, pero con la diferencia que cada punto poseerá el valor de su antecesor en el siguiente ciclo. En una primera fase de implementación del algoritmos, se unieron los trazos, dando origen a una animación simple que simula "el movimiento de una burbuja de jabón en dos dimensiones¨ (figura 5.12).

Además se han implementado un par de funciones referentes al tema de interacción: primero se puede hacer uso de las flechas del teclado (arriba y abajo) y el mouse (arrastrar de derecha - izquierda) para manipular el grado de la ecuación y factor de multiplicación y como consecuencia modificar el comportamiento de la animación así como su forma.

Figura 5.12

```
/*Se Declaran tres objetos del tipo Wave, es posible implementar
mas de un sistema de partículas gracias a la clase*/

Wave wave0;
Wave wave1;
Wave wave2;

//Es necesario pre-cargar la textura correspondiente a las partículas
PImage img;
float forma = 0;

void setup() {
  background(255);

  //Al implementar P2D, es usado el modo de render por OPENGL
  //lo cual permite un framerate mayor al ser implementado

  fullScreen(P2D);

  // Se inicia el sistema de partículas en el centro de la
  //pantalla, largo, amplitud y periodo.

  wave0 = new Wave(new PVector(width/2,height/2),35500,300,4050);
  wave1 = new Wave(new PVector(width/3,height/3),35500,500,4050);
  wave2 = new Wave(new PVector(width/1.5,height/1.5),35500,200,4050);
}

void draw() {
```

```
  //Se utiliza un rectángulo transparente como fondo de la ventana
  //permitiendo crear efecto barrido.

  rectMode(CORNER);
  fill(230,140);rect(0,0,width,height);

  //método que calcula la posición de la partículas

  wave0.calculate();

  //método que despliega la imagen

  wave0.display();
  wave1.calculate();
  wave1.display();
  wave2.calculate();
  wave2.display();

 // saveFrame("enjambre-######.png");

}


void keyPressed() {

  //Controles por medio del teclado (arriba y abajo), aumenta o disminuyen la
  //variable forma, la cual permite modificar la figura final

  if (keyCode == UP) forma += 1;
  if (keyCode == DOWN ) forma -= 1;
}



class Wave {

  int xspacing = 2; //Inicialmente es interpretado como el espaciamiento horizontal
                    //entre las partículas, pero posteriormente representa el grado de
                    //saturación de los trazos en la imagen.

  int w;              //Tamaño del trazo o longitud de Wave.
  PVector ahora;    //Estos vectores es una manera alternativa de registrar las posiciones
  PVector antes;     //actuales y pasadas de cada partícula.

  PVector origin;           //Para indicar el punto de inicio del trazo (Wave).
  float theta = 10000.0;   //Inicia el ángulo en 10000 dado que apartir de un valor grande
                           //los cambios en la forma son significativos.
```

```
float amplitude;        //Amplitud de trazo
float period;           //periodo del trazo (también es una senoidal).
float dx;               //Valor de incremento en X, calculado en función del periodo y
                        //espaciamiento.

//Se inicia el sistema de partículas

Particle[] particles;

// Se inicializa el sistema de partículas en el centro de la
//pantalla, largo, amplitud y periodo.

Wave(PVector o, int w_, float a, float p) {
  antes = new PVector();
  ahora = new PVector();
  origin = o.get();
  w = w_;
  period = p;
  amplitude = a;

  //Cálculo de dx en función del periodo y espaciamiento

  dx = (TWO_PI / period) * xspacing;

  //La cantidad de partículas iniciadas es determinada por la longitud total de la onda o
  //trazo y el espaciamiento en X.

  particles = new Particle[int(w/xspacing)];
  for (int i = 0; i < particles.length; i++) {
      particles[i] = new Particle();
      }
  }


void calculate() {

  //Al incrementar el valor de theta se puede interpretar como la velocidad angular.

  theta += 0.002;
```

/*En esta sección se explica el comportamiento generativo en la función:
El valor de x es aumentado lentamente conforme a la variable theta, que en cuyo caso es una especie de velocidad angular. La variable x posteriormente es afecta de forma mínima debido a que dx es pequeño, esta variable es un factor de multiplicación que afecta lentamente al sistema.

El valor de la posición del mouse, visto como mouseX ayuda a crear una animación dinámica. Por otro lado la variable "forma" al ser modificada por las acciones del usuario por medio del teclado permite modificar la forma de la figura final en tiempo real.

Finalmente la funcíón noise() modifica el el radio en la ecuación (dictado por la amplitud) y al mismo tiempo x es la variable que origina el comportamiento generativo, ya que como se ha mencionado aumenta lentamente y es ideal para modelar dicho comportamiento.
*/

```
    float x=theta;
    float baile2;
    for (int i = 0; i < particles.length; i++) {
                              particles[i].setLocation((origin.x+(cos((width/(mouseX
+0.1))*x*2)*amplitude*noise(x)*2))+(noise(x)*100),(origin.y+(sin((width/(mouseX
+0.1))*x*forma)*amplitude*noise(x)*2))+(noise(x*100)*100));
       x+=(dx*100)+dx;
      }
    }

 void display() {
   for (int i = 0; i < particles.length; i++) {
      ahora=particles[i].location;
        if(i>0)
         {

         //De acuerdo a la posición en el arreglo de la partícula, se modifican
         //sus propiedades de color.

         if(i%7==0){
           imageMode(CENTER);
           tint(255,255);
           //image(img,ahora.x,ahora.y);
           }
         if(i%17==0){
           imageMode(CENTER);
           tint(55,255);
           image(img,ahora.x,ahora.y);
           }
         if(i%9==0){
           imageMode(CENTER);
           tint(200,255);
           //image(img,ahora.x,ahora.y);
           }
         }
      antes=particles[i].location;
    }
   }
}
```

En las figuras 5.13, 5.14. 5.15 y 5.16 es posible observar capturas de pantalla correspondientes a distintas animaciones producidas por la interacción del usuario. Para tener una mejor percepción del efecto producido, es posible visitar la siguiente dirección de internet para observar en video la aplicación: https://armyrdz.wordpress.com/2016/07/22/mdt33-exe/.

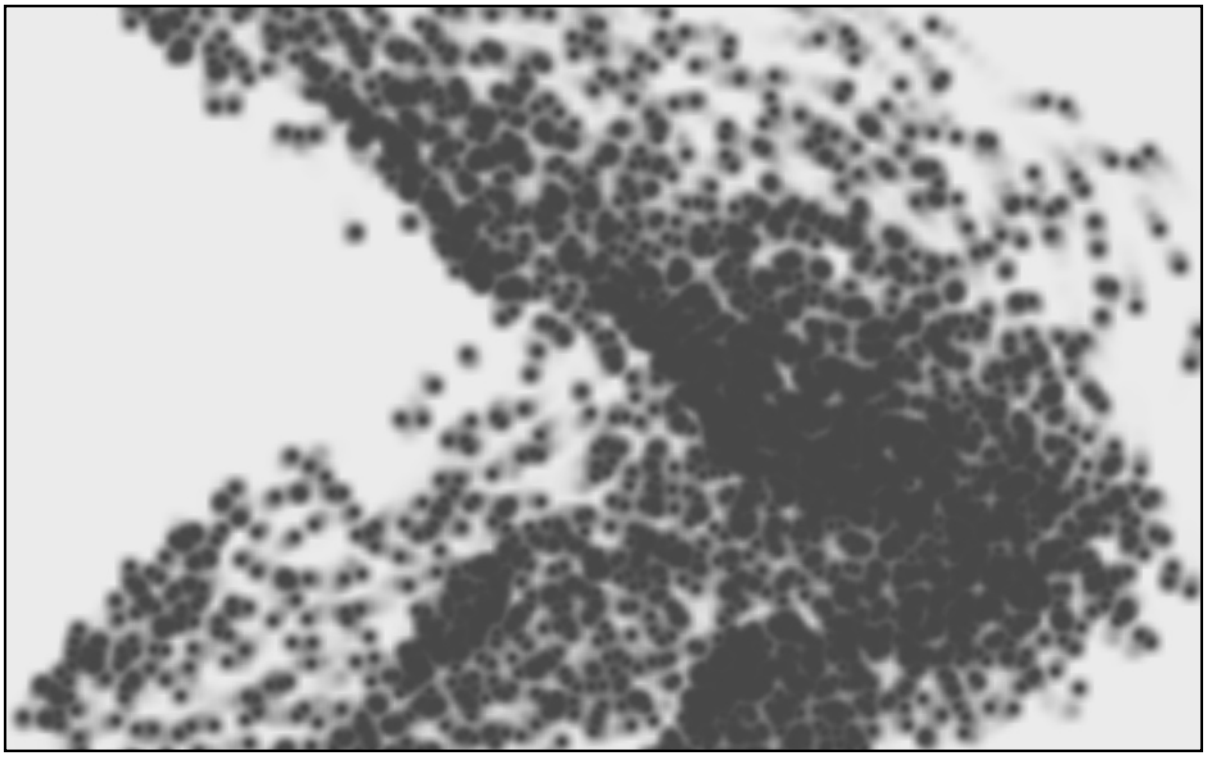


Figura 5.13

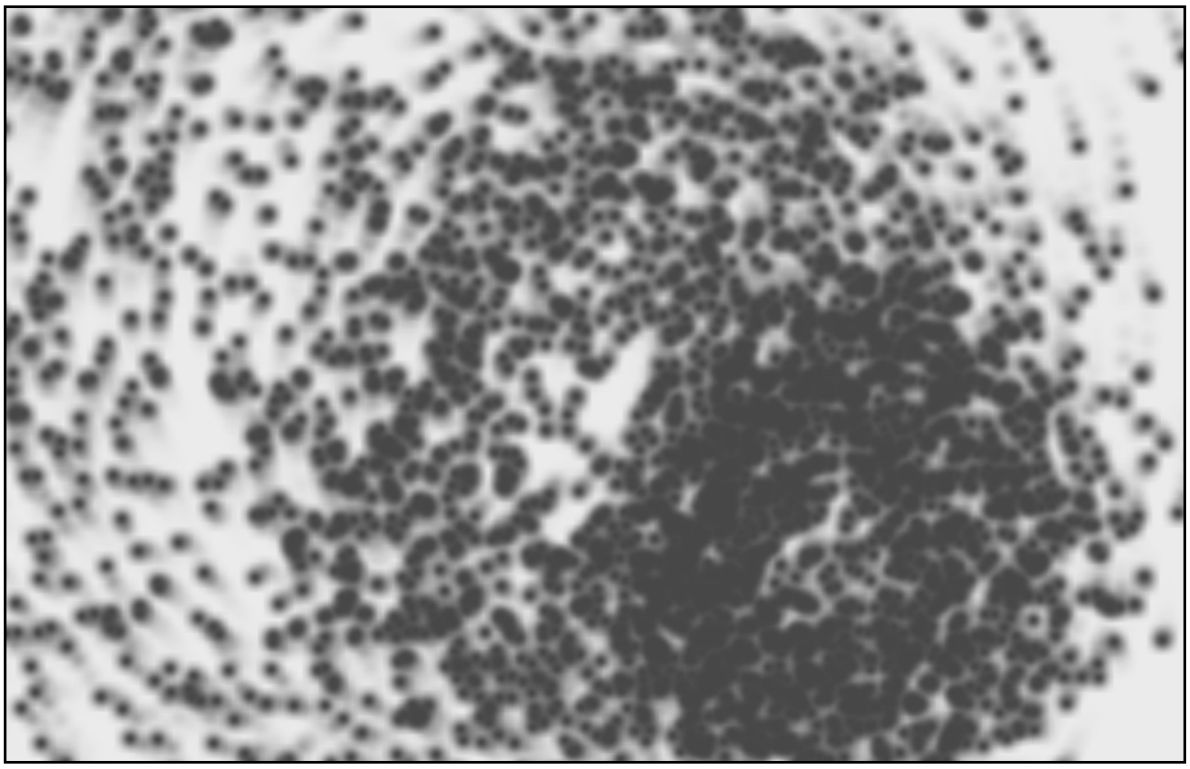


Figura 5.14

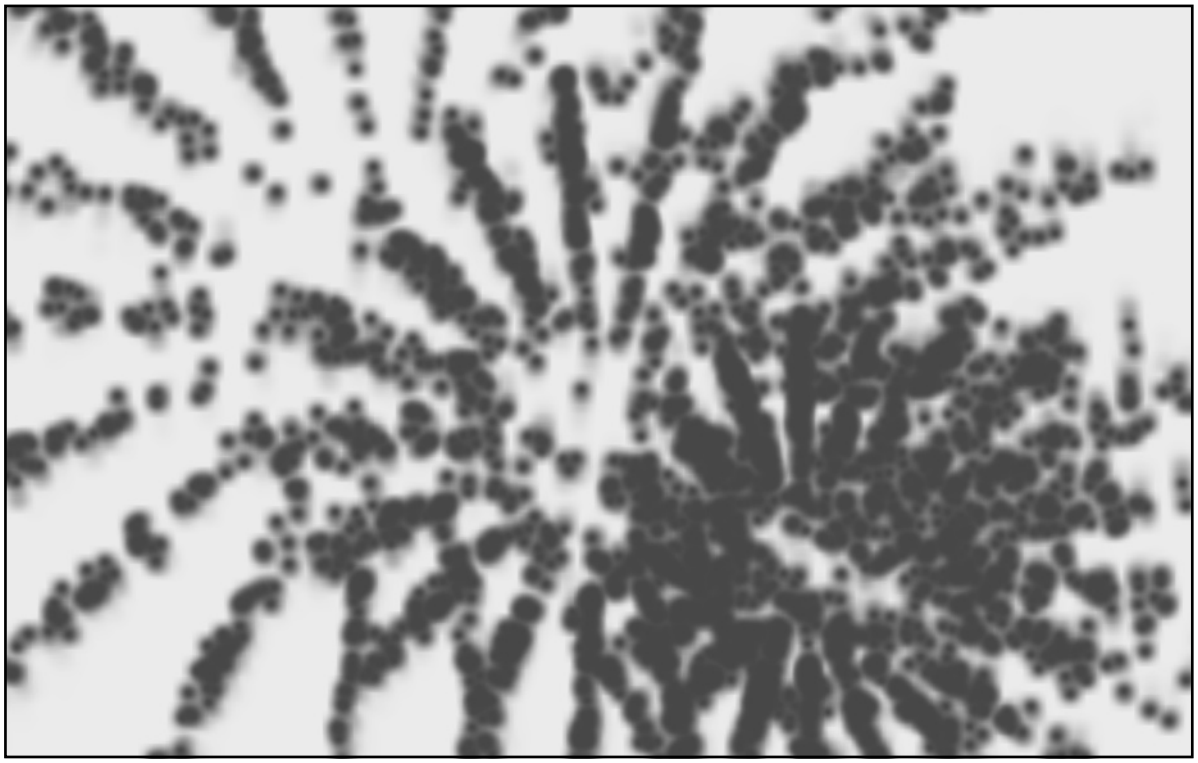


Figura 5.15

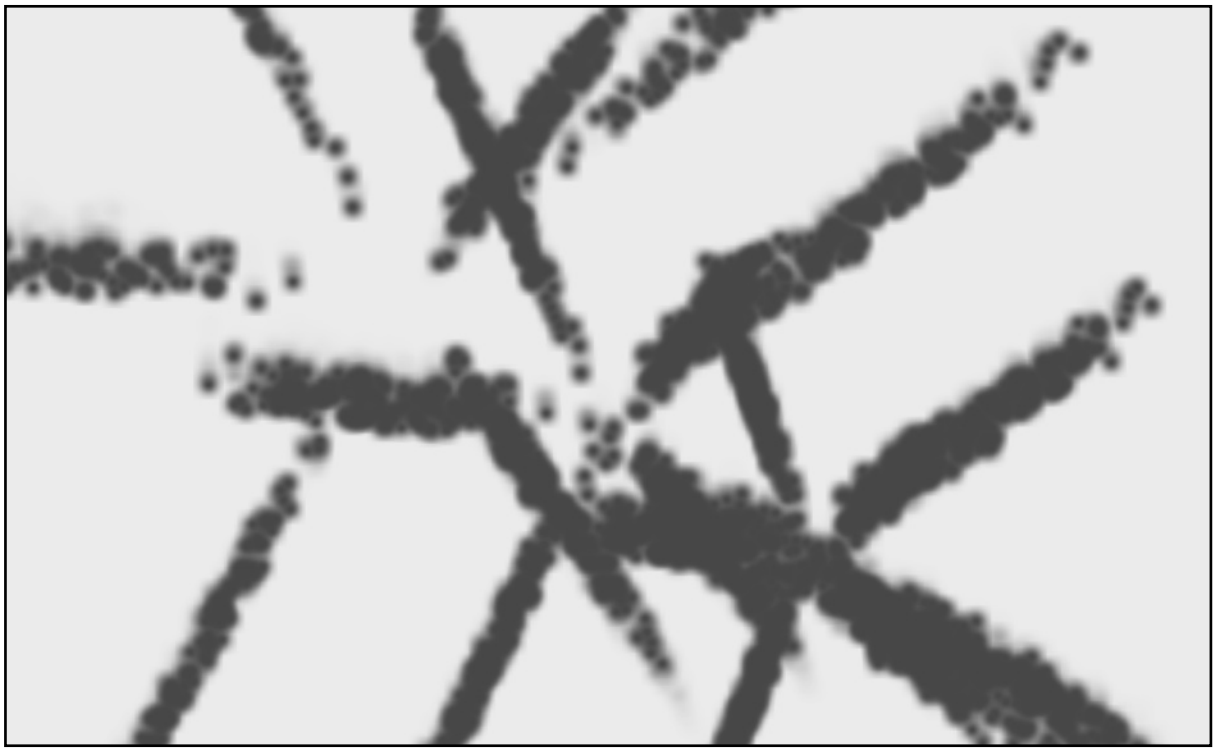


Figura 5.16

5.1.4 Agentes

La aplicación está basada inicialmente en los trabajos de Hartmut Bohnacker, Benedikt Grob y Julia Laub en su publicación: *"Generative Design*". Para ser precisos retoma los principios expuestos la sección de agentes "dumb" y agentes inteligentes, en el caso original "se plantea un circulo como la forma inicial, posteriormente se plantea que cada punto del círculo es representado por un agente y el movimiento de los puntos causará gradualmente que la forma cambie, ocasionando un amplio repertorio de formas"[23].

