KnE Engineering

SIIPRIN Conference Proceedings
Simposio Iberoamericano en Programación Informática
(Ibero-American Symposium on Computer Programming)
SIIPRIN'2017

Knowledge E
Engaging minds

Conference Paper

Scientific Computing and the Huygens' Principle

Alonso Álvarez, Narcisa Salazar, and José Tinajero

Escuela Superior Politécnica de Chimborazo, Código Postal 060155, Riobamba, Chimborazo, Ecuador

Corresponding Author:
Alonso Álvarez
aalvarez@espoch.edu.ec

Received: 28 July 2017
Accepted: 5 September 2017
Published: 30 January 2018

Publishing services provided by Knowledge E

© Alonso Álvarez et al. This article is distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use and redistribution provided that the original author and source are credited.

Selection and Peer-review under the responsibility of the SIIPRIN Conference Committee.

Abstract

Mathematics has been present in the development of society since time immemorial; great figures have dedicated their entire life to analysis and research in various branches of this broad science. Scientific Computing is closely related to the design and construction of mathematical models aimed at solving scientific, social and engineering problems. There are several applications of this discipline, for example for mathematical simulations of differential equations in partial derivatives that describe the propagation of a variety of waves such as sound waves, or heat conduction problems in different media, which can be solved using The Fourier Analysis. Through Graphical Computing the Huygens Principle can be verified. All these models can be implemented through the computer in order to facilitate the complex calculations that must be done to solve problems of this type and depending on the case to condense all this information into a graph "A good graph says more than a thousand words (Chinese Proverb)".

Keywords: Mathematics, Scientific Computing, Graphical Computing, Huygens Principle.

1. Introducción

La Computación Científica ha jugado tradicionalmente un papel muy importante en el avance de la informática. En los inicios de ésta era prácticamente la única fuerza impulsadora, proporcionando los problemas que eran motivación para el desarrollo tanto de software como de hardware. La Computación Científica acabó convirtiéndose en sinónimo de cálculo intensivo, de gráficos por computadora y de grandes máquinas (ordenadores vectoriales, etc.) con un número muy limitado de usuarios que se ocupaban de problemas extremadamente técnicos y muy lejos de cualquier aplicación inmediata.

OPEN ACCESS

2. Computación Científica

Computación Científica es la ciencia aplicada que se encarga de desarrollar los MODELOS MATEMÁTICOS y computacionales de los procesos vinculados a los problemas científicos o tecnológicos de las ciencias naturales o de la ingeniería. Estos modelos sirven para manipular y controlar el problema real al que representan. La Computación Científica tiene como objetivo proveer una base matemática y computacional a las simulaciones numéricas la reconstrucción o predicción de los fenómenos y procesos de la ciencia y la ingeniería mediante modelos computacionales. La Computación Científica es una vasta área multidisciplinaria que abarca aplicaciones (ciencia/ingeniería), matemática (análisis numérico), computación y computación gráfica. [1]. (Figura 1).

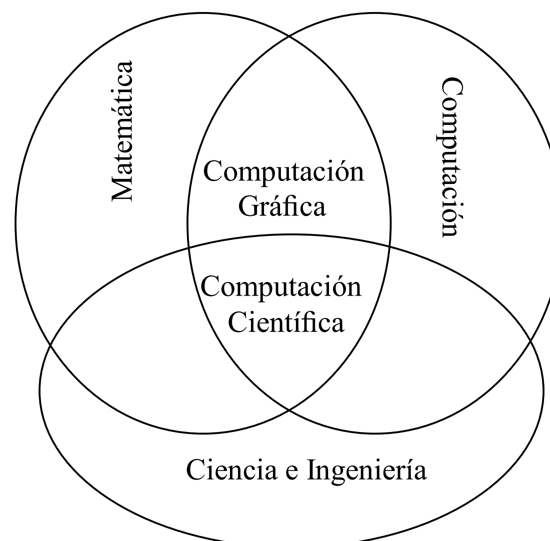

Figura 1: Relación de Computación Gráfica con otras áreas.

La Computación Gráfica es la máxima expresión visual de la Matemática en el contexto computacional. La abstracción de los modelos matemáticos se concretiza a través de algoritmos que hacen posible la programación de componentes importantes dentro de las herramientas computacionales. [2]

Modelos computacionales y simulaciones computacionales ha resultado una parte importante del repertorio de la investigación, ayudando o reemplazando, en algunos casos, a la experimentación, tal como en la:

- Reconstrucción y comprensión de escenarios conocidos (desastres naturales).
- Optimización de escenarios conocidos (procesos técnicos).
- Predicción de escenarios desconocidos (clima, nuevos materiales).
- Gráficos de modelos físicos.

3. Principio de Huygens

Mediante el estudio de los patrones de interferencia puede verificar el principio de Huygens, según la cual una onda con el plano de frente de onda puede ser vista como el efecto de las ondas puntiformes infinitas alineados a lo largo de una línea recta. [3]

Huygens visualizó un método para pasar de un frente de onda a otro. Cuando el movimiento ondulatorio alcanza los puntos que componen un frente de onda, cada partícula del frente se convierte en una fuente secundaria de ondas, que emite ondas secundarias (indicadas por semicircunferencias) que alcanzan la próxima capa de partículas del medio. Entonces estas partículas se ponen en movimiento, formando el subsiguiente frente de onda con la envolvente de estas semicircunferencias.

El proceso se repite, resultando la propagación de la onda a través del medio. Esta representación de la propagación es muy razonable cuando la onda resulta de las vibraciones mecánicas de las partículas del medio, es decir una onda elástica pero no tendría significado físico en las ondas electromagnéticas donde no hay partículas que vibren. [4].

3.1. Modelos Matemáticos

3.1.1. Modelo Matemático para la ecuación de ondas con una fuente puntiforme

Supongamos de ubicar una fuente de ondas en la posición (cxi, cyi).

Sea $(x, y) \in R^2$ un punto de nuestro espacio de estudio:

$$ki = W\,[d((x,y); (cxi,cyi)) - tp*v]$$

Formula 1: Argumento para la ecuación de ondas

W (Velocidad angular) (W = 1.5)

v (Velocidad de propagación) (v = 9.3)

tp (tiempo) (tp = 1)

d ((x,y); (cxi,cyi)) (distancia de (x, y) a (cxi, cyi))

$$Zi = \text{Seno}\,(Ki) + 1$$

Formula 2: Modelo para la ecuación de ondas

3.1.2. Modelo Matemático para verificar el Principio de Huygens

Para probar el principio vamos asignar 21 fuentes alineados a lo largo del eje x en una distancia de 1 cm y se colocan entre -10 cm y 10 cm. Estudiando la figura de interferencia de las ondas podemos verificar el Principio de Huygens.

$$Z = \sum_{i=-10}^{10} Zi$$

Formula 3: Modelo para el principio de Huygens

Para demostrar este principio se implementará el modelo matemático mediante computación gráfica. La complejidad de los cálculos requiere mayor tiempo de ejecución por lo que se implementará el programa a través del uso de hilos en JAVA.

4. Implementación en JAVA

4.1. JAVA

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. [5]

La idea de Java es que pueda realizarse programas con la posibilidad de ejecutarse en cualquier contexto, en cualquier ambiente, siendo así su portabilidad uno de sus principales logros. [6]

4.2. Procesamiento multihilos en JAVA

El procesamiento multitarea permite ejecutar varios procesos a la vez, por tanto, se puede desarrollar programas con menos tiempo de ejecución haciéndolos más eficientes, pero la ejecución simultanea tiene límites, debido a las condiciones propias del hardware. Este límite viene dado por el número de núcleos que tenga el computador, actualmente todos los computadores tienen al menos dos núcleos, entonces podríamos ejecutar dos procesos a la vez en el peor de los casos. El sistema operativo divide el tiempo de procesamiento, no solo entre las diferentes aplicaciones, sino también entre cada hilo dentro de una aplicación, agilitando el flujo de la aplicación.

En el procesamiento multitarea varios procesos comparten los recursos de un computador, es decir, se puede dividir procesos específicos dentro de una sola aplicación en hilos diferentes y cada uno de los hilos se puede ejecutar en paralelo.

Java permite la creación de programas con varios hilos para la ejecución de procesos. De hecho, todos los programas con diferente gráfico como AWT o Swing son multihilos ya que el dibujado de las ventanas corren en un thread distinto al principal.

Hay dos formas de implementar la ejecución hilos en Java, se utiliza la extensión a la clase Thread o la implementación de la interfaz Runnable.

La diferencia entre Interfaz y Clase es que una interfaz solamente puede contener métodos abstractos y variables estáticas y finales, es decir constantes, mientras que las clases pueden implementar métodos y contener variables que no sean constantes. Además, una interfaz no puede implementar cualquier método y una clase que implemente una interfaz debe implementar todos los métodos definidos en esa interfaz. Una interfaz puede extenderse de múltiples interfaces a diferencia de las clases.

Para ejercicios no muy complejos donde las clases no heredan de otras la clase Thread funciona muy bien. Basta con extender Thread y sobrescribir el método run(). Esta clase encapsula el control de hilos de ejecución. Es muy común confundir un objeto Thread con un hilo de ejecución.

Sin embargo, cuando las aplicaciones se complican y las clases extienden otras es mejor emplear la interfaz Runnable. De hecho, la interfaz Runnable es la única forma de implementar multihilos en Java, para este caso, ya que el IDE no soporta la herencia múltiple.

5. Desarrollo

Se implementa la interfaz Runnable para la clase Tapete

```
public class Tapete extends Vector implements Runnable {...
```

Se agrega los atributos I, t, i que representan un vector de BufferedImage, el tiempo y la posición inicial para cada hilo

```
private BufferedImage[] I;
private double t;
private int i;
```

Creamos un constructor que además de definir la paleta de colores, nos permita inicializar estos valores para cada objeto creado

```
public Tapete(BufferedImage[] I, double t, int i){
this.I = I;
this.t = t;
this.i = i;
```

```
}
```

Se implementa el método threadEncender () que contiene los cálculos que generan la onda pero difiere del encender() conocido en que recibe el tiempo inicial como entero y devuelve un bufferedImage

```
public BufferedImage threadEncender(double tp) {
BufferedImage I = new BufferedImage(600, 500, BufferedImage.TYPE_INT_ARGB);
double W = 1.5;
double v = 30;
double K1;
double Z = 0, Z1 = 0;
for (int i = 0; i < 600; i++) {
for (int j = 0; j < 500; j++){
for (double m = -10; m <= 10; m++) {
procesoCarta(i, j);
double d1 = Math.pow(Math.pow(Sx + m, 2) + Math.pow(Sy, 2), 0.5);
K1 = W * (d1 - tp * v);
Z1 = Math.sin(K1) + 1; Z += Z1;
}
int col1 = (int) (Z * 0.38);
color = (Color) P1.get(col1);
pixel(i, j, I);
Z = 0;}
}
return I;
}
```

Naturalmente en la Figura 2 se puede observar que el frente de onda no aparece totalmente alineado, porque las fuentes son pocas y a una considerable distancia, sin embargo el efecto generado es convincente de la validez del Principio de Huygens.

Nota: Para trazar el gráfico se utilizó la siguiente paleta de colores

```
private void P1(){
this. P1.add(Color.BLACK);
this. P1.add(Color.decode("#000080"));//NAVY
this. P1.add(Color.GREEN);
```

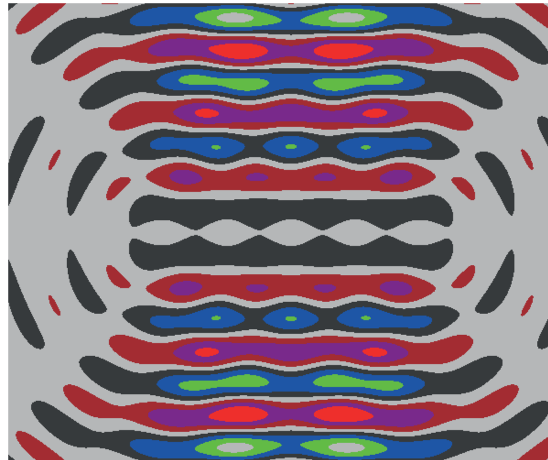

Figura 2: Principio Huygens.

```
this. P1.add(Color.decode("#00FFFF"));//AQUA o CYAN
this. P1.add(Color.RED);
this. P1.add(Color.decode("#800080"));//PURPLE
this. P1.add(Color.decode("#800000"));//MAROON
this. P1.add(Color.LIGHT_GRAY);
this. P1.add(Color.DARK_GRAY);
this. P1.add(Color.BLUE);
this. P1.add(Color.decode("#00FF00"));//LIME
this. P1.add(Color.decode("#C0C0C0"));//SILVER
this. P1.add(Color.decode("#008080"));//TEAL
this. P1.add(Color.decode("#FF00FF"));//FUCHSIA
this. P1.add(Color.YELLOW);
this. P1.add(Color.WHITE);
}
```

5.1. Animación

Vamos a simular una animación cinematográfica del Principio de Huygens. Escogemos 9 instantes temporales consecutivos distanciados en 1/20 segundos del proceso Huygens. De esta forma luego de 9 instantes temporales se completará un periodo T ya que de acuerdo a los datos provistos por las constantes resulta que T es igual a 9/20 segundos.

Para cada instante se traza la forma del Principio de Huygens sobre el monitor (fotogramas) y mediante la Programación Concurrente, los fotogramas aparecerán velozmente una a continuación de otra, simulando de esta manera el movimiento de las ondas.

Implementamos el método run(). Este método crea los fotogramas y los almacena en el vector de bufferedImage. La implementación de sobre escritura de este método es obligatorio ya que será llamado por el método start() del hilo correspondiente para la ejecución del proceso deseado.

```
@Override
public void run() {
double dt = 0.05;
for (int j = i; j < 100; j += 9) {
I[j] = threadEncender(t / 18);
t += dt;
}
}
```

Ahora implementamos la clase ThreadLienzo que extiende de Thread. El método run() de esta clase recorrerá el vector de fotogramas, pasado en su constructor, para presentar cada uno de ellos

```
public class ThreadLienzo extends Thread {
private AreaTrabajo AreaTrabajo;
private BufferedImage[] I;
public ThreadLienzo(AreaTrabajo AreaTrabajo, BufferedImage[] I) {
this.AreaTrabajo = AreaTrabajo;
this.I = I;
}
@Override
public void run() {
espera(5000);
for (int i = 0; i < 200; i++) {
if (I[i] == null) {
espera(20); i–;
}
else{
```

```
AreaTrabajo.setLienzo(I[i]);
espera(100);
AreaTrabajo.pintar();
}
}
}
```

Dentro del botón que ejecuta el proceso en la interfaz, se incluye la creación del vector de BufferedImage que almacenará los fotogramas. Además se crea 9 objetos de la clase Tapete y se les asigna el tiempo y posición del vector correspondiente para la ejecución del proceso run(). Estos serán ejecutados como hilos mediante new Thread(obj).start(). Cada objeto tipo Tapete creara una parte del total de fotogramas que serán presentados por el objeto Lienzo de forma paralela conforme se vayan generando.

```
private void btnDemostracionActionPerformed(java.awt.event.ActionEvent evt) {
BufferedImage[] foto = new BufferedImage[1000];
ThreadLienzo Lienzo = new ThreadLienzo();
double n = 0.0;
int i = -1;
Tapete oThread1 = new Tapete (foto, n += 0.1, ++i);
Tapete oThread2 = new Tapete (foto, n += 0.1, ++i);
Tapete oThread3 = new Tapete (foto, n += 0.1, ++i);
Tapete oThread4 = new Tapete (foto, n += 0.1, ++i);
Tapete oThread5 = new Tapete (foto, n += 0.1, ++i);
Tapete oThread6 = new Tapete (foto, n += 0.1, ++i);
Tapete oThread7 = new Tapete (foto, n += 0.1, ++i);
new Thread(oThread1).start();
new Thread(oThread2).start();
new Thread(oThread3).start();
new Thread(oThread4).start();
new Thread(oThread5).start();
new Thread(oThread6).start();
new Thread(oThread7).start();
System.out.println("PINTAR");
```

```
Lienzo.I = foto;
Lienzo.AreaTrabajo = AreaTrabajo;
Lienzo.start();
}
```

La captura evidencia el tiempo registrado para la ejecución de la animación.

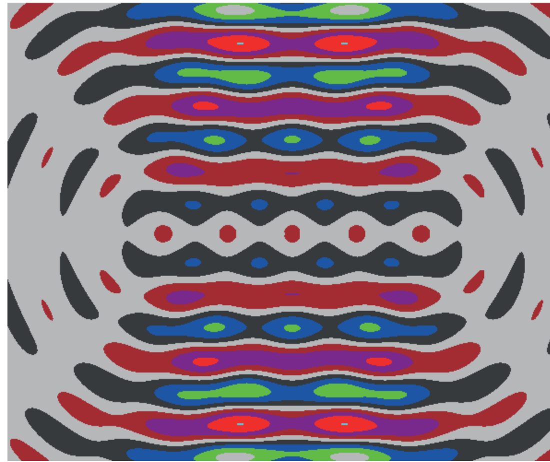

Figura 3: Fotograma tp=1.05.

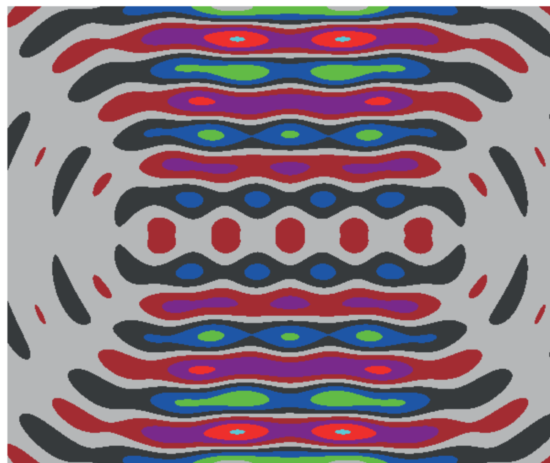

Figura 4: Fotograma tp=1.1.

6. Conclusiones

Analizando las ecuaciones y los gráficos resultantes se puede concluir que los puntos de máxima altura son blancos, los de mínima altura son negros y los puntos de altura nula se encuentran entre los colores gris obscuro y gris claro.

A pesar que el número de fuentes (21) no es lo suficientemente grande y que la separación entre ellas es considerable si se puede apreciar la veracidad del principio

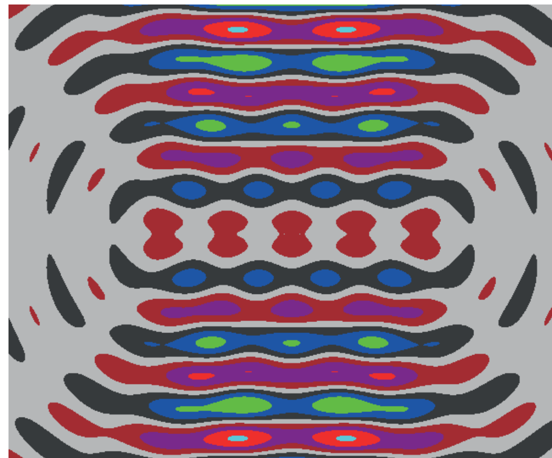

Figura 5: Fotograma tp=1.15.

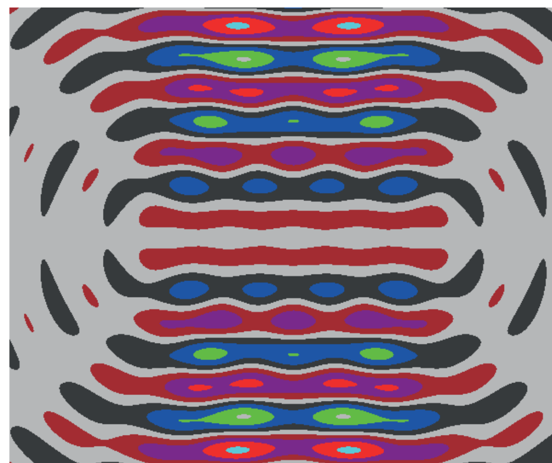

Figura 6: Fotograma tp = 1.2.

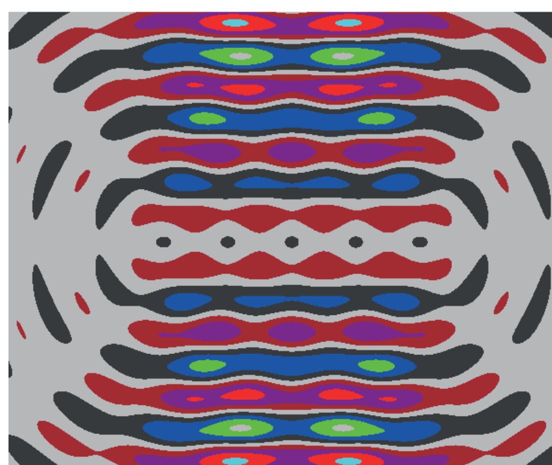

Figura 7: Fotograma tp = 1.25.

de Huygens. Cabe resaltar que al realizar la simulación con 41 fuentes a una distancia menor entre fuentes los gráficos resultantes no sufren cambios considerables, sin embargo el tiempo de proceso si fue notorio. Además se pudo observar que con una

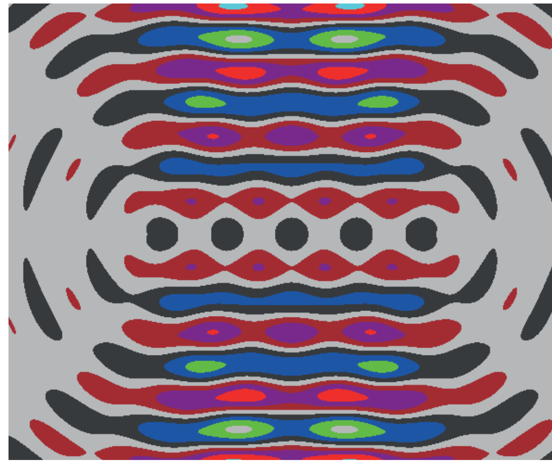

Figura 8: Fotograma tp = 1.3.

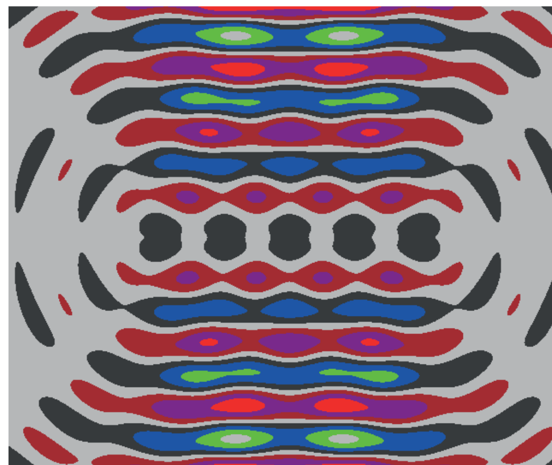

Figura 9: Fotograma tp = 1.35.

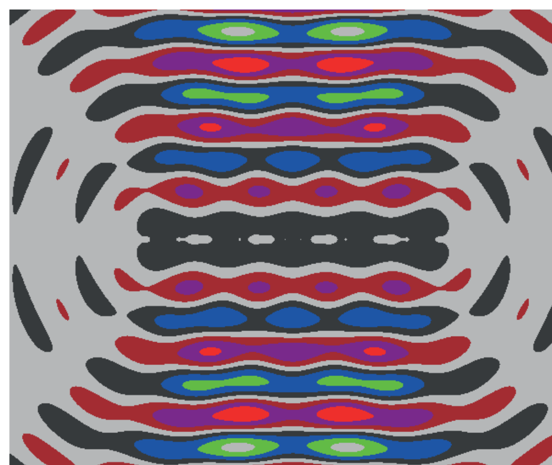

Figura 10: Fotograma tp = 1.4.

computadora de características aceptables (6 gigas en RAM, Core (TM) 2 DUO, 500 gigas en disco) se logró hacer una aceptable animación en tiempo real.

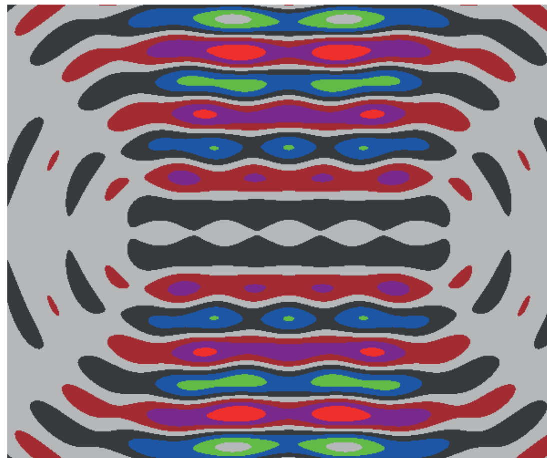

Figura 11: Fotograma tp = 1.45.

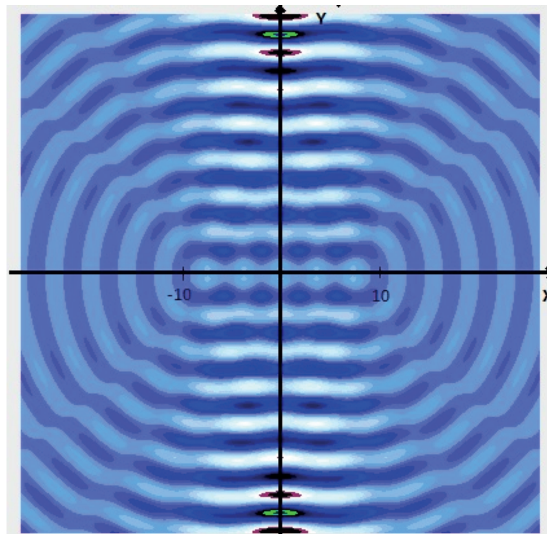

Figura 12: Grafica General de la Simulación en un solo tono de colores.

Para la implementación se utilizó 9 hilos que se encargan de generar y almacenar los fotogramas en un vector y un hilo que realiza la presentación de los fotogramas. Debido al procesamiento en paralelo fue necesario darle un retardo inicial al hilo encargado la animación mientras los fotogramas iniciales se generan. El segundo retardo permite la visualización de los fotogramas al recorrer el vector.

Referencias

[1] V. Martín, ¿Tecnicas de Computación Científica,¿ 2015. [En línea]. Available: http://www.personal
.fi.upm.es/~vicente/tcc/tcc.html.

[2] L. Castañeda, ¿Computación Gráfica y Visual,¿ 2009. [En línea]. Available: https://sites
.google.com/site/leissicl/computacion-grafica-y-visual.

[3] F. Ruiz, ¿Principio de Huygens,¿ 2005. [En línea]. Available: http://acacia.pntic.mec.es/~jruiz27
/huygens/huygens.html.

[4] J. Byous, ¿Java technology,¿ The early years, 1998. [En línea]. Available: http://www
.oracle.com/technetwork/java/index.html.

[5] Facultad de Informatica, Universidad Complutense, ¿NetBeans,¿ 2015. [En línea]. Available: www.fdi.ucm.es.

[6] G. J. C. Zambrano, ¿Definición de Java,¿ 2016. [En línea]. Available: http://desarrollodesoftware
ijhondhalin.blogspot.com/2016/09/java.html.

[7] Bini, D.; Menchi, O., "Matematica, mondo reale e calcolatore", Zanichelly, Italia. 2001.

[8] M. Domínguez, ¿Matemáticas y computación científica,¿ 2014. [En línea]. Available: https://sites.google.com/site/unmsmcc/-que-es-computacion-cientifica.

[9] E. Peña, ¿Computación Científica, San Marcos,¿ 2013. [En línea]. Available: https://camp
.ucss.edu.pe/ingenium/index.php/sistemas/132-computacion-cientifica..