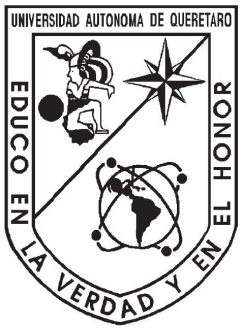

Universidad Autónoma de Querétaro

Facultad de Ingeniería

Licenciatura en
Matemáticas Aplicadas

GPU Math: Biblioteca para el uso computacional de la GPU

TESIS

Que como parte de los requisitos para obtener el grado de

Licenciado en Matemáticas Aplicadas

Presentan:

Alcántara López Fernando Javier
y
Mota Escamilla Juan Carlos

Dirigido por:

Dr. Roberto Augusto Gómez Loenzo

Centro Universitario
Querétaro, Qro.
Enero de 2013
México

RESUMEN

En el presente trabajo se describe la realización del proyecto que consistió en desarrollar una biblioteca matemática con el uso del chip contenido en la tarjeta gráfica llamado GPU. Este trabajo se compone de cinco capítulos y un anexo. El primer capítulo describe los conocimientos más básicos acerca de la computadora que una persona debe saber para poder comprender los siguientes capítulos, ya que habla además del dispositivo clave que se exploto en la realización del proyecto en el que se basó esta tesis, dicho dispositivo es la tarjeta gráfica y la GPU. El segundo capítulo describe toda la información referente a la programación que se utilizó; esto comienza con la descripción de las diferentes maneras que ya existen para poder uso de la GPU y la manera que se utilizó en el presente trabajo, describiendo además las ideas claves para poder instruir a la GPU para que realice las operaciones que uno desea. En el tercer capítulo se profundiza sobre la manera en como se construyó la biblioteca además de describir las diversas funciones que se decidieron incluir en dicha biblioteca y por supuesto las razones que se tuvieron para ello. En el cuarto capítulo se muestran algunos resultados obtenidos con la biblioteca, mostrando además algunas comparaciones realizadas con los tiempos de ejecución que tuvieron dichos algoritmos ejecutados sobre la CPU. Como resultado destacable y adicional se habla sobre un software construido con ayuda de la biblioteca llamada GPU Math, dicho software fue llamado OPTICT. En el quinto capítulo se escriben las conclusiones de trabajar con la GPU y las obtenidas con GPU Math. Posterior al quinto capítulo se encuentra un anexo, en el cual se escribió el código desarrollado para una de las partes de la biblioteca GPU Math; la razón por la cual no se escribió el código completo es debido a que éste es demasiado extenso, para aquellas personas que deseen conocer más acerca del código y los ejecutables desarrollados, éstos se encuentran guardados en los discos entregados a las bibliotecas de la facultad de Ingeniería y Central de la Universidad Autónoma de Querétaro.

Palabras clave: Tarjeta gráfica, GPU, CPU, shader.

A mi madre Alicia López Nuñez
por todo su apoyo.
Por que gracias a ti soy quien soy.

AGRADECIMIENTOS

Recuerda agradecerle a Robertito.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

I. INTRODUCCIÓN

En la actualidad, todas las industrias y todas las empresas exigen que sus procesos sean cada ves más rápidos y eficientes pues requieren agilizar la generación de sus productos a la vez que éstos deben tener mayor calidad y menor costo. Todo lo anterior es necesario para que las empresas e industrias puedan *competir* en este mundo globalizado.

Existe un gran conjunto de procesos industriales y empresariales que constantemente están en evolución y casi siempre tienden a la eficiencia y a la calidad. Como un ejemplo podemos mencionar a los materiales que cada vez son más fuertes y livianos, o a la metodología de producción, etc. Pero existe un subconjunto de procesos que también evoluciona rápidamente y de los que se hablará parcialmente en esta tesis: son los **procesos computacionales**. Dichos procesos están involucrados en empresas, industrias, centros de investigación, etc. y por ello se ha vuelto muy importante su desarrollo e innovación.

Los procesos computacionales son algoritmos que se ejecutan en una computadora y se encargan de resolver alguna tarea específica. Este subconjunto de procesos a su vez posee otro subconjunto que son los **procesos computacionales matemáticos** (también conocidos como **algoritmos numéricos**), los cuales son los responsables de resolver problemas que involucran el procesamiento de números.