A continuación se presenta el mismo principio para la siguiente aplicación la cual es implementada con propósitos interactivos así como exploración de formas a través de estos algoritmos que permiten ahorrar altos recursos de cómputo a diferencia de los requeridos para generar modelos en tercera dimensión por medio software especializado.

Otro elemento retomado es la implementación de la biblioteca especializada en generar archivos pdf de gran formato, con la finalidad de exportar la imágenes generadas y ser impresas o proyectadas en algún otro dispositivo (*import processing.pdf.**).

```
/*INSTRUCCIONES//////////////////////////////////////////////////////////////
  Oprimir tecla "s" o "S" para guardar un frame
  Oprimir tecla "Delete" o "Backspace" para reiniciar la aplicación
  Oprimir tecla "V" o "v" para iniciar la captura de cuadros
  Oprimir tecla "R" o "r" para iniciar grabación de PDF
  Oprimir tecla "X" o "x" para cambio de forma inicial
  Oprimir tecla "Y" o "y" para cambio de forma inicial
  Oprimir tecla "Z" o "z" para cambio de forma inicial
  Oprimir tecla "e" o "E" para terminar grabación de PDF

  Arrastrar el mouse a través de la ventana para modificar el color.

//////////////////////////////////////////////////////////////////////////*/
```

```
Como se ha mencionado se ha optado por implementar la biblioteca especializada
en pdf para posteriores impresiones*/

import processing.pdf.*;
import java.util.Calendar;

//Las variables bool que hacen referencia a grabar permiten capturar los cuadros de
//la animación y posteriormente si se desea formar con ellos una video.

boolean recordPDF = false;
boolean record = false;

//Resolución de la forma así como la separación entre puntos

int formResolution = 1500;
float stepSize = 0.5;

//Radio inicial de las figuras
float initRadius = 150;

//Declaración del centro de las figuras y arreglos que contienen las información de los
//puntos o agentes en el espacio.

float centerX, centerY, centerZ;
float[] x = new float[formResolution];
float[] y = new float[formResolution];
float[] z = new float[formResolution];


void setup(){
  size(displayWidth, displayHeight,P3D);
  smooth();

  /*Se establecen las características iniciales de la figura.
  Los ángulos declarados permiten generar figuras en tras dimensiones mediante
  sus ecuaciones paramétricas.*/

  centerX =0;
  centerY = 0;
  centerZ = 0;
  float angle = radians(20/float(formResolution));
  float angle2 = radians(720/float(formResolution));
  for(int i=0; i<formResolution; i++){
     x[i] = sin(angle2*i*100) * cos(angle*i*100) * cos(angle*i*100) * initRadius;
     y[i] = (sin(angle2*i*100) * sin(angle*i*100) * initRadius);
     z[i] = cos(angle2*i*100) * initRadius;
   }
  stroke(255, 10);
```

```
  background(0);
}


void draw(){

   //Una cámara es configurada para observar la animación y rotación de la figura

   float zm = 250;
   float sp = 0.01 * frameCount;
   camera(zm* sin(sp), zm* cos(sp) , zm * sin(sp), 0, 0, 0, 0, 0, -1);

   //El radio de los agentes es modificado en este caso, de manera aleatoria.

   initRadius+= random(-stepSize,stepSize);

   //La posición de los puntos o agentes es modificada de acuerdo al factor aleatorio

   for(int i=0; i<formResolution; i++){
     x[i] += random(-stepSize,stepSize);
     y[i] += random(-stepSize,stepSize);
     z[i] += random(-stepSize,stepSize);
    }

   //Nuevo esquema de color HSB, es útil para modificar el color empleando interacción
   //ya que el matiz depende de un solo valor y no tres como el caso de RGB.

   colorMode(HSB, 350, 100, 100);
   stroke(0,0,0, 5);
   strokeWeight(5);

   //Dibuja un punto en la posición del agente.

   for(int i=0; i<formResolution; i++){
       point(x[i]/2+centerX, y[i]/2+centerY,z[i]/2+centerZ);
     }

   //Se establecen las características de trazo

   strokeWeight(0.75);
   noFill();




   /*Cuando una forma es dibujada mediante la función  curveVertex(), el primer valor
```

```
   y el último no son dibujado ya que se trata de puntos de control. Estos puntos de control
   aseguran sea exactamente completado*/

   beginShape();
   //Inicio del punto de control

curveVertex(x[formResolution-1]+centerX, y[formResolution-1] +centerY,
z[formResolution-1] + centerZ);

      //Solo estos puntos son dibujados

      for(int i=0; i<formResolution; i++){
         curveVertex(x[i]/1.5+centerX, y[i]/1.5+centerY,z[i]/1.5+centerZ);
        }
      curveVertex(x[0]+centerX, y[0]+centerY,z[0]+centerZ);

      //Fin del punto de control

      curveVertex(x[1]+centerX, y[1]+centerY,z[1]+centerZ);
      endShape();

      //Otra figura pero con diferente escala es dibujada con el mismo principio.

      stroke(0,0,0, 5);
      strokeWeight(0.75);
      noFill();

      beginShape();
                     curveVertex(x[formResolution-1]+centerX,  y[formResolution-1]+centerY,
z[formResolution-1] + centerZ);

      for (int i=0; i<formResolution; i++){
      curveVertex(x[i]+centerX, y[i]+centerY,z[i]+centerZ);
        }
      curveVertex(x[0]+centerX, y[0]+centerY,z[0]+centerZ);

      curveVertex(x[1]+centerX, y[1]+centerY,z[1]+centerZ);
   endShape();

   /*Un cubo que rota con centro en (0,0,0), es trasparente y son bordes para
   crear el efecto barrido en espacios 3D, esto es similar al rectángulo transparente
   implementado anteriormente.*/

   pushMatrix();
   rectMode(CENTER);
   noStroke();
   fill(0,0,100,10);
   box(5000);
```

```
    rotateX(sp*10000);
    noFill();

    //Grabación de cuadros

    popMatrix();
    if(record){
      saveFrame("screen-####.jpg");
     }
}



//////////////////////INTERACCIÓN//////////////////////////


void keyReleased() {

  //Oprimir tecla "s" o "S" para guardar un frame

  if (key == 's' || key == 'S') saveFrame(timestamp()+"_##.png");

  //Oprimir tecla "Delete" o "Backspace" para reiniciar la aplicación

  if (key == DELETE || key == BACKSPACE) background(255);

  //Oprimir tecla "V" o "v" para iniciar la captura de cuadros

  if (key =='V' || key =='v') {
     record = true;
    }

  //Oprimir tecla "R" o "r" para iniciar grabación de PDF

  if (key =='r' || key =='R') {
    if (recordPDF == false) {
      beginRecord(PDF, timestamp()+".pdf");
      println("recording started");
      recordPDF = true;
      stroke(0, 50);
    }
  }



 if (key =='X' || key =='x') {
```

```
//Oprimir tecla "X" o "x" para cambio de forma inicial

centerX =0;
centerY = 0;
centerZ = 0;
float angle = radians(20/float(formResolution));
float angle2 = radians(720/float(formResolution));
for (int i=0; i<formResolution; i++){
  x[i] = sin(angle2*i*100) * sin(angle2*i*200) * cos(angle*i*100) * initRadius;
  y[i] = (sin(angle2*i*100) * sin(angle*i*100) * initRadius);
  z[i] = cos(angle2*i*100) * initRadius;
  }
}

if (key =='Y' || key =='y') {

//Oprimir tecla "Y" o "y" para cambio de forma inicial

centerX =0;
centerY = 0;
centerZ = 0;
float angle = radians(20/float(formResolution));
float angle2 = radians(720/float(formResolution));
for (int i=0; i<formResolution; i++){
  x[i] = sin(angle2*i*100) * sin(angle2*i*200) * cos(angle*i*100) * initRadius;
  y[i] = (sin(angle2*i*200) * cos(angle*i*100) * cos(angle2*i*100) *initRadius);
  z[i] = cos(angle2*i*100) * initRadius;
  }
}


if (key =='Z' || key =='z') {

//Oprimir tecla "Z" o "z" para cambio de forma inicial

centerX =0;
centerY = 0;
centerZ = 0;
float angle = radians(20/float(formResolution));
float angle2 = radians(720/float(formResolution));
for (int i=0; i<formResolution; i++){
   x[i] = sin(angle2*i*100) * sin(angle2*i*100) * cos(angle*i*100) * initRadius;
   y[i] = (sin(angle2*i*200) * cos(angle*i*100) * cos(angle2*i*100) *initRadius);
   z[i] = sin(angle*i*100) * cos((angle2*i*100)+2)* initRadius;
   }
}
```

```
  //Oprimir tecla "E" o "e" para terminar grabación de PDF

  else if (key == 'e' || key =='E') {
    if (recordPDF) {
      println("recording stopped");
      endRecord();
      recordPDF = false;
      background(255);
    }
  }

}

//Captura de fecha
String timestamp() {
  Calendar now = Calendar.getInstance();
  return String.format("%1$ty%1$tm%1$td_%1$tH%1$tM%1$tS", now);
}
```

//FIN//

Las imágenes 5.17, 5.18, 5.19 y 5.20 muestran los resultados obtenidos.

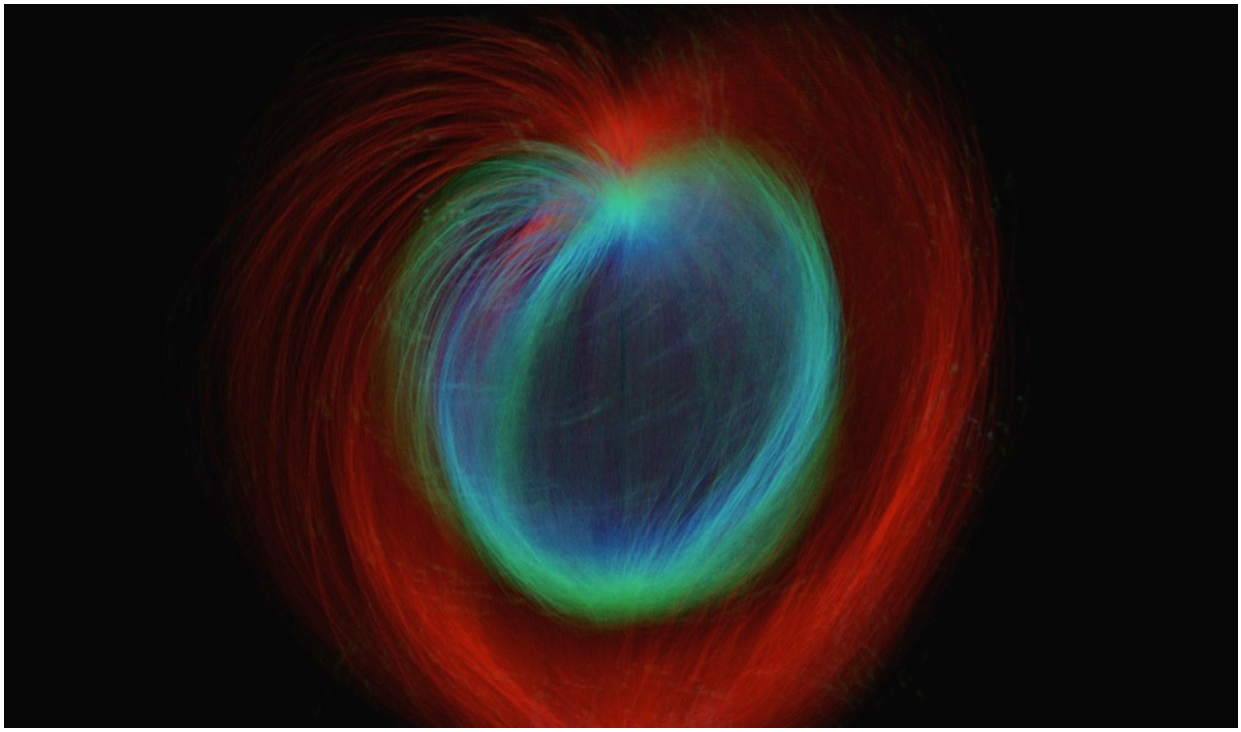


Figura 5.17

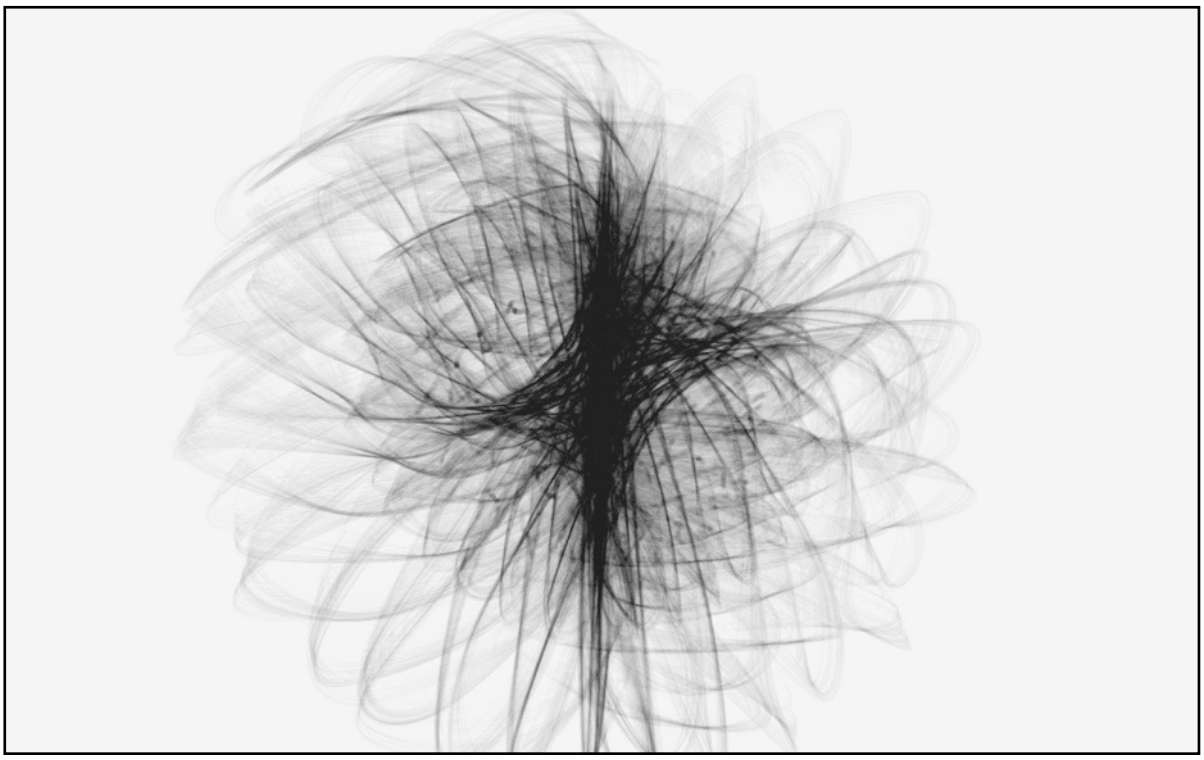


Figura 5.18

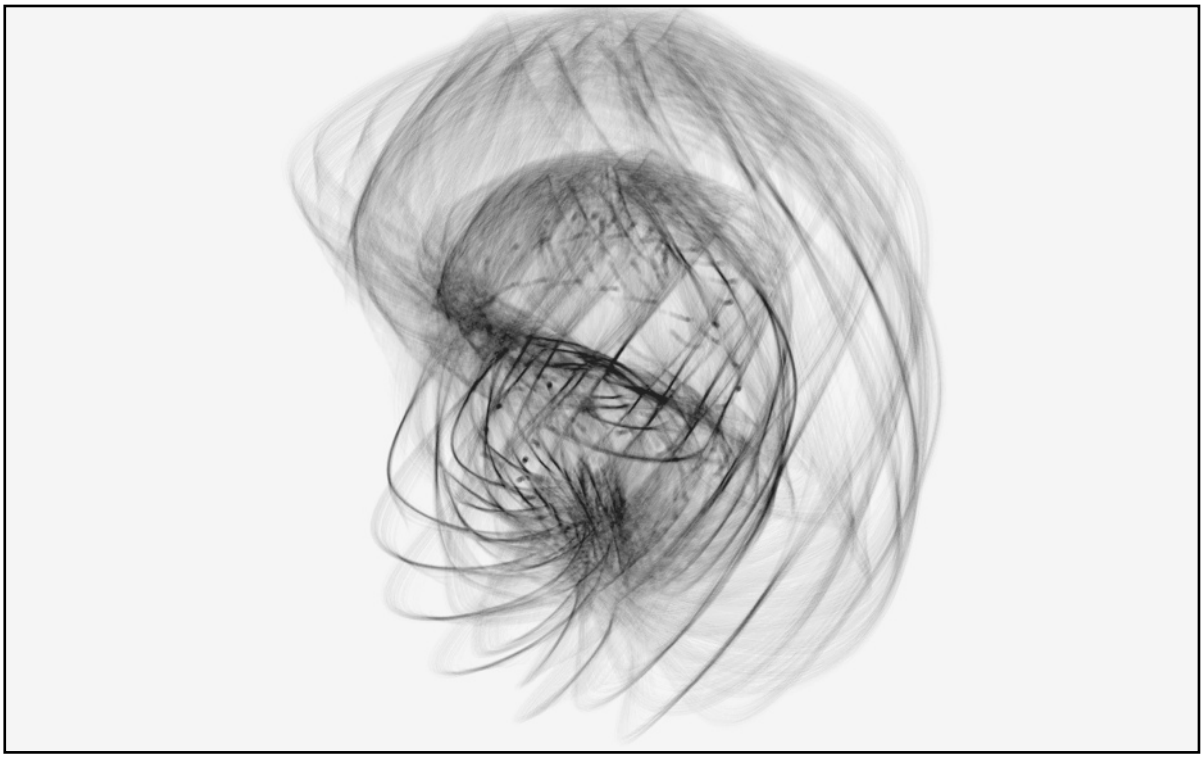


Figura 5.19

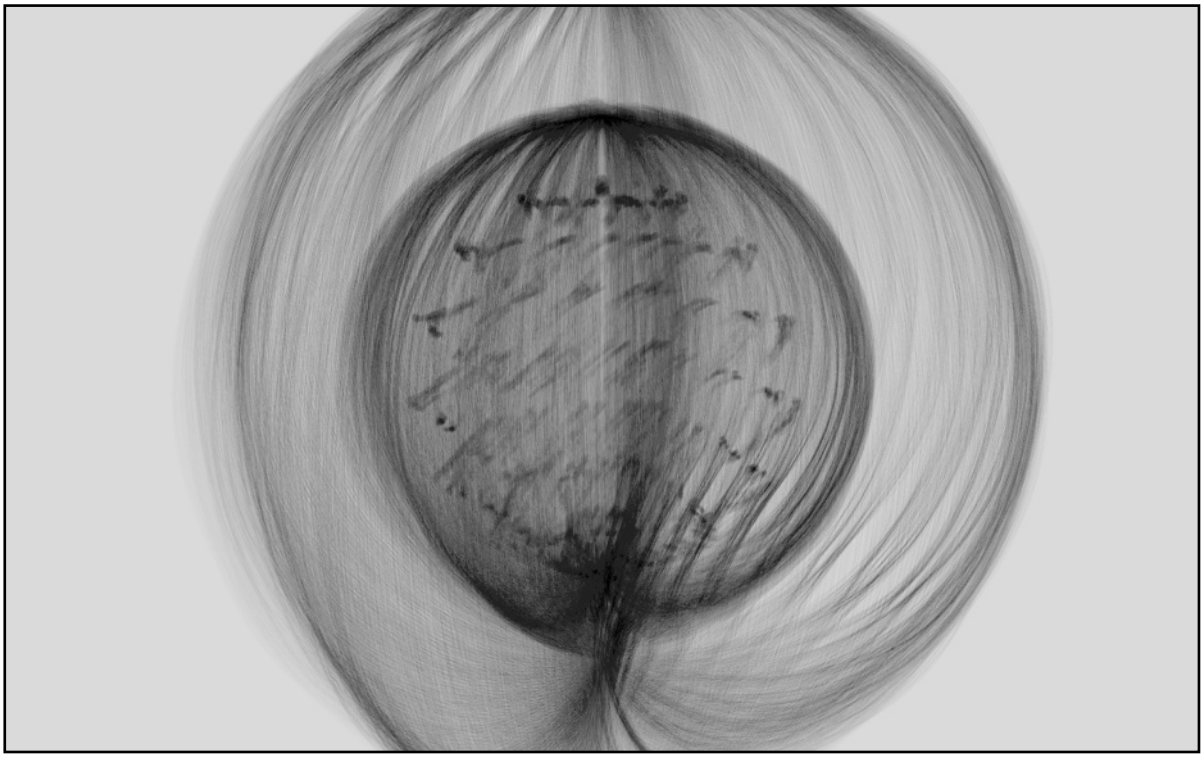


Figura 5.20

Las visualizaciones en video pueden encontrarse en las siguientes ligas de internet: https://armyrdz.wordpress.com/2016/09/11/orb422/, https://armyrdz.wordpress.com/2016/09/09/orb717/ y https://armyrdz.wordpress.com/2016/09/08/orbbit/.

5.1.5 Atractores

Como su nombre lo indica, los atractores son una especie de imanes o elementos con gravedad virtuales capaces de atraer o repeler otros objetos. “Su función está basada en el tiempo; cada iteración en el programa, todos los cuerpos en el sistema sufren un efecto mínimo debido a la fuerza que generan dichos elementos. Se ha comprobado que algunas formas únicamente pueden generarse mediante esta manera (caso contrario se vuelve sumamente complicado).”[24]

En un mundo virtual en el cual las fuerzas de atracción o repulsión son simuladas, son necesarios al menos dos objetos: atractores; puntos que atraen o repelen y objetos que son sujeto a la fuerza del primero, en algunos casos son referidos como nodos. En esencia los atractores y nodos son simplemente puntos en el espacio. La información mas importante de estos es su posición (coordenadas).

De igual manera un objeto que se puede mover es entendido como un nodo. Adicionalmente a su posición en el espacio, este también posee características como velocidad y aceleración. En el cómputo gráfico, la velocidad se entiende como un vector el cual especifica el número de pixeles que recorre la posición en cada ciclo del programa. Por otro lado la aceleración es el cambio de la velocidad en cada ciclo del programa.

Un atractor simula las fuerzas de atracción. Como sucede con los planetas o los imanes, entre más cerca estén los nodos del atractor, mayor será la fuerza que experimenten. Al igual que con los nodos, los cálculos de la fuerzas se realizan paso a paso entorno a las características que rodean al atractor.

Para la siguiente aplicación se han retomado las ideas de Daniel Shiffman y su publicación "*Nature of Code"* en el cual se detalla como implementar un sistema de fuerzas y posteriormente un sistema que incluya atracción entre los objetos. Inicialmente se plantean para los objetos del sistema lo siguiente:

1. Cada objeto tiene una posición el cual es interpretado como un vector: PVector location1, PVector location2, PVector location3, ... , PVector locationN.
2. Cada objeto tiene masa: float mass1, float mass2, float mass3, ... , float massN.
3. Existe la variable float G que representa la constante de gravedad universal.

Una vez tenido el conocimiento de lo anterior, la fuerza de gravedad es computada como un vector: PVector force, primero se calculará la dirección de la fuerza y posteriormente la intensidad de ésta de acuerdo a la masas y las distancias de los objetos.
Un vector es la diferencia entre dos puntos, para ejemplificar un vector entre dos puntos podemos definir:

```
PVector dir = PVector.sub(locationAttractor, locationNode);
```

En este caso la dirección de la fuerza de atracción entre dos objetos se define como:

```
PVector dir = PVector.sub(locationAttractor, locationNode);
dir.normalize();
```

Para obtener un vector unitario que denote la dirección de la fuerza, es necesario normalizar el vector después de sustraer las posiciones. Ahora es necesario computar los vectores de magnitud y escala, se necesitará proceder de la siguiente manera:

```
float m = (G * massAttractor * massNode) / (distance * distance);
dir.mul(m);
```

Para obtener la distancia entre ambos objetos es importante regresar un poco y definir la distancia entre ambos justo antes de normalizar el vector de dirección.

```
float distance = force.magnitude();
```

Finalmente el fragmento encargado de calcular la dirección y magnitud de la fuerza quedará de la siguiente forma[25]:

```
PVector dir = PVector.sub(locationAttractor, locationNode);
float distance = force.magnitude();
float m = (G * massAttractor * massNode) / (distance * distance);
dir.normalize();
dir.normalize();
```

La siguiente aplicación implementa los principios mencionados anteriormente. Esto ha permitido desarrollar sistemas en dos y tres dimensiones, así como múltiple objetos que interactuan entre sí para generar diferentes texturas. Cabe señalar que hasta este punto se han explicado aplicaciones con una carga gráfica y con un mínimo de intervención interactiva, más adelante se agregará este elemento para aumentar la complejidad de las aplicaciones así como el reto que representa concebirlas y englobarlas en una metodología. Por ahora la única interacción presente en esta aplicación es por medio del mouse.

Para el caso de esta aplicación se ha implementado un sistema de 1000 nodos y 20 atractores en dos y tres dimensiones respectivamente. Estos número no son inamovibles permitiendo generar nuevas formas de acuerdo a como el usuario lo desee. Finalmente las posiciones iniciales de los atractores y los nodos son colocados de manera aleatoria de tal modo que se decremente la posibilidad de figuras repetidas por cada ejecución de la aplicación.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/*Esta sección corresponde a la declaración de las características del sistema así como de los
objetos que lo componen*/

////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//Los siguientes arreglos corresponden al número de atractores y nodos del sistema
Mover[] movers = new Mover[1000];
Attractor[]  a = new Attractor[20];

void setup() {
  size(1200, 800, P3D);
  smooth();
  background(0);

  //Posicionamiento de la cámara en el espacio 3D

  camera(400, 0, 6000, (width/2), (height/2), 200,  -6.0, 0.0, -1.0);

  //Se inician los nodos

  for (int i = 0; i < movers.length; i++) {
    movers[i] = new Mover(1, random(width), random(height));
  }

  //Los atractores son iniciados

  for (int i = 0; i < a.length; i++) {
  a[i] = new Attractor();
  }
}

void draw() {
   colorMode(HSB, 1200, 800, 100);

   //Para cada nodo se aplica la fuerza ejercida de cada atractor

    for(int j = 0; j < a.length; j++){
        for (int i = 0; i < movers.length; i++) {
            PVector force = a[j].attract(movers[i]);
            movers[i].applyForce(force);
            movers[i].update();
            movers[i].display();
          }
        }
 }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

/*Esta sección representa la clase y los métodos correspondientes a los atractores, este fragmente es el que se ha retomado en gran medida de los trabajos de Daniel Shiffman en su publicación “The Nature of Code”*/

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class Attractor {

 // Son declaradas las características de cada atractor

 float mass;    // masa
 float G;       // Contante de Gravedad
 PVector location;   //Posición

 Attractor() {

  //La posición de los atractores de define de forma aleatoria

  location = new PVector(random(0,width),random(0,height),random(-15,15));
  mass = random(39,40);
  G = 1;
  }

 PVector attract(Mover m) {
  PVector force = PVector.sub(location,m.location);   // Calcular dirección de la fuerza
  float d = force.mag();                              // Distancia entre los objetos

  // La distancia se limita para evitar resultados extremos entre elementos muy lejanos
  // o muy cercanos entre si.

  d = constrain(d,140.0,150.0);
  force.normalize();                                  //Normalizar Vector
  float strength = (G * mass * m.mass) / (d * d);      //Se calcula la fuerza de gravedad
  force.mult(strength);                               // Vector de fuerza--> magnitud * dirección
  return force;
 }

 // Método para desplegar atractores en  pantalla

 void display() {
  ellipseMode(CENTER);
  strokeWeight(1);
  stroke(0);
  fill(175,200);
  pushMatrix();
  lights();
```

```
    translate(location.x, location.y, location.z);
    //sphere(mass);
    //translate(0,0,location.z);
    ellipse(location.x,location.y,mass*2,mass*2);
    popMatrix();
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

/*Esta sección representa la clase y los métodos correspondientes a los nodos, así como el factor interactivo en la aplicación"*/

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//Clase Mover, que se ha definió como Node

class Mover {

  PVector location;
  PVector velocity;
  PVector acceleration;
  float mass;

  Mover(float m, float x, float y) {
    mass = m;

      //La posición ha sido definida por números aleatorios dentro de los rangos de valores establecidos.

          location = new PVector(random((random(3800)+width)/2,width/2),random((random(4000)+height)/2,height/2),random(-1660,1600));

    //Se establecen los valores de los vectores de aceleración y velocidad.

    velocity = new PVector(0.6, 0.6, 0.6);
    acceleration = new PVector(0, 0, 0);
  }

  void applyForce(PVector force) {

     //Cálculo de fuerzas

     PVector f = PVector.div(force, mass);
     acceleration.add(f);
   }
```

```
  void update() {

    //Aplicación de fuerzas

    velocity.add(acceleration);
    location.add(velocity);
    location.add(velocity);
    location.add(velocity);
    location.add(velocity);
    location.add(velocity);
    location.add(velocity);
    acceleration.mult(0);
  }

  // Método para desplegar nodos en la pantalla

  void display() {
    stroke(mouseY,mouseY,100,10);
    point(location.x, location.y,location.z);
  }
}
```

Las figuras 5.21, 5.22, 5.23 y 5.24 muestran algunos de los resultados obtenidos ejecutando la aplicación anterior.

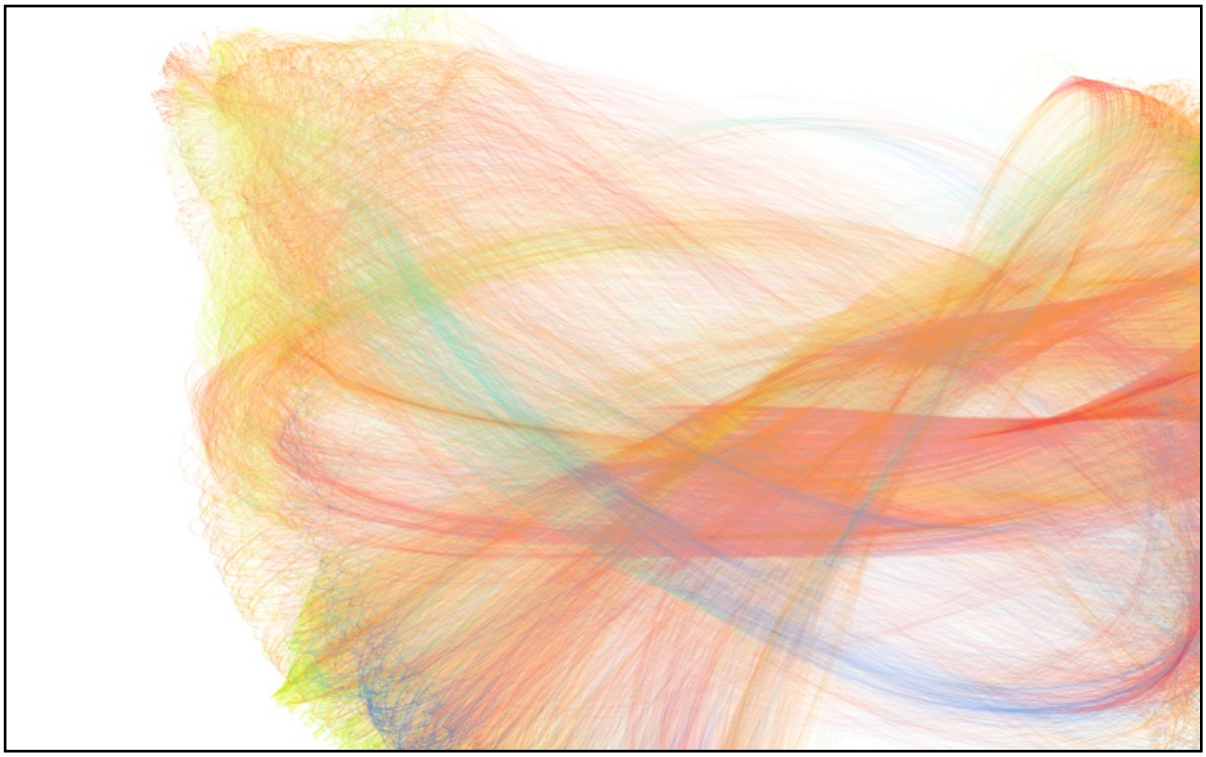


Figura 5.21
Sistema de dos dimensiones.

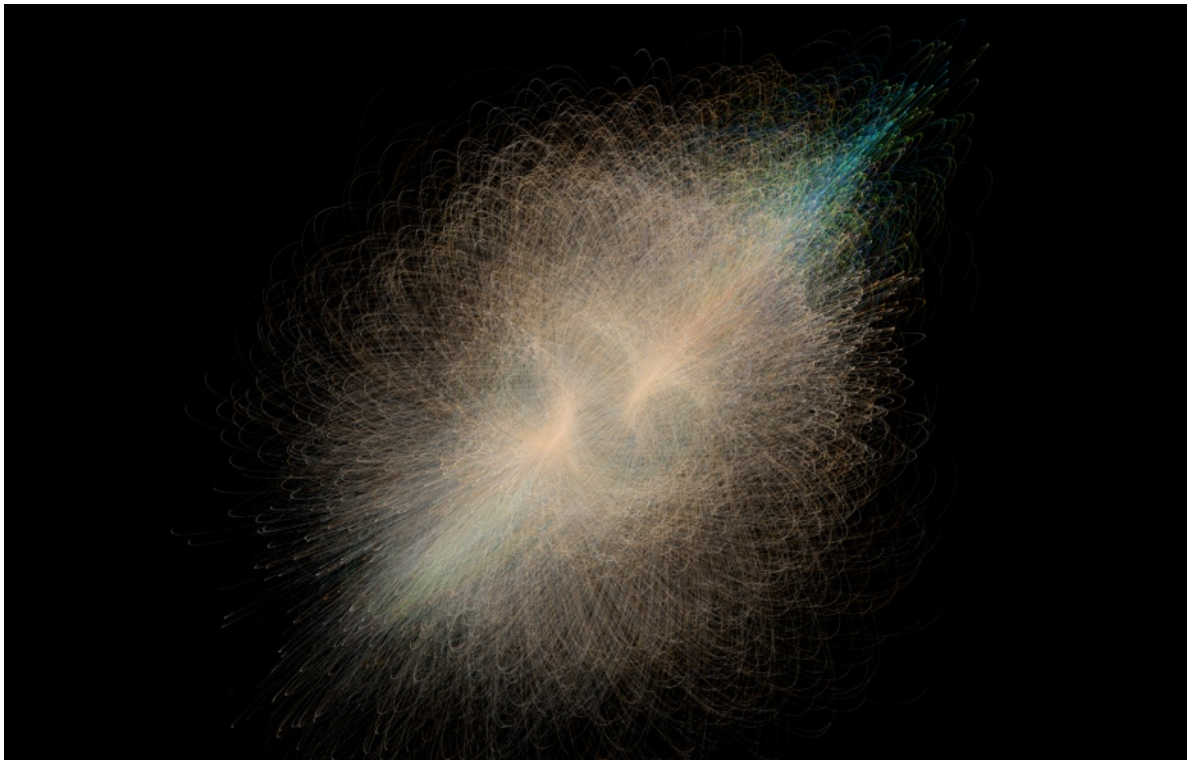


Figura 5.22
Sistema de tres dimensiones.

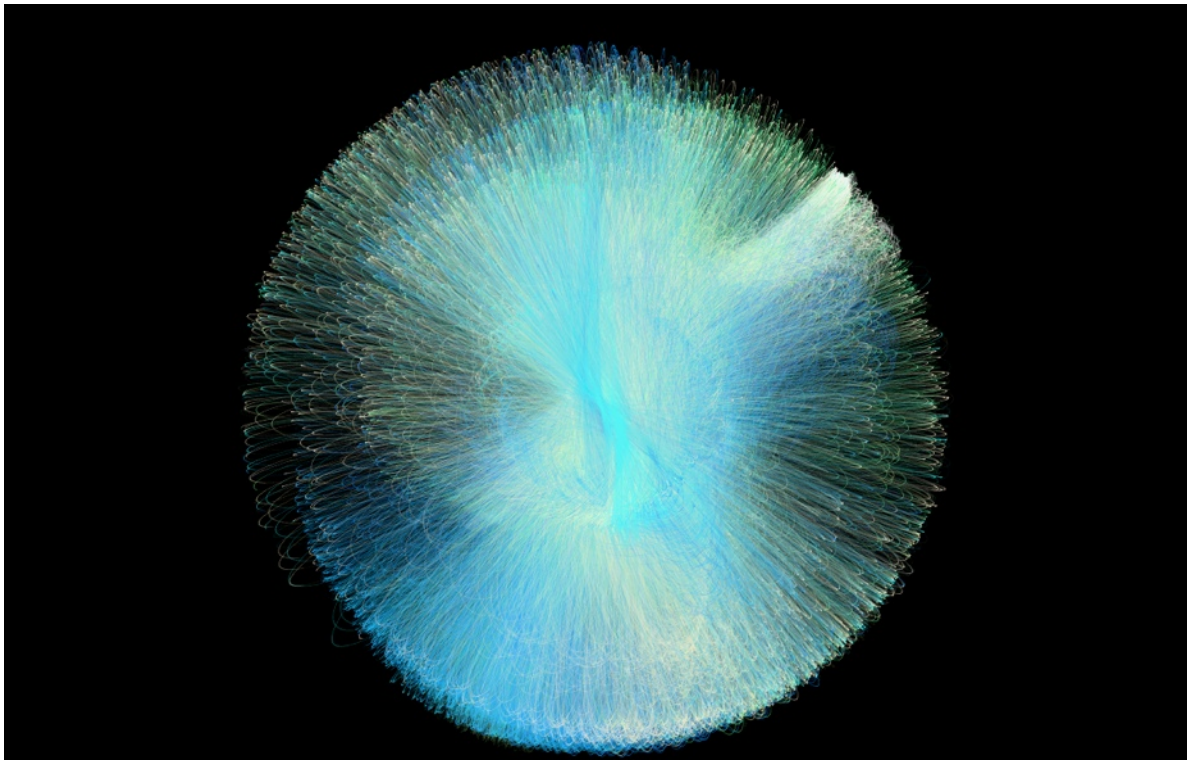


Figura 5.23
Sistema de tres dimensiones.

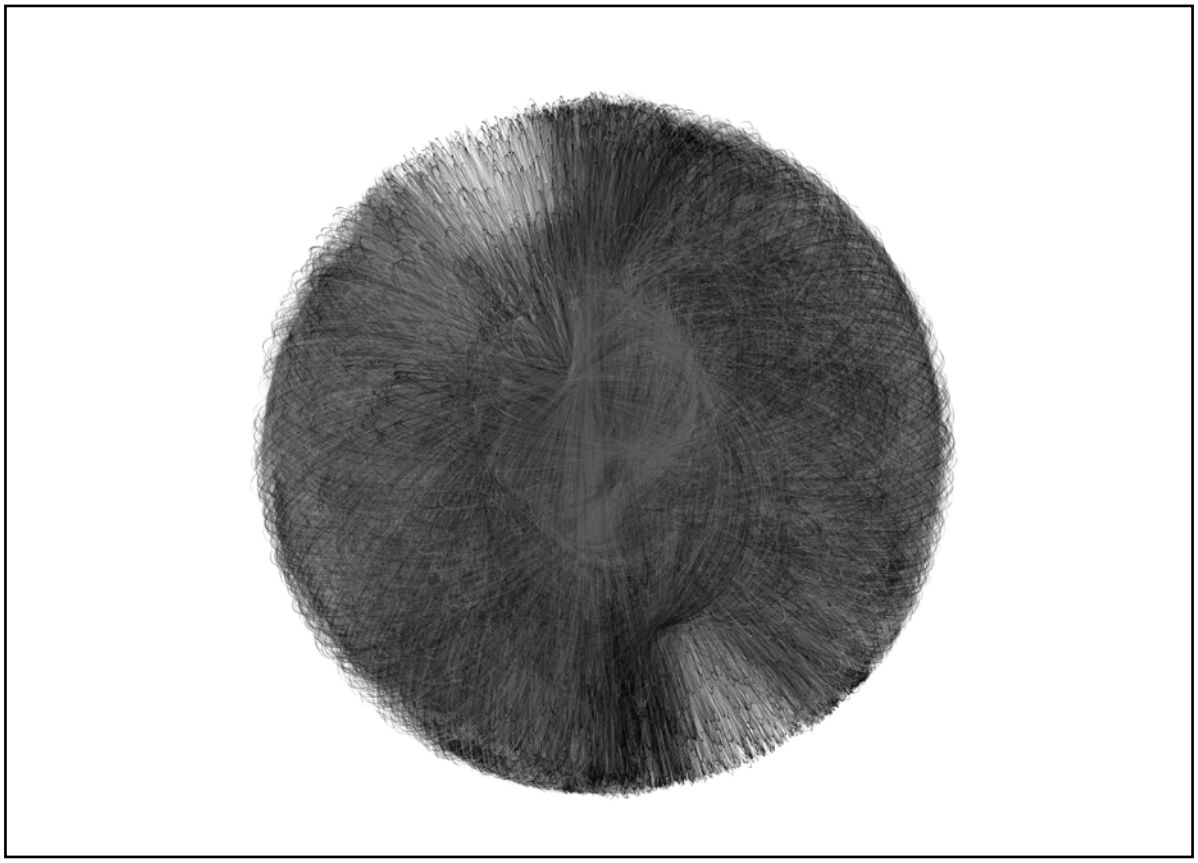


Figura 5.24
Sistema de tres dimensiones.

5.2 P5.js, una alternativa web

Se decidió considerar al desarrollo web como una alternativa dada la creciente demanda en el mercado de experiencias interactivas en este medio. El desarrollo en P5.js, permite una rápida transición de aplicaciones desarrolladas en Processing a la web, intentando aproximarse a la sintaxis y convenciones de Processing lo mas cercano posible. P5.js está dirigido para el desarrollo en JavaScript, mientras que Processing lo está para Java. Ambos lenguajes poseen amplias diferencias en cuanto a estructura y alcances pero es posible llegar a homologar funciones que permitan una rápida migración entre los lenguajes.

La primera versión beta de P5.js fue lanzada en agosto del 2014. A partir de ahí ha sido empleada para numerosos proyectos alrededor del mundo, además la contribución por parte de la comunidad de miles de usuarios para añadir funciones, solucionar errores, documentar, generar ideas y discusiones ha permitido que la herramienta evolucione y madure con rapidez [26].

En esta pequeña aplicación se pretende demostrar la facilidad de transición hacia sistemas web, así como una alternativa más para crear aplicaciones interactivas con un enfoque generativo de una forma ágil y sencilla.

Al ser una aplicación web, es necesario por naturaleza administrar un servidor que aloje dicha aplicación. Para fines demostrativos, se ha optado por utilizar un servicio gratuito ya que esta aplicación no compromete información del usuario ni es parte de un sistema crítico.
La interacción se centra cuando el mouse es presionado y arrastrado sobre la ventana del navegador de internet, creando así formas interesantes para el usuario.

```
//Se declara _num  el número de líneas totales, tamaño de cada línea y su respectivo largo

var _num = 196 ,i,value;
var pArr = [];
var tamano= 50, largo=1;


function setup() {

 //Se crea ahora una ventana de navegador

 createCanvas(windowWidth, windowHeight);
 background(255);

 x=tamano;
 y=tamano;
 sub=0;

 // Se realiza una operación para determinar el número de líneas, crear los objetos y
 //distanciar el punto de origen de cada línea de acuerdo a las proporciones de la ventana

      for(t=1;  t<=((round(windowWidth/tamano))*(round(windowHeight/tamano)));  t++)
pArr.push(t.toString());
   for (i=0;i<pArr.length;i++){
       y=(i*tamano)+tamano;
       pArr[i] = new linea(i);
       pArr[i].init(y-sub,x);
       if(((i+1)%(floor(windowWidth/tamano)))===0 && i!==0){
         x+=tamano;
         sub+=floor(windowWidth/tamano)*tamano;
       }
   }
}
```

```
function draw() {
  noStroke();

  //Se crea un rectángulo para añadir transparencia y efecto barrido

  fill(255,15);
  quad(0,0,width,0,width,height,0,height);

  for (var i=0;i<pArr.length;i++) {
   pArr[i].calculalinea();
  }
```

/*Si mientras el mouse se encuentra presionado el largo de la línea es menor a 10 veces el tamaño original, entonces continúa aumentando. Si no está presionado, entonces el largo disminuye hasta una unidad*/

```
  if (mouseIsPressed){
    value = 0.5;
    if(largo<=(tamano*10))
      largo+=1;
  }

  else{
    if(largo>1)
    largo-= 1;
    value=1;
   }
}

function linea(){
this.center=new createVector(this.x,this.y);
this.mouse=new createVector(mouseX,mouseY);
this.x;
this.y;
this.id;

 this.linea = function(numero){
    this.id = numero;
    // init();
  };

  this.init = function(x_,y_) {
        this.x = x_;
        this.y = y_;

        // Se crea un vector con las coordenadas indicadas

        this.center = new createVector(this.x,this.y);
```

```
  };

    this.calculalinea = function() {

      //Se calcula la dirección de cada vector (cada línea) con respecto a su origen y la posición del mouse

     this.mouse= new createVector(mouseX,mouseY);
     this.center = new createVector(this.x,this.y);
     this.mouse.sub(this.center);
     this.mouse.normalize();
     this.mouse.mult(largo);
     translate(this.x,this.y);
     stroke(0,50);
     strokeWeight(value);
     line(0,0,this.mouse.x,this.mouse.y);
     resetMatrix();
    }
}

//Si el usuario modifica las dimensiones de la ventana del navegador, entonces la aplicación se reinicia con nuevos cálculos.

function windowResized() {
 setTimeout(function () {
       location.reload()
    }, 100);
}

                                        /*FIN*/
```

Finalmente es necesario agregar unas cuantas líneas al archivo “index.html” las cuales adicionan de las bibliotecas necesarias así como la definición del archivo JavaScript que contiene la aplicación.

```
<script src="libraries/p5.js" type="text/javascript"></script>
<script src="libraries/p5.dom.js" type="text/javascript"></script>
<script src="libraries/p5.sound.js" type="text/javascript"></script>
<script src="sketch.js" type="text/javascript"></script>
```

Finalmente los resultados obtenidos pueden ser consultados en la siguientes direcciónes de internet: http://armyrdzz.net23.net/lineas/ y http://armyrdzz.net23.net/curvas/. La figura 5.25 representa una captura de la interacción del usuario con la aplicación.

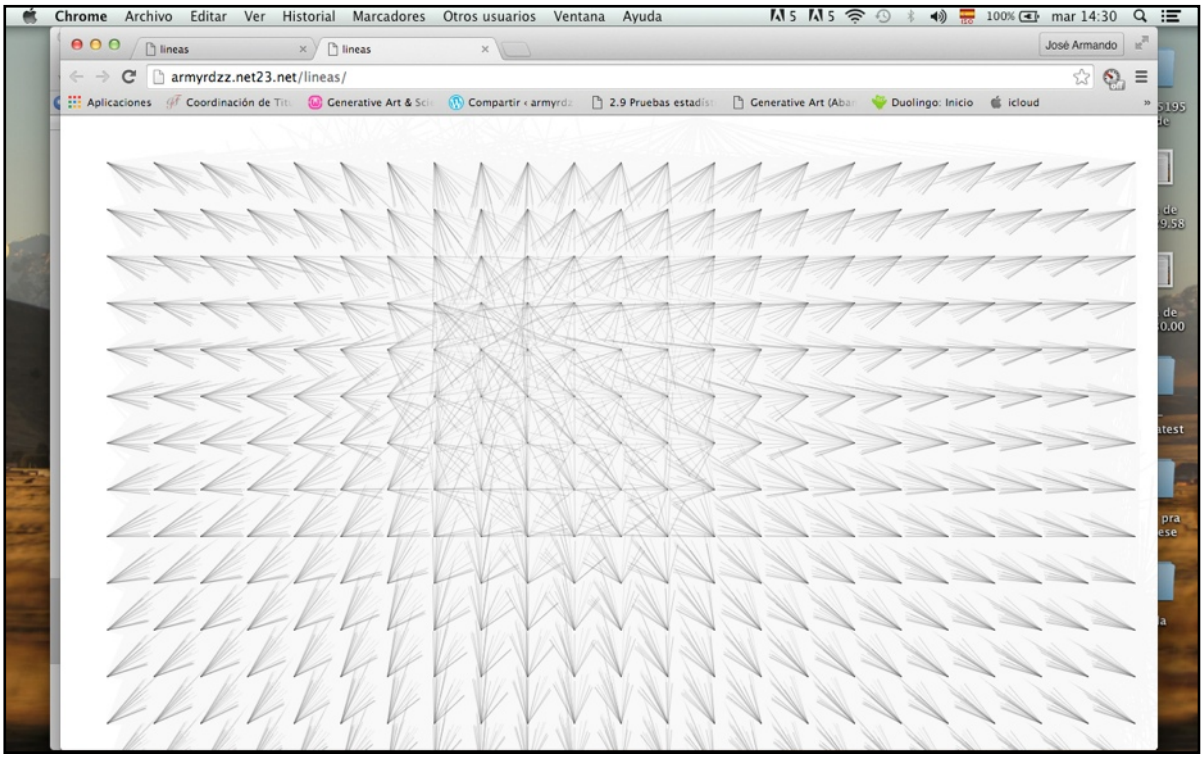


Figura 5.25

5.3 Aplicaciones Interactivas de los Algoritmos Generativos

Para finalizar con las aplicaciones, se expondrán un ejemplo desarrollado que involucra el factor interactivo con uno a mas usuarios. Además se considerarán mayores aspectos a la metodología que antes se ha mencionado. Cabe mencionar que factores como la instalación física del sistema así como de sus componentes, pruebas para el espacio definitivo, costos y funciones del software que requieren mayores capacidades de procesamiento (de los cual se detallará mas adelante) serán mencionadas e implementadas de manera teórica, mientras que las características del software omitidas serán implementadas y puestas a prueba aunque únicamente para fines demostrativos ya que afectan drásticamente el desempeño del funcionamiento de la aplicación final.

Estas aplicaciones involucran la implementación de dispositivos para interacción como el Kinect, implementación del protocolo de comunicación “Syphon” (de forma teórica, ya que en las pruebas realizadas esta función requiere amplia capacidad de procesamiento gráfico) y uso de proyectores para aplicaciones de video mapping (por medio de “Syphon”).

5.3.1 Kinect, una alternativa para desarrollo de software interactivo

En esta aplicación se ha integrado a la interacción como elemento fundamental. Se ha decidido utilizar el sensor Kinect como principal interfaz entre el usuario y el sistema, además se ha implementado el protocolo de comunicación gráfica “Syphon” para transmitir la información a un programa de video mapping, el cual permite adaptar la imagen desplegada por un proyector a casi cualquier superficie que sea necesaria.

Por el lado del software, se ha decidido implementar un algoritmo de tipo “blob detection” el cual a diferencia de los algoritmos convencionales basados en la detección de manchas por medio de los colores. En otras palabras este tipo de algoritmos en su forma más básica se basan en detectar por medio del análisis de la imagen, patrones de colores específicos. A partir de reglas se determina si existe un “blob” o no, estas reglas están basadas en una distancia máxima predeterminada entre un pixel de un determinado color con otro del mismo, esta distancia está establecida mediante la cantidad de pixeles de separación entre uno y otro. Cuando existe un conjunto de puntos conglomerados en una región de la imagen se genera un “blob” nuevo, cuando un punto queda fuera de la distancia máxima establecida, entonces se determina que no pertenece al “blob”. Otro factor para determinar la existencia o no de un “blob” es mediante un proceso posterior, el cual mediante el cálculo de los puntos frontera del “blobs” (x máxima, y máxima, x mínima, y mínima) se determina el tamaño del “blob” y su centro, gracias a la obtención de estos datos se pueden descartar aquellos que no cumplen con un tamaño mínimo previamente establecido, es decir, se evita la creación de innecesarios que por su tamaño puede tratarse de ruido. Este algoritmo se repite por cada cuadro de video recibido.

Por otro lado el algoritmo implementado no está basado en los colores de la imagen, si no en objetos que se encuentran en un rango de profundidad previamente delimitado. Como se explicó anteriormente, el kinect es capaz de analizar la información de profundidad del entorno y por medio de las bibliotecas mencionadas es posible determinar el rango de profundidad que se desea analizar y por consiguiente descartar el resto de objetos en el entorno a esta variable se le ha denominado threshold. El algoritmo analiza la imagen obtenida por el Kinect y coloca los puntos en un espacio tridimensional, aquellos puntos que si se encuentran en dicho rango son considerados y son representados por un pixel de un color específico, posteriormente son almacenados en una pre-imagen la cual nunca es desplegada. El resto de pixeles descartados son coloreados de gris y son colocados en la misma pre-imagen. En este punto se crean imágenes que ahora son enviadas al

algoritmo de “blob detection” por lo cual se podrá notar que la interacción entre el usuario y el sistema está basada en “que tan lejos o cerca” se encuentre el usuario del sensor y sus movimientos.

El algoritmo “blob detection” está basado en los trabajos de Computer Vision de Daniel Shiffman[27] así como algunas mejoras implementadas. Dichas mejoras están basadas en un mejor desempeño para la creación y eliminación de “blobs”; su persistencia en el tiempo en caso de no ser detectados por breves instantes de tiempo y no ser tratados simplemente como nuevos “blobs” y asignar identificadores a cada “blob” para evitar que se intercambien las propiedades de cada uno cuando uno o varios “blobs” se cruzan.

Para el primer caso se ha optado por almacenar los “blobs” en un arreglo y mediante las reglas antes mencionadas eliminarlos. De igual forma cuando alguno de estos desaparece definitivamente del rango de profundidad establecido. Por otra parte se ha establecido por medio de una variable contadora el tiempo máximo que un “blob” puede desaparecer de la visión del sensor, esto para evitar la pérdida de la información que ha generado cada uno de estos. Es muy probable que el usuario de forma involuntaria se aleje del rango establecido o que por motivos de cómputo la mancha deje de ser detectada, es por eso que se ha decidido implementar esta función al algoritmo.

Finalmente siempre se almacena la posición actual de cada “blob” a fin de calcular para el siguiente cuadro la distancia de la posición actual con la anterior y así determinar si se trata del mismo a pesar de que se llegasen a cruzar en la imagen. A pesar de resultar una mayor carga de procesamiento, se obtiene una mejor experiencia de usuario como se constató en las pruebas realizadas de las cuales se hará una mención detallada mas adelante. A continuación se presenta el código correspondiente a la clase “Blob” la cual se encarga de realizar el primer procedimiento mencionado, posteriormente se presentará la clase “Kinect Tracker” la cual realiza las operaciones de control de “Blobs” (añadir, eliminar y clasificar) y análisis de las imágenes entregadas por el Kinect.

5.3.1.1 Planeación de actividades

Nombre de la tarea	Fecha de Inicio	Fecha final	Duración
Planeación	**19/09/16**	**23/09/16**	**5d**
Diseño	19/09/16	21/09/16	3d
Análisis de Costos	22/09/16	23/09/16	2d
Desarrollo de prototipos	**26/09/16**	**07/10/16**	**10d**
Desarrollo de algoritmo generativo a implementar	26/09/16	03/10/16	6d
Pruebas de rendimiento	04/10/16	05/10/16	2d
Desarrollo de software de rastreo	29/09/16	05/10/16	5d
Pruebas de sofware prototipo	06/10/16	07/10/16	2d
Desarrollo segunda fase	**10/10/16**	**21/10/16**	**10d**
Implementación de sensor Kinect al software de rastreo	10/10/16	12/10/16	3d
Implementación de algoritmo generativo al software	13/10/16	14/10/16	2d
Pruebas de funcionales y de rendimiento	17/10/16	21/10/16	5d
Desarrollo tercera fase	**24/10/16**	**04/11/16**	**10d**
Implementación de canal de comunicación Syphon con software de video mapping	24/10/16	26/10/16	3d
Pruebas funcionales y de rendimiento	27/10/16	28/10/16	2d
Construcción de la instalación	31/10/16	04/11/16	5d
Calibración del sistema en el entorno de funcionamiento	04/11/16	04/11/16	1d
Pruebas finales	04/11/16	04/11/16	1d
Entrega	07/11/16	07/11/16	1d

Figura 5.26
Listado de Actividades

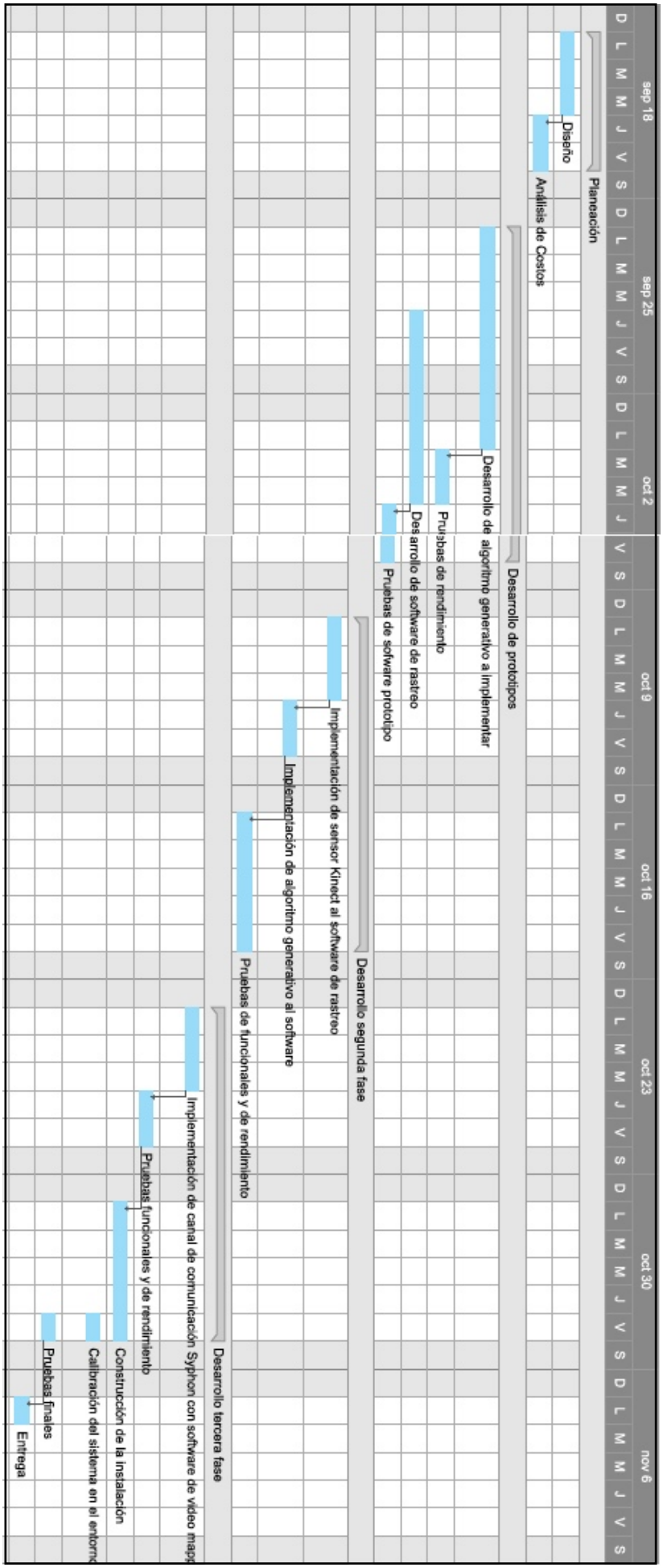


Figura 5.27
Diagrama Gantt

5.3.1.2 Código fuente

Clase Blob

```
class Blob {
   float minx,miny,maxx,maxy;
   int id;
   boolean taken = false;
   int lifespan = 1000;


   Blob(float x, float y) {

    //Posición X, Y, ancho y alto del blob

    minx = x;
    miny = y;
    maxx = x;
    maxy = y;
    //ID blob
    id = 0;

   }
   void add(float x, float y){
    minx = min(minx,x);
    miny = min(miny,y);
    maxx = max(maxx,x);
    maxy = max(maxy,y);
   }

   void become(Blob other){
    minx = other.minx;
    miny = other.miny;
    maxx = other.maxx;
    maxy = other.maxy;
   }

   float size(){
    return (maxx - minx) * (maxy - miny);
   }

   PVector getCenter() {
    float x = (maxx-minx) * 0.5 + minx;
    float y = (maxy-miny) * 0.5 + miny;
    return new PVector(x,y);
   }
```

```
  PVector getPos() {
    float x = (minx * 1.6875 +maxx * 1.6875)/2;
    float y = (miny * 1.5 +maxy * 1.5)/2;
    return new PVector(x,y);
  }

  boolean checkLife(){
     lifespan--;
     if(lifespan < 0){
       return true;
     } else{
       return false;
     }
  }

  void show() {
    stroke(0);
    fill(255);
    strokeWeight(2);
    rectMode(CENTER);
   // println(id);
   // ellipse((minx * 1.6875 +maxx * 1.6875)/2 ,(miny * 1.5 + maxy * 1.5)/2, 50,50);
  }

  boolean isNear(float x, float y){

    //Centro del Blob

    float cx = (minx + maxx) / 2;
    float cy = (miny + maxy) / 2;

    float d = tracker.distSq(cx,cy,x,y);
    if(d < threshold * threshold) {
      return true;
    } else {
      return false;
    }
  }
}
```

Clase Kinect Tracker

```
class KinectTracker {

  //Profundidad inicial
  int threshold = 745;

  //Posición RAW
  PVector loc;

  //Posición interpolada
  PVector lerpedLoc;

  //Información de profundidad
  int[] depth;


  //Imagen que será desplegada al usuario
  PImage display;


  KinectTracker() {

    //inicia componentes

    kinect.initDepth();
    kinect.enableMirror(true);

    //crea un a imagen en blanco

    display = createImage(kinect.width, kinect.height, RGB);

    //Configura ambos vectores
    loc = new PVector(0, 0);
    lerpedLoc = new PVector(0, 0);
  }

  void track() {

    //Obtiene la profundidad como un arreglo de enteros

    depth = kinect.getRawDepth();
    if (depth == null) return;


    for (int x = 0; x < kinect.width; x++) {
        for (int y = 0; y < kinect.height; y++) {
```

```
          int offset =  x + y*kinect.width;

          //Toma la profundidad raw

          int rawDepth = depth[offset];

          //Evita el límite

          if (rawDepth < threshold) {

              boolean found = false;

                for(Blob b : currentBlobs){
                   if(b.isNear(x,y)){
                      b.add(x,y);
                      found = true;
                      break;
                     }
                  }
                if(!found){
                   Blob b = new Blob(x,y);
                   currentBlobs.add(b);
                  }
            }
          }
      }

for(int i = currentBlobs.size()-1; i>=0; i--) {
  if (currentBlobs.get(i).size() < profundidadBlob){
  currentBlobs.remove(i);
  }
}


//Compara currentBlobs con los blobs de cada cuadro

// No hay Blobs

if(blobs.isEmpty() && currentBlobs.size() > 0 ){
   for(Blob b: currentBlobs){
     b.id = blobCunter;
     blobs.add(b);
     systems.add(new ParticleSystem (b.getPos()));
     blobCunter++;
   }
} else if (blobs.size() <= currentBlobs.size()){
    for(Blob b : blobs){
```

```
    float recordD = 1000;
    Blob matched = null;
    for(Blob cb : currentBlobs){
       PVector centerB = b.getCenter();
       PVector centerCB = cb.getCenter();
       float d = PVector.dist(centerB, centerCB);
       if(d < recordD && !cb.taken) {
         recordD = d;
         matched = cb;
         }
       }
      matched.taken = true;
      b.become(matched);
    }

    for(Blob b : currentBlobs){
      if(!b.taken){
        b.id = blobCunter;
        blobs.add(b);
        systems.add(new ParticleSystem (b.getPos()));
        blobCunter++;
      }
    }
  } else if(blobs.size() > currentBlobs.size()){

   for(Blob b : blobs){
    b.taken = false;
   }

   for(Blob cb : currentBlobs){
    float recordD = 1000;
    Blob matched = null;
    for(Blob b : blobs){
       PVector centerB = b.getCenter();
       PVector centerCB = cb.getCenter();
       float d = PVector.dist(centerB, centerCB);
       if(d < recordD && !b.taken) {
         recordD = d;
         matched = cb;
         }
       }
      if(matched != null){
      matched.taken = true;
      matched.lifespan = 1000;
      matched.become(cb);
      }
   }
```

```
        for(int i = blobs.size() - 1; i>=0 ; i--){
          Blob b = blobs.get(i);
          if(!b.taken){
             if(b.checkLife()){
             blobs.remove(i);
             systems.remove(i);}
          }
        }

    }

   for (int i = blobs.size() - 1; i>=0 ; i--){

     Blob b = blobs.get(i);
     b.show();
  //Cálculos y visualización de partículas
     ParticleSystem ps = systems.get(i);
     ps.addParticle((b.getPos()));
     ps.run();

   }

    //Interpola la posición
    lerpedLoc.x = PApplet.lerp(lerpedLoc.x, loc.x, 0.3f);
    lerpedLoc.y = PApplet.lerp(lerpedLoc.y, loc.y, 0.3f);
}

  float distSq(float x1, float y1, float x2, float y2) {
    float d = (x2 - x1) * (x2 - x1) + (y2- y1) * (y2 - y1);
    return d;
  }

  PVector getLerpedPos() {
    //Calculado a partir de width/kinect.width y height/kinect.height

    lerpedLoc.x = lerpedLoc.x * 1.6875;
    lerpedLoc.y = lerpedLoc.y * 1.5;
    return lerpedLoc;
  }

  PVector getPos() {

    //Calculado a partir de width/kinect.width y height/kinect.height

    loc.x = loc.x * 1.6875;
    loc.y = loc.y * 1.5;
    return loc;
  }
```

```
  void display() {

    PImage img = kinect.getDepthImage();

    //Cuando ninguna imagen es encontrada
    if (depth == null || img == null) return;

    //Los puntos de profundidad son abtenidos de nuevo y  desplegados ahora en pantalla

    display.loadPixels();
    for (int x = 0; x < kinect.width; x++) {
      for (int y = 0; y < kinect.height; y++) {

        int offset = x + y * kinect.width;
        int rawDepth = depth[offset];
        int pix = x + y * display.width;
        if (rawDepth < threshold) {
          //si son encontrados los puntos, se pintan de rojo
          display.pixels[pix] = color(150, 50, 50);
        } else {
          display.pixels[pix] = img.pixels[offset];
        }
      }
    }
    display.updatePixels();

    //Dibuja imagen
    //image(display, 0, 0);
     image(display, 0, 0, width, height);
  }

  int getThreshold() {
    return threshold;
  }

  void setThreshold(int t) {
    threshold =  t;
  }
}
```

Una vez explicados los algoritmos de análisis de la imagen y se han obtenido las posiciones de los objetos o usuarios que se encuentran en el rango de profundidad establecido, se procederá a crear un sistema de partículas para cada “blob” existente. He aquí la importancia de refinar el algoritmo para administrar correctamente la creación y eliminación de los “blobs” bajo ciertas condiciones ya que de lo contrario, rápidamente el sistema se saturaría de elementos que afectarían de manera inmediatamente el desempeño y rendimiento de la interacción.

Dichos sistemas de partículas está basado en una aplicación expuesta anteriormente (sistemas de atractores múltiples) la cual implementa principios básicos de la física, tales como gravedad, fuerzas y aceleración. Para cada “blob” le es asignado un sistema de partículas que con cada refresco de pantalla se añade una nueva partícula a cada uno, cada partícula posee características propias, por ejemplo: posición velocidad inicial, aceleración, tamaño y color. Para evitar la saturación de partículas y por tanto el rendimiento del programa, se tiene como parámetro la transparencia del color de la partícula, cada ciclo la transparencia es disminuida (inicialmente con el valor de 255) hasta llegar a cero. Si la dicho valor alcanza el valor de cero o negativo, entonces la partícula es eliminada del sistema. Cada partícula tiene una velocidad y tamaño diferentes, el tamaño disminuye con cada ciclo y la velocidad aumenta debido al factor de aceleración.

Finalmente en esta sección se implementa el protocolo de comunicación “Syphon”, para el cual es necesario de manera similar enviar los elementos gráficos a una pre-imagen la cual será enviada a través del protocolo, recibida por el software de video mapping y finalmente proyectada. Es importante destacar que esta última función se puso a prueba y tuvo un resultado positivo desde el punto de vista funcional, pero no por el lado del rendimiento, ya que en esta última fase requiere que en la misma computadora esté corriendo el programa de mapeo de imágenes. Finalmente se obtuvieron animaciones lentas y carentes de cualquier sentido de interacción. A continuación se muestra el código correspondiente a la creación de los sistemas de partículas con la clase ParticleSystem, así como la creación de cada una de las partículas del sistema con sus respectivas características con la clase Particle.

Clase ParticleSystem

```
//Creador de Partículas

class ParticleSystem {
  ArrayList<Particle> particles;
  PVector origin;

  ParticleSystem(PVector location) {
    origin = location.get();
    particles = new ArrayList<Particle>();
  }

  void addParticle(PVector locationP) {
    particles.add(new Particle(locationP));
  }

  void run() {

    //Las particulas invisibles son removidas para evitar la sobre carga de procesamiento

    Iterator<Particle> it = particles.iterator();
    while (it.hasNext()) {
      Particle p = it.next();
      p.run();
      if (p.isDead()) {
        it.remove();
      }
    }
  }
}
```

Clase Particle

```
class Particle {
  PVector location;
  PVector velocity;
  PVector acceleration;
  float lifespan, size, velocityRad, sizeRad;
  int r,g,b;

  Particle(PVector l) {
    acceleration = new PVector(0, 0.05);
    velocity = new PVector(random(-2, 2), random(-2, 2));
    velocityRad = random(0.01);
    location = l.get();
```

```
  lifespan = 105.0;
  size=60;
  sizeRad=20;
  r=int(random(100,255));
  g=int(random(100,255));
  b=int(random(100,255));
}

void run() {
  update();
  display();
}

void update() {
  velocity.add(acceleration);
  location.add(velocity);
  lifespan -= 0.5;
  size-=0.5;
  sizeRad-=0.5;
}


void display() {
  noStroke();
  fill(r,0,0,lifespan);
  ellipse(location.x, location.y, size, size);
  stroke(r,0,0,lifespan+90);
  pushMatrix();
     translate(location.x, location.y);
     rotate(millis()*velocityRad);
     line(0,0, size/3,size/3);
       pushMatrix();
         rotate(millis()*velocityRad*2);
         line(0,0, size/3,size/3);
       popMatrix();
  popMatrix();
}

// */



/*
  void display() {
  canvas.beginDraw();
 noStroke();
 fill(r,0,0,lifespan);
 canvas.ellipse(location.x, location.y, size, size);
```

```
  stroke(r,0,0,lifespan+90);
  pushMatrix();
      translate(location.x, location.y);
      rotate(millis()*velocityRad);
      canvas.line(0,0, size/3,size/3);
        pushMatrix();
          rotate(millis()*velocityRad*2);
          canvas.line(0,0, size/3,size/3);
        popMatrix();
   popMatrix();
 canvas.endDraw();
 // image(canvas, 0, 0);
  server.sendImage(canvas);
  }
  */

  // Is the particle still useful?
  boolean isDead() {
    if (size < 0.0) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Finalmente se muestra la última sección de la aplicación la cual declara una interfaz para el operador, la cual sirve para ejecutar las siguientes acciones de ajuste: Establecer el intervalo de profundidad que se desea analizar del entorno (teclas arriba y abajo, arriba para aumentar la profundidad y abajo para disminuir) y manipular el motor de giro del Kinect (teclas “u” y “d”, “u” para aumentar en un grado hacia arriba el ángulo del sensor y “d” para disminuir en un grado hacia abajo el ángulo de dicho sensor). De igual manera en esta sección se declaran las bibliotecas a utilizar ; los arreglos de elementos correspondientes a los sistemas de partículas y “blobs” y finalmente se declara el canal de comunicación del protocolo “Syphon”. Esta sección en la aplicación está definida como AveragePointTracking.

AveragePointTracking

```
//Las bibliotecas son importadas
import codeanticode.syphon.*;
import java.util.Iterator;
import org.openkinect.freenect.*;
import org.openkinect.processing.*;

//Ángulo físico del kinect

float angle;

//Tamaño mínimo para que se construya

float threshold = 100;

//Tamaño mínimo para que se muestre en pantalla un blob

float profundidadBlob = 500;

ArrayList<Blob> blobs;

//Este arreglo es para conservar la persistencia del orden creado de cada blob

ArrayList<Blob> currentBlobs = new ArrayList<Blob>();
int blobCunter = 0;

//Arreglo de sistemas de partículas

ArrayList<ParticleSystem> systems;

//Declaración de objetos

KinectTracker tracker;
Kinect kinect;
ParticleSystem ps;

//Creación del server Sypon para enviar frames a un programa para mapeo de video

PGraphics canvas;
SyphonServer server;

void settings() {
  size(1080,720, P3D);
  PJOGL.profile=1;
}
```

```
void setup() {
  //size(1080, 720);

  canvas = createGraphics(400, 400, P3D);

  //Se inicia el objeto kinect y objeto tracker

  kinect = new Kinect(this);
  tracker = new KinectTracker();
  angle = kinect.getTilt();

  //Se inicia el sistema de partículas

  systems = new ArrayList<ParticleSystem>();
  server = new SyphonServer(this, "Processing Syphon");
}

void draw() {
  background(255);

  //Arreglo de Blobs para asignar un sistema a cada objeto

   blobs = new ArrayList<Blob>();

    //Despliega imágenes
       ////////////////////////////////////////IMPORTANTE, ESTA LINEA ES PARA REALIZAR
CALIBRACIÓN

  //tracker.display();

    //Ejecuta análisis
  tracker.track();
  currentBlobs.clear();
  //systems.clear();
}

//Se ajusta la profundidad de visión por medio de la teclas ariba y abajo
//La posición fisica del sensor se manipular por medio de las teclas "U" = ARRIBA y "D" =
ABAJO

void keyPressed() {
  int t = tracker.getThreshold();
  if (key == CODED) {
    if (keyCode == UP) {
      t+=5;
      tracker.setThreshold(t);
    }
```

```
  else if (keyCode == DOWN) {
     t-=5;
     tracker.setThreshold(t);
   }
  }
  else if (key == 'U' || key == 'u' ) {
     angle++;
   }
  else if (key == 'D' || key == 'd' ) {
     angle--;
   }
  angle = constrain(angle, 0, 30);
  kinect.setTilt(angle);
}
```

5.3.1.3 Pruebas

Las pruebas dinámicas realizadas fueron dirigidas hacia la funcionalidad y rendimiento de la aplicación. Las pruebas de funcionalidad se enfocaron en obtener los resultados esperados mediante pruebas dinámicas, es decir, ejecutando cada módulo del software de manera independiente y así verificar la exactitud del comportamiento obtenido. Cabe resaltar que se verificaron de manera individual los siguientes módulos:

-Correcta obtención e interpretación de la información entregada por el sensor (figura 5.28 y figura 5.29). Para este módulo es necesario escalar la información espacial entregada por el sensor, para que sea posteriormente adaptada al formato mínimo de resolución 720p. De igual forma la implementación de un algoritmo capaz de discernir los objetos localizados en el rango de profundidad establecido.

Figura 5.28

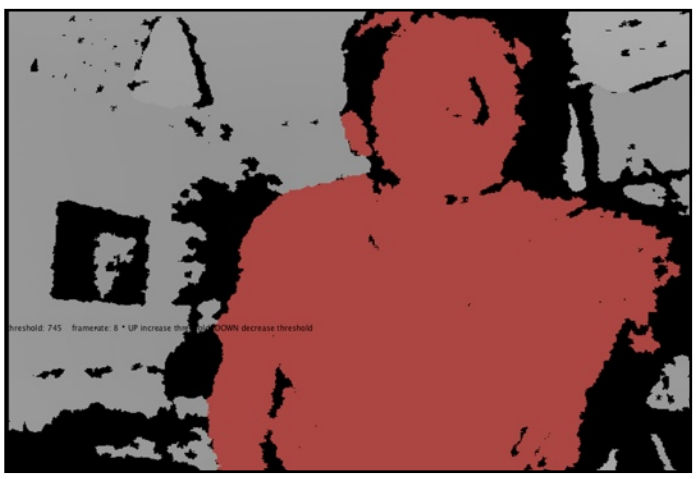


Figura 5.29

-Funcionamiento del algoritmo “blob detection” (figura 5.30). Para esta fase del desarrollo de la aplicación fue necesario implementar el algoritmo anterior con la detección de “blobs” así como su gestión, de la cual ya se ha descrito anteriormente.

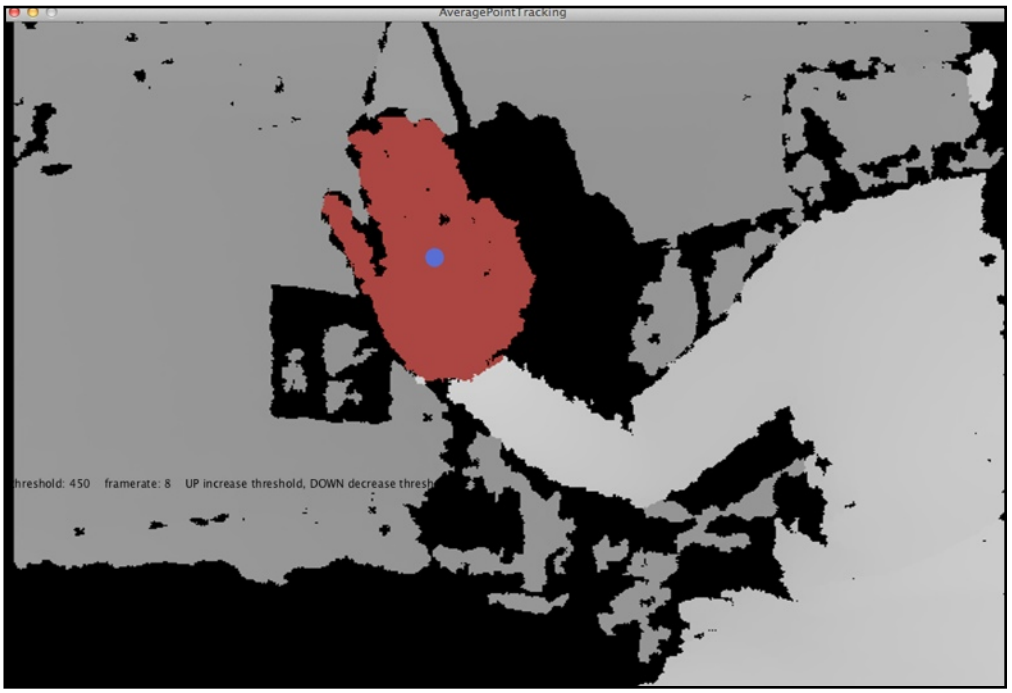


Figura 5.30

-Sistemas de partículas (figura 5.31 y figura 5.32). Una vez realizadas las implementaciones anteriores se procedió a añadir los sistemas de partículas asociados a cada “blob”, por tanto se tuvieron que realizar pruebas de estrés que permitieron visualizar los límites de interacción posibles sin afectar la experiencia de usuario. Para esto fue necesario suprimir la vista del operador la cual permite analizar la profundidad del sensor y ajustar los valores según el tipo de espacio en el que sea instalado el sistema. La visualización final únicamente se limita al resultado de la interacción con el usuario.

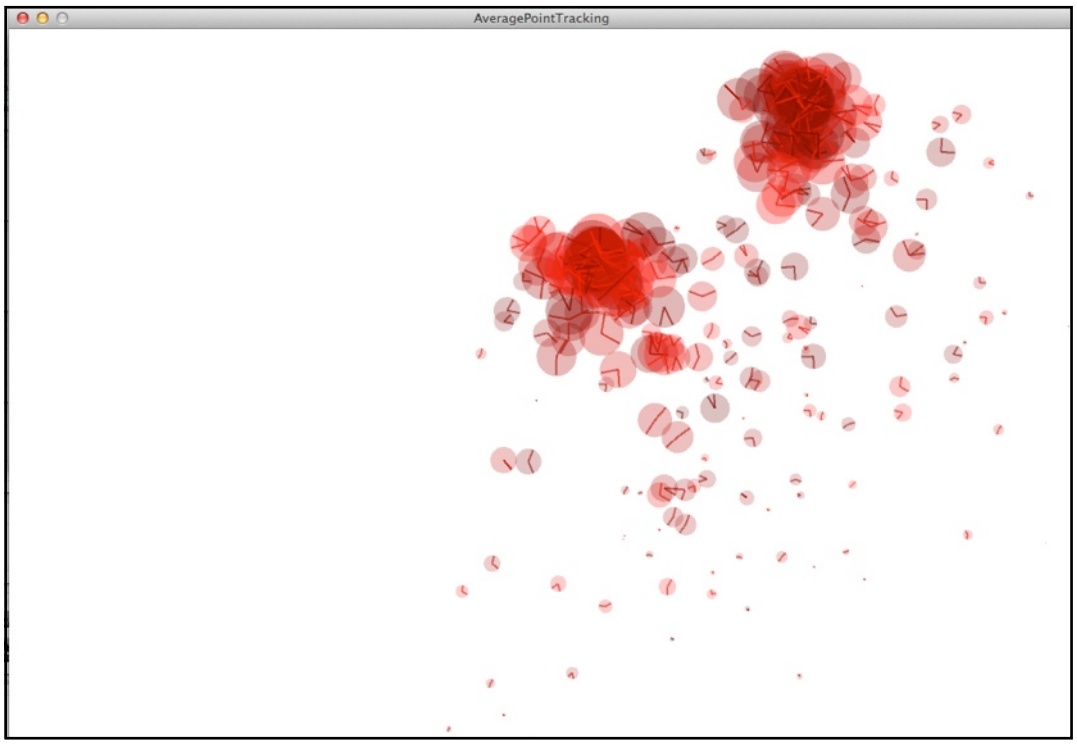


Figura 5.31

Figura 5.32
Pruebas con usuario.

-Comunicación del protocolo “Syphon” y uso de software para video mapping (figura 5.33). Por último cabe mencionar que la última prueba de rendimiento no fue superada dadas las características de la computadora en en que fue ejecutada la aplicación: MacBook, Mod. 2009, con procesador Intel Core 2 Duo a 2.26 GHz, 6 Gb de memoria RAM y tarjeta gráfica NVIDIA GeForce 9400M 256 MB. Es necesario contar con un equipo con características de procesamiento superiores para hacer uso de esta opción, pero cabe resaltar que este último módulo de la aplicación fue implementado y se comprobó su funcionalidad, aunque no con los resultados de desempeño esperados. El software de video mapping elegido para complementar la aplicación fue MadMapper, del cual existen versiones de prueba que permitieron realizar las pruebas pertinentes, además cuenta con la función de recepción de “Syphon” de manera integrada.

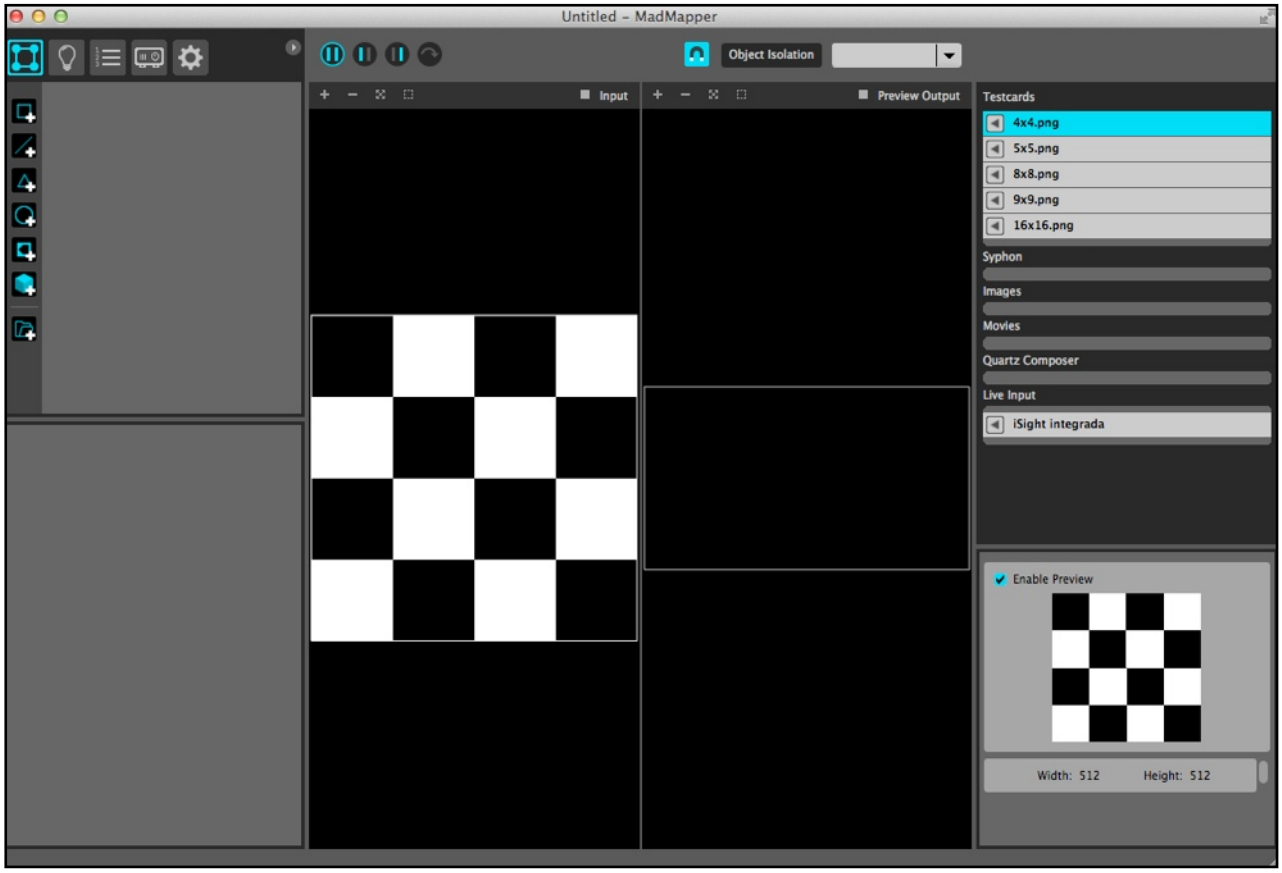


Figura 5.33
Interfaz gráfica del software de video mapping, MadMapper. A la derecha de la imagen es posible apreciar la función de recepción del protocolo “Syphon”.

5.3.1.4 Instalación

Desde el punto de vista de la interacción, la instalación es el espacio físico en donde la aplicación es ejecutada y en donde se da lugar a una interacción coherente con las características con que fue diseñado el sistema. En algunos casos si se requiere, es necesario adecuar el espacio a las características mínimas para que el sistema funcione correctamente. La intención de esta aplicación es que sea instalada en un espacio cuya distancia entre el techo y el suelo sea de entre 3.5 y 4 metros. El sensor y el proyector son colocados en el techo apuntando en dirección hacia el suelo, la computadora deberá estar situada a una distancia máxima de 10 metros del sensor y el proyector, el objetivo es que los usuarios caminen por debajo del sensor y justo sobre ellos sea proyectado un sistema de partículas que siguen los movimientos del usuario. El área de interacción depende de la distancia a la que son colocados el proyector y el sensor del suelo, finalmente se tiene previsto que el sistema funcione óptimamente con usuarios con estatura de entre 50 centímetros y 1.8 metros. A continuación se muestra un diagrama de la instalación física de la aplicación.

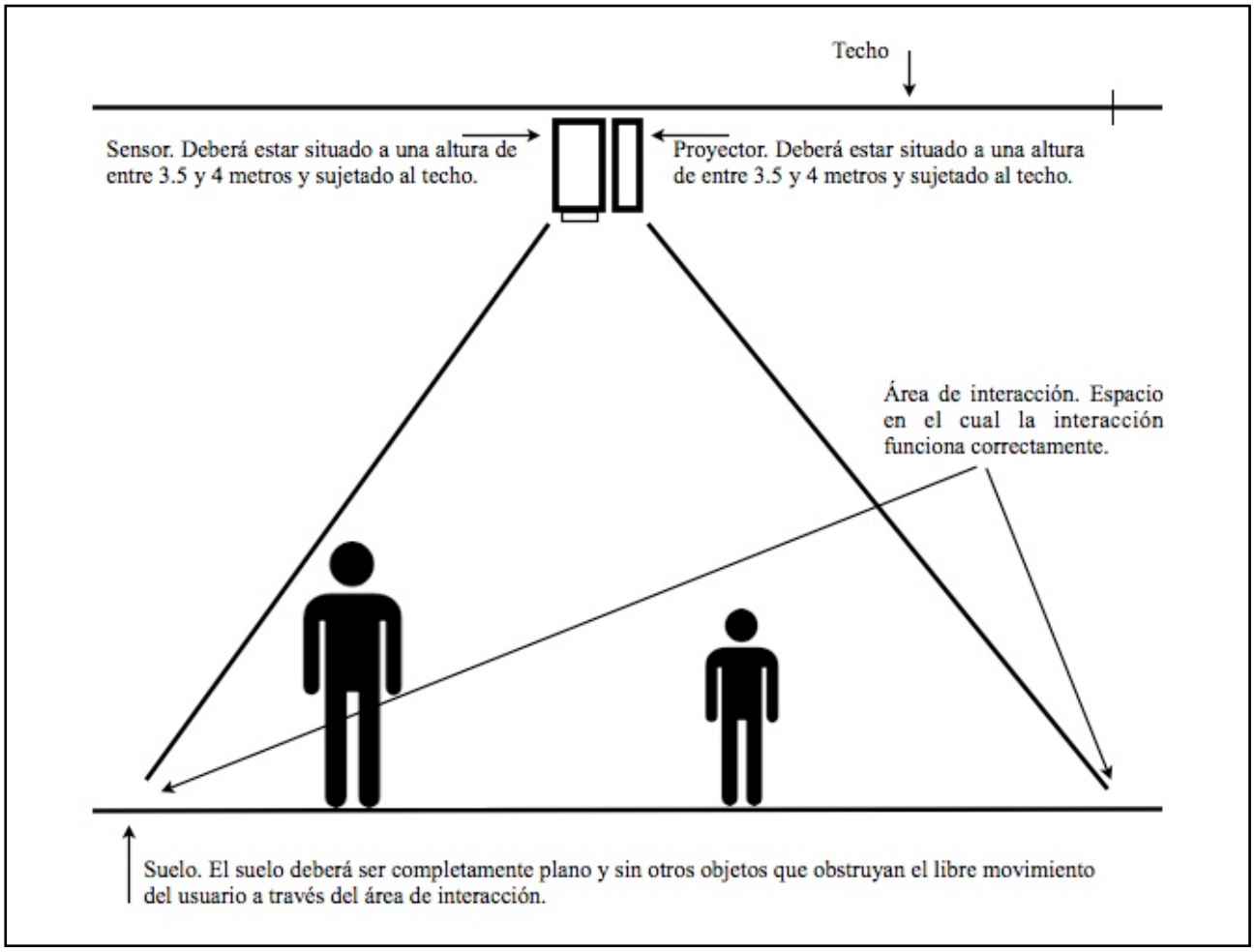


Figura 5.34
Vista frontal del sistema instalado en un espacio físico.

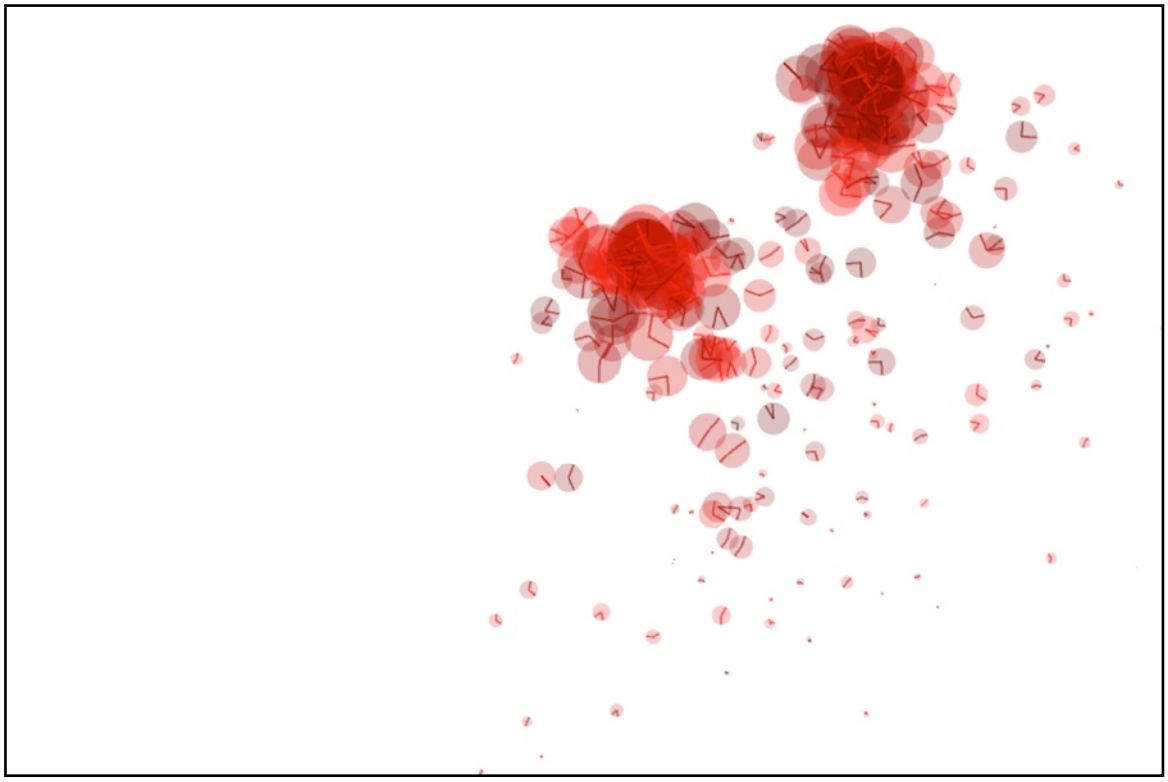


Figura 5.35
Vista superior del sistema instalado en un espacio físico. Las flechas en la imagen denotan la trayectoria del usuario en el área de interacción.

5.3.1.5 Análisis de resultados de rendimiento

Las condiciones mínimas necesarias para ejecutar la aplicación son: sistema operativo OSX versión 10.8.5 en adelante, procesador Intel Core 2 Duo a 2.26 GHz, 6 Gb de memoria RAM y tarjeta gráfica NVIDIA GeForce 9400M 256 MB, tener instalado el software Processing versión 3.0 o superior (opcional).

En las aplicaciones gráficas expuestas, la complejidad de los algoritmos no compromete el desempeño de la aplicación ni mucho menos rompe con la experiencia interactiva del usuario, por otro lado al implementar otros dispositivos y software que trabajan de forma paralela en la misma computadora, es necesario exigir mejores capacidades de cómputo que satisfagan la demanda de recursos que este tipo de aplicaciones requieren; tal es el caso del sistema expuesto en la sección 5.3.1, en la que fue implementado el sensor Kinect, el canal de comunicación “Syphon” y el uso de un software de video mapping.

Para obtener una experiencia completa con la aplicación, es necesario contar con un equipo que cuente con las siguientes características como mínimo: sistema operativo OSX versión 10.8.5 en adelante, procesador intel pertenecientes a las series i5/7 o equivalente en otras marcas, 8 Gb de

memoria RAM y tarjeta gráfica NVIDIA GeForce con al menos 1 Gb de memoria interna. Esta información está basada en características usualmente encontradas en las computadoras para videojuegos y edición profesional de video,[28] además se probó el sistema en diferentes computadoras y se consiguieron resultados positivos a partir de las especificaciones antes mencionadas.

5.3.1.6 Análisis de resultados de interacción

En las aplicaciones de carácter generativo, la interacción juega un papel de vital importancia ya que representa el vínculo entre el usuarios y el sistema que genera animaciones o comportamientos "naturales". Como se ha mencionado, el rendimiento no se ve involucrado cuando en las aplicaciones la interacción no requiere de una carga de procesamiento adicional, es decir cuando no se requiere la implementación de sensores en los que se precisen cálculos complejos, por ejemplo la obtención de la posición del usuario. Por otro lado, este tipo de interacción se basa en acciones simples como mover el mouse, oprimir botones o teclas de la computadora o en algunos casos el uso de la web cam. La intención de este trabajo es demostrar otras alternativas con las que un usuario puede relacionarse con una aplicación capaz de generar sistemas generativos en tiempo real. Para lograr lo propuesto se tuvo que colocar en contexto el tipo de interacción con los recursos disponibles, para el último caso expuesto se implementó el sensor kinect el cual presentó una primera limitante: el "frame rate" del dispositivo alcanza hasta los 30 fps en condiciones de funcionamiento estándares (640 x 480 pixeles de resolución) aunque se ha podido demostrar que es capaz de funcionar a 1280 x 1024 pixeles de resolución aunque con un "frame rate" inferior[29].

En muchos casos la implementación de este sensor puede representar el fracaso de la experiencia de usuario con la aplicación, ya que si los objetos dentro del área de visión del dispositivo se mueven demasiado rápido, entonces se pierde la continuidad de la trayectoria de éstos en los cálculos. Existen otras alternativas para solucionar este problema: la primera, implementar el sensor desarrollado por SONY, el cual posee características similares al de Microsoft pero con la desventaja de la poca documentación existente para dicho dispositivo. La segunda alternativa es emplear cámaras de alta velocidad, como las cámaras Grasshopper (www.ptgrey.com) desarrolladas por la compañía Point Grey, pero con la seria desventaja de los altos costos para el desarrollo de la aplicación e instalación.

Para la última aplicación se optó por modificar el enfoque de la interacción y realizar el análisis de la imagen de forma cenital, lo cual significa que el usuario camina por debajo del sensor. Por tanto

el movimiento del usuario debajo del campo de visión del sensor es mas lento que si éste estuviera moviendo sus manos enfrente del sensor, esta información se traduce en un mejor desempeño de la aplicación y mayor continuidad en las animaciones generadas.

Finalmente esta aplicación está basada en el análisis de la imagen por medio del sistema infrarrojo del dispositivo, por lo que la modificación de la iluminación de los espacios en los que el sistema es instalado, no afecta su funcionamiento dado que el discernimiento de las formas en el campo de visión del Kinect es mediante la información de profundidad. Las únicas situaciones que afectan al funcionamiento de este sistema, son aquellas que aquejan directamente a la luz infrarroja.

5.3.1.7 Análisis de Costos

La demanda de aplicaciones interactivas en las que intervenga el factor visual ha tenido una demanda creciente, sobre todo en el mercado del entretenimiento. Por otro lado otras industrias que demandan esta clase son proyectos son: Marketing, este mercado basa su demanda en los eventos llamados “launching” de productos, los cuales usualmente requieren de instalaciones interactivas que vuelvan mas atractivo el lanzamiento de un nuevo producto. Otro mercado se encuentra en el diseño industrial, el cual constantemente se reinventa e integra nuevas tecnologías, un ejemplo de ello es la creación de “showrooms” o mobiliario capaz de ser interactivo y desplegar información importante y atractiva para los usuarios.

Los costos son calculados entorno a cinco elementos principales: El desarrollo del software, hardware o tecnología, la instalación (como el espacio de interacción, lo cual involucra el diseño y producción del mobiliario necesario), construcción y mantenimiento. Los últimos tres elementos quedan fuera del alcance de este trabajo, ya que involucran la participación del conocimiento experto pertenecientes a otras disciplinas.

En el desarrollo de software y hardware es necesario calcular los costos basados en la complejidad del sistema requerido, el personal requerido para realizar la aplicación, el costo de la tecnología y su implementación. A continuación se muestra el ejemplo de los costos de una aplicación interactiva desarrollada.

Concepto	**Costo**
Instalación	$40,000.00
Software de la instalación	$20,000.00
Mantenimiento *	$3,000.00
Construcción	$15,000.00
Gestión inicial del proyecto	$7,000.00

Total	$85,000.00

*Costo mensual del mantenimiento.

Los costos expuestos en el análisis anterior son representados en pesos mexicanos, dichos valores se obtuvieron a partir de cotizaciones pertenecientes a proyectos efectuados en la industria[30].

Conclusiones

A partir de las aplicaciones desarrolladas, la información recopilada de diferentes autores que recientemente han realizado trabajos relacionados con el tema y las demandas actuales del mercado se ha llegado a una serie de conclusiones, las cuales denotan la importancia de los equipos de trabajo multidisciplinarios capaces de desarrollar proyectos de esta naturaleza.
Se demostró que los algoritmos generativos son una alternativa económica en términos de procesamiento, para generar comportamientos semejantes a los encontrados en la naturaleza, es decir presentan un estado inicial que con el paso del tiempo es modificado de una forma congruente a este primer estado. En otras palabras se busca que la evolución del sistema sea lo mas orgánica posible y con el transcurso de cada ciclo del algoritmo se mantenga la identidad visual de cada aplicación.

Como se ha mencionado, este tipo de algoritmos se ejecutan de manera iterativa variando los valores iniciales del sistema en cada ciclo de ejecución. Si bien se mencionaron una serie de herramientas capaces de poder generar valores pseudo aleatorios, la mas importante a mencionar es el Ruido Perlin, la cual con modificaciones en el propio algoritmo o en los valores evaluados permitió obtener un desempeño en la aplicación mas cercano a lo propuesto en este trabajo. Por otro lado las demás herramientas demostraron ser útiles en casos específicos o simplemente la poca documentación dificultó su implementación. En otros casos otras herramientas demostraron cumplir con las expectativas a cambio de exigir mayor cantidad de recursos computacionales, finalmente se demostró que un usuario es incapaz de determinar a simple vista el grado de aleatoriedad de una aplicación, aunque en el sentido estricto se obtuvieron datos “mas orgánicos”, en términos funcionales, esto no represento una diferencia significativa tal que valiera emplear tales recursos en su implementación.

Como se pudo constatar con las diferentes aplicaciones expuestas en este trabajo de tesis, la programación orientada a objetos es un factor esencial para el desarrollo de sistemas generativos, primeramente por que se trata de un conjunto de elementos independientes que interactúan unos con otros creando patrones y formas que únicamente son posibles de obtener por este medio. Cada elemento del sistema posee características propias que definen su comportamiento dentro del sistema y esta propiedad de individualidad dificulta que el sistema se torne predecible y repetitivo,

es decir que cada ocasión en que el algoritmos es ejecutado, los resultados finales son completamente diferentes.

En algunos de los algoritmos presentados, la función de ruido **noise()** jugó un papel importante para definir el valor inicial de las variables que poseía cada elemento del sistema, es decir una distribución orgánica de sus valores para obtener estados iniciales del sistema completamente aleatorios. Cabe señalar que un estado inicial aleatorio no es por definición una propiedad de los sistemas generativos, pero suele ser un recurso para aumentar la probabilidad de que el estado final del sistema se repita. Por otra parte, otros sistemas no tienen un estado final definido y se encuentran en constante evolución dando lugar a dos casos: el primero en el cual el sistema se aleja de los parámetros iniciales llegando a un punto en el cual el resultado gráfico deja de tener sentido y se convierte en un conjunto de trazos aleatorios. En el segundo caso el sistema oscila orgánicamente alrededor de los parámetros iniciales y dada la condición de individualidad de los elementos del sistemas, constantemente se crean nuevas formas y geometrías. Particularmente se buscó que los algoritmos presentados en este trabajo se encontraran en el segundo caso y un ejemplo claro de ello fue la aplicación sobre partículas y ecuaciones paramétricas en 2D.

Por el lado de la interacción se propusieron de igual manera una serie de medios por los cuales es posible proponer nuevas formas de interacción, así como aplicaciones que requieren de una fuerte carga de innovación. Una de ellas fue el sensor Kinect del cual se estudiaron y explotaron sus características para llevarlo a aplicaciones mas allá de los videojuegos como una herramienta útil con la cual el usuario puede interactuar a distancia y sin necesidad de elevar demasiado los costos en el desarrollo de la aplicación. Sin embargo, si los límites propios del dispositivo no son evaluados en contexto con los alcances de la aplicación, pueden llevar al fracaso la experiencia del usuario y por tanto el proyecto mismo. De igual manera se logró implementar exitosamente un canal de comunicación que permitiera enviar información en forma de imágenes a otro software externo, en este caso video mapping para lograr adecuar la interfaz gráfica a los distintos espacios en los que la aplicación llegase a ser instalado.

A pesar de dirigir los sistemas generativos hacia un enfoque interactivo, la principal problemática se presentó al incorporar formas de interacción mas complejas, las cuales añaden un peso considerable al procesamiento en la aplicación. Por lo que fue necesario delimitar los alcances gráficos de la aplicación así como enfocar la forma de interacción a maneras mas adecuadas y acordes a los recursos disponibles.

La mayoría de las aplicaciones expuestas en este trabajo tienen una demanda moderada de procesamiento a pesar de tratarse de aplicaciones gráficas. Esto logró medirse por medio del aumento de elementos desplegados en el sistema sin afectar el “frame rate” de la aplicación, pero como se mencionó, añadir dispositivos de interacción disminuyen notablemente el desempeño de la aplicación, es decir afectando directamente la velocidad de refresco desplegada. Es por esta razón que la mayoría de los desarrollos expuestos en este trabajo son enfocados alrededor de los algoritmos generativos.

De las herramientas propuestas el Kinect demostró ser una herramienta útil para generar experiencias interactivas a un bajo costo. Actualmente existen otras herramientas similares en cuanto a características pero con capacidades superiores al dispositivo desarrollado por Microsoft, pero con la seria desventaja de los costos de implementación y la falta de documentación, lo cual afecta a los tiempos de desarrollo, tal es el caso de las cámaras Grasshopper las cuales permiten obtener la información a un “frame rate” mayor y con resoluciones superiores.

Una herramienta que se pretendió implementar fue el sensor Leap Motion, el cual se encuentra aún en fase de desarrollo y experimentación, pero que ha demostrado ser una alternativa innovadora para crear experiencias interactivas. Las expectativas generadas por este dispositivo y una segunda versión de éste, han llevado a escasear las existencias de dicho producto, por lo cual los tiempos de entrega con el proveedor se han visto seriamente afectados. Por lo cual las aplicaciones desarrolladas con este dispositivo, por cuestiones de tiempo, quedaron fuera del alcance de este trabajo, pero cabe destacar que se trata de una herramienta útil que cuenta con una amplia documentación que constantemente es actualizada.

Como es posible identificar, casi la totalidad de las aplicaciones fueron desarrolladas con el lenguaje Processing, la razón por la que fue empleado es por que existe una amplia y activa comunidad que constantemente crea nuevas aplicaciones utilizando esta herramienta. Por otra parte se expuso brevemente el alcance de la herramienta P5.js para lenguaje javascript debido a la relevancia que ha tomado en los últimos meses y la creciente demanda de aplicaciones web interactivas. Prueba de ello son son los últimos trabajos y proyectos presentados en importantes festivales de artes, ciencia y tecnología, por ejemplo el festival SXSW en Austin Texas el cual reconoce y premia a las propuestas mas innovadoras en el campo de transmedia y multimedia las cuales pudieron ser desarrolladas gracias a los avances científicos y tecnológicos más recientes. Como se mencionó, existe una basta variedad de lenguajes, frameworks y software con la capacidad de utilizar sus características para crear de igual forma impresionantes aplicaciones interactivas.

Algunas de ellas como vvvv son herramientas con licencia gratuita pero para uso estrictamente personal, si se desea emplearla para fines lucrativos, es necesario adquirir una licencia especial. Otras herramientas como OpenFrameworks y Cinder, a pesar de basarse en lenguajes populares (c++ y Objective C) su estructura no es completamente intuitiva, por lo que la curva de aprendizaje para estas herramientas se puede tornar mas lenta, por otra parte no se ignora el hecho que dichos frameworks son altamente utilizados por la industria hoy en día. Por los motivos explicados, aplicaciones desarrolladas en dichas herramientas a pesar de sus capacidades no pudieron ser incluidas en este trabajo.

La facilidad de aprendizaje dada la similitud con el lenguaje Java y OpenGL, bibliotecas documentadas y una comunidad activa de desarrolladores, fueron los motivos por los que se eligió Processing como principal herramienta de desarrollo en este trabajo.

Actualmente la industria de los medios digitales ha llevado a que las nuevas tendencias integren a la tecnología como vía para la innovación. Artistas, diseñadores, arquitectos e ingenieros convergen en los espacios de trabajo para generar propuestas vanguardistas que promuevan la modificación de los estándares en mercados como: la publicidad, ciudades inteligentes, espectáculos, multimedia, entre otros. De igual manera este movimiento hacia la tecnología, ha llevado a la necesidad de requerir expertos con formación científica y creativa capaces de generar nuevas vías de interacción con la información.

Finalmente los algoritmos generativos son una vía por la cual se representa el cambio o la evolución de un sistema de manera gráfica, permitiendo explorar nuevas formas de representar la información en tiempo real. Esta clase de algoritmos también han permitido explorar nuevas formas o geometría que en trabajo conjunto con disciplinas como la arquitectura y el diseño industrial pueden ser implementadas en la creación de productos. En otras palabras, es posible desarrollar software que por sí mismo sea capaz de generar conceptos visuales, ideas arquitectónicas, experiencias interactivas, por mencionar algunas. Lo anterior no pretende automatizar los procesos creativos de las personas, si no ser una herramienta de apoyo alterna para crear propuestas innovadoras.

Bibliografía

[1] C. M. Bishop and J. Lasserre, Generative or Discriminative? getting the best of both worlds. In Bayesian Statistics 8, Bernardo, J. M. et al. (Eds), Oxford University Press. 2007. pp 3–23.

[2] Celestino Soddu. (1992). Generative Art Geometry. Logical interpretations for Generative Algorithms.. XVII Generative Art Conference , pp 1-11.

[3] Matt Pearson. (2011). Generative Art. Shelter Island, NY: Manning.

[4] Perlin, Ken (July 1985). "An Image Synthesizer". SIGGRAPH Comput. Graph. pp 287—296.

[5] Scolari, Carlos Alberto (2004). Hacer clic. Hacia una sociosemiótica de las interacciones digitales. Barcelona: Gedisa.

[6] Matt Pearson. (2011). Generative Art. Shelter Island, NY: Manning.

[7] Daniel Shiffman. (2012). The Nature Of Code, Simulating Natural Systems With Processing, CA, USA: Daniel Shiffman

[8] Perevalov Denis. (2013). Mastering openFrameWorks: Creative Coding Demystified, Birmingham, UK: Pack Publishing.

[9] Joshua Noble. (2012). Programming Interactivity. Sebastopol, CA: O Reilly. pp 8-10

[10] Rafael Lozano-Hemmer.(2014). Rasero y doble rasero—Standards and Double Standards.

[11] Weichert, Frank; Bachmann, Daniel; Rudak, Bartholomäus; Fisseler, Denis (2013-05-14). "Analysis of the Accuracy and Robustness of the Leap Motion Controller". Sensors (Basel, Switzerland).

[12] Donenfeld, Jason A. "AirTunes 2 Protocol". ZX2C4. Retrieved April 11, 2011.

[13] http://www.nvidia.es/object/jetson-embedded-systems-es.html

[14] Joshua Noble. (2012). Programming Interactivity. Sebastopol, CA: O Reilly. p 21.

[15] Lozano-Hemmer, Rafael. (2015). Pseudomatismos. Mexico City, México: RM

[16] Clinton, Keith. (2010). Agile Game Development With SCRUM. Boston , MA: Addison-Wesley

[17] Joshua Noble. (2012). Programming Interactivity. Sebastopol, CA: O Reilly. p 22.

[18] Fuentes Kraffczyk, J. (2003). Realidad virtual aplicada al tratamiento del trastorno de lateralidad y ubicación espacial. Licenciatura en Ingeniería en Sistemas Computacionales. Universidad de las Américas Puebla.

[19] MAEDA John. (2006) Leyes de la Simplicidad, diseño, tecnología, negocios y vida. Gedisa Editorial. Primera edición.

[20] Roger S. Pressman. (1998). Ingeniería del Software un enfoque práctico. España: McGrawhill

[21] Greg James, “Operations for Hardware-Accelerated Procedural Texture Animation” in “Game Programming Gems 2” ed. Mark DeLoura, Charles River Media, 2001, p 497.
[22] Daniel Shiffman. (2012). The Nature Of Code, Simulating Natural Systems With Processing, CA, USA: Daniel Shiffman, p 150.
[23] Bohnacker Hartmut; Grob Benedikt; Laub Julia. (2012). Generative Design. Princeton Architectural Press. New York, USA. pp 218-235.
[24] Bohnacker Hartmut; Grob Benedikt; Laub Julia. (2012). Generative Design. Princeton Architectural Press. New York, USA. pp 292.
[25] Daniel Shiffman. (2012). The Nature Of Code, Simulating Natural Systems With Processing, CA, USA: pp 89-97.
[26] McCarthy Lauren; Reas Casey; Fry Ben. (2015). Getting Started with P5.js. Maker Media. CA, USA: p XI.
[27] http://shiffman.net/learning/
[28] https://www.videomaker.com/article/c3/15264-recommendations-for-the-best-video-editing-computer
[29] Gibson, Ellie (June 5, 2009). "E3: Post-Natal Discussion". Eurogamer. Eurogamer Network. pp. 1–2. Retrieved June 9, 2009.
[30] https://www.wirespring.com/dynamic_digital_signage_and_interactive_kiosks_journal/articles/Budgeting_for_an_Interactive_Kiosk_Project-200.html