El tema central de la presente tesis son los procesos computacionales matemáticos realizados con ayuda de la unidad central de procesamiento de una tarjeta gráfica. Se hablará de la tarjeta gráfica de una computadora y de como la misma es aprovechada para realizar cálculos de mucha rapidez: la capacidad de procesamiento de una tarjeta gráfica supera por muchas veces (de 40 hasta 300 veces, y más) a la capacidad del procesador central de una computadora. Pero principalmente se hablará del proyecto de creación de una biblioteca matemática implementada con base en la tarjeta gráfica. Esta biblioteca constará de múltiples algoritmos incorporados en la tarjeta gráfica, los cuales provendrán de diversas áreas de las matemáticas numéricas con el fin de brindar una idea general del potencial de las tarjetas gráficas, esto exhibiendo cálculos mucho más rápidos a los que se podrían obtener implementándolos en la CPU.

Las áreas de las matemáticas de las que se tomarán los algoritmos que se implementaran en GPU Math son las siguientes: álgebra lineal, cálculo, análisis numérico y procesamiento de imágenes.

A continuación se darán algunos conceptos introductorios básicos.

I.1. Conceptos fundamentales

Existen dos conceptos fundamentales en la computación: *hardware* y *software*.

El ***hardware*** es el conjunto de componentes físicos de una computadora. Por ejemplo, el *hardware* de una computadora personal está conformado por monitor, teclado, mouse, tarjeta gráfica, etc.

El ***software*** es el conjunto de programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación. Por ejemplo, entre el *software* más común se encuentran procesadores de texto, código fuente, documentación, etc.

En las siguientes subsecciones se explicará brevemente algunos componentes de *hardware* y *software* que son de vital importancia para la realización de GPU Math.

I.1.1 Hardware

Tarjeta gráfica

Una tarjeta gráfica es un dispositivo que se encuentra en toda (o en casi toda) computadora de hoy en día. ésta se encarga de procesar todos los datos que provienen del CPU para después transformarlos en datos (imágenes, escenas 2D, escenas 3D, ventanas, etc.) que puedan ser procesados por dispositivos de salida tales como el monitor.

Los componentes básicos de una tarjeta gráfica son: procesador de graficos (GPU), conexiones/bus de entrada (PCI, etc.), interfaz de salida (VGA, HDMI, etc.), dispositivos de refrigeración, alimentación de voltaje y memoria gráfica de acceso aleatorio.

La GPU (del inglés, *Graphics Processing Unit*) es la unidad central de procesamiento de gráficos, un chip que contiene varios microprocesadores donde se procesa toda la información antes de desplegarse en la pantalla. La GPU, al encargarse del procesamiento de gráficos, desliga de una gran carga de trabajo al procesador central con el fin de que éste pueda dedicarse a otras funciones. Los microprocesadores de la GPU son los encargados de procesar la información para cada pixel. En las secciones posteriores se retomará el tema de la GPU para explicar a mayor detalle su funcionamiento.

Las conexiones/buses de entrada son todos aquellos puertos por los que entra información a la tarjeta gráfica. Por otro lado, las conexiones de salida son los puertos por los que la tarjeta gráfica envía información a otro dispositivo.

Los sistemas de refrigeración generalmente se componen de uno o varios ventiladores así como de un disipador, es decir, de una placa de metal, generalmente de aluminio en las partes más propensas a calentarse.

La alimentación eléctrica para las tarjetas gráficas no suele ser difícil de administrar, sin embargo, puesto que el continuo mejoramiento a las tarjetas gráficas exigen una mayor cantidad de energía, en ocasiones se agrega un conector adicional, el cual permite una conexión directa entre la fuente de alimentación y la tarjeta gráfica.

La memoria gráfica está formada por chips de memoria que almacenan y transportan información; existen dos tipos de memoria gráfica:

Dedicada Es la que la GPU utiliza exclusivamente para sí.

Compartida Es cuando se utiliza memoria RAM de la unidad central de procesamiento (CPU).

Hay que hacer notar que la memoria dedicada aporta una mayor eficiencia que la compartida y es la que mejores resultados da debido a que no es necesario emplear el bus de datos de la computadora para leer/escribir información en la memoria dedicada.

Las características de la memoria gráfica son: la **capacidad**, la cual determina el número máximo de datos y texturas que pueden ser procesadas por la memoria gráfica; la

velocidad de memoria, ésta nos dice la velocidad a la que se pueden transportar los datos procesados; y el **ancho de banda**, el cual nos informa de la tasa de datos que se pueden transportar por unidad de tiempo, generalmente medido en gigabytes por segundo(GB/s).

En la figura 1.1 se pueden apreciar los componentes más importantes de una tarjeta gráfica.

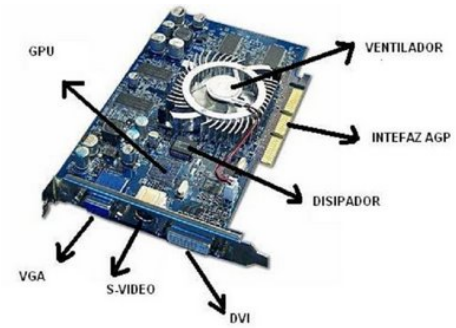

Figura 1.1: Partes de una tarjeta grafica.

I.1.2 Software

El lenguaje de la programación

Como se comento al inicio de la introducción, esta tesis describirá la creación de una biblioteca matemática, el cual utilizará el por demás conocido lenguaje de programación C/C++ [1]. A continuación se hablará un poco acerca de este lenguaje y la manera en como se suelen utilizar diversas técnicas para realizar los algoritmos y programas que vemos cotidianamente en nuestras computadoras.

Para aclarar ideas, es necesario decir que un lenguaje de programación es un idioma artificial creado para expresar procesos que una máquina pueda llevar a cabo, este idioma artificial es utilizado para controlar el comportamiento físico o lógico de una máquina. Dicho idioma está creado por un conjunto de reglas sintácticas, símbolos y semánticas que definen su estructura y el significado de sus expresiones.

Es necesario hacer destacar que al proceso de escribir, probar, depurar, compilar y mantener el código fuente de un programa informático se le llama programación. Este proceso incluye la aplicación de procedimientos lógicos, dichos procedimientos incluyen los siguientes pasos:

1. El desarrollo lógico del programa
2. Escritura de la lógica del programa empleando un lenguaje de programación
3. Compilación del programa para convertirlo en lenguaje de máquina
4. Prueba y depuración del programa

[1] Lenguaje creado en 1972 por el matemático aplicado Dennis Ritchie, dicho lenguaje es tipificado de medio nivel y esta orientado a la implementación de Sistemas Operativos

Ahora que hemos aclarado lo que es un lenguaje de programación hablaremos de las diferentes maneras de escribir un programa.

Programación estructurada

La programación estructurada era una técnica muy interesante en 1960, ya que además de permitir programas fiables y eficientes ayudaba a la comprensión posterior de este mismo.

Dicha técnica utilizaba solamente tres estructuras: **secuencial**, **selección** e **iteración**. La estructura secuencial es aquella donde se ejecutan las instrucciones una tras otra, es decir, de manera secuencial. La estructura de selección también llamada **selectiva** permite bifurcaciones en el programa a una o varias instrucciones. La estructura iterativa es aquella donde se utilizan los bucles iterativos, es decir, instrucciones que se repitan mientras se cumple una condición.

Puesto que en la actualidad se es más ambicioso, principalmente por la interfaz gráfica que se suele implementar a los actuales programas, ya es insuficiente el uso de la programación estructurada.

Es necesario hacer entender que en aquella época se utilizaba mucho el comando **GOTO**, una de las principales razones para la dificultad posterior de entender un programa; hasta que surgió la programación estructurada que, con ayuda del Teorema del programa estructurado, ayudo a que se utilizara más esta técnica para mejorar el nivel con el que se hacían los programas en aquel entonces.

Programación modular

La programación modular es una técnica de programación que consiste en dividir un programa en subprogramas más simples de manera sucesiva hasta obtener subprogramas que se puedan resolver fácilmente. A este proceso se le conoce como refinamiento sucesivo, análisis descendente, entre otros nombres.

Esta técnica surgió como una evolución a la técnica de la programación estructurada.

Se le conoce como **módulo** a cada parte de un programa que resuelve uno de los subprogramas en que se dividió el programa original; en la actualidad, a los módulos, se les suele conocer más comúnmente como procedimientos o funciones. Sin embargo, cabe aclarar que esto no es del todo correcto, ya que un módulo puede constar de varias funciones o procedimientos.

Programación orientada a objetos

La programación orientada a objetos o POO en su forma abreviada utiliza varias técnicas, tales como la herencia, la abstracción, el polimorfismo y el encapsulamiento. Además de que utiliza **objetos** para sus interacciones. Un objeto es una entidad que tiene un determinado *estado*, *método o comportamiento* e *identidad*.

El estado de un objeto se refiere a los datos o informaciones de los que este está compuesto, puede ser uno o varios atributos.

El comportamiento hace referencia a los métodos a los que responde dicho objeto.

La identidad del objeto es la propiedad que lo caracteriza de otros objetos, es decir, su identificador.

La POO es la técnica más usada y solicitada en la actualidad, la cual permite a los programas ser valiosos por sí mismos y no necesariamente por lo que realizan, ya que un

algoritmo que utiliza la POO puede cambiar su interfaz gráfica sin gran dificultad como se observa en la imagen 1.2.

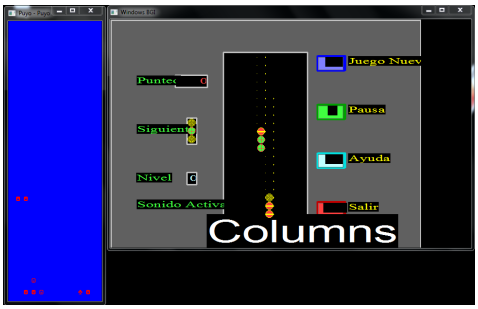

Figura 1.2: Ejemplo de un mismo algoritmo con diferentes interfaces creado con ayuda de la programación orientada a objetos.

En la imagen anterior se muestran dos programas ejecutable semejantes al tetris que todos conocemos, la destacable en la imagen anterior son las diversas diferencias en cada ejecutable, más aún debido a que utilizan el mismo algoritmo a base de POO.

Ahora se retomará el tema de la GPU ya que la presente tesis se basa en explotar los beneficios de la misma. Las siguientes secciones explicarán el funcionamiento de una GPU y sus ventajas.

I.2. La GPU

Una GPU como ya se dijo es la unidad que procesa los gráficos. En un principio las GPUs solamente se dedicaban a acelerar el trabajo de la memoria pero más tarde se agregaron primitivas para acelerar cálculos geométricos como lo son la rotación y la traslación de vértices en diferentes sistemas coordenados, además de facilitar el dibujado de triángulos, círculos, rectángulos, etc. Recientemente, se incluyó en las GPUs un soporte para *shaders* programables, los cuales pueden ejecutar diversas tareas tales como el producto de matrices y muchos otros cálculos que en general se dedican a realizar cálculos para desplegar los gráficos de una manera más rápida; el hecho de que ya se puedan programar los *shaders*, por sí solos ya representan una gran ventaja debido a la estructura paralela que poseen las GPUs. De esto se hablará con mayor profundidad a continuación, pero para entender esto es necesario entender la evolución que han tenido las GPUs. Por ello describiremos en breve el funcionamiento de las GPUs clásicas para entender el funcionamiento de las GPUs actuales.

I.2.1 Funcionamiento de la GPU

La descripción que se dará a continuación es con el fin de entender el largo camino que han recorrido las GPUs, tanto por su complejidad estructural como su funcionamiento interno que se describirá a continuación. En la actualidad las GPUs pueden implementar diversos algoritmos en paralelo.

Debido a que las GPUs se dedican al procesamiento gráfico, es necesario un sistema gráfico de sintetización de imágenes que describa escenas rápidamente—en los videojuegos esta velocidad llega a ser de alrededor de 60 veces por segundo—. Los diseñadores de GPUs dividen esta sintetización en varias etapas especializadas.

Primeramente se necesita saber que la GPU procesa de manera independiente (en paralelo) la información de los objetos que se desea mostrar en la pantalla. La geometría de la imagen es procesada con una serie de pasos que se describen a continuación, mientras que los datos de los pixeles siguen un proceso diferente; la secuencia con la que se siguen estos procesos es fija.

Procesamiento de vértices

En la primera etapa se considera que toda imagen esta constituida a base de triángulos, y es por ello que se divide cualquier forma compleja en triángulos. Más tarde se llevan los vértices de éstos triángulos a la tubería gráfica uno a uno con ayuda de una biblioteca gráfica. La GPU convierte a las figuras formadas los vértices en colecciones de triángulos cuando se necesitan.

Es necesario notar que en un inicio el desarrollo de las GPUs era a través del lenguaje ensamblador, lo cual era demasiado tedioso y difícil.

En la segunda etapa la GPU transforma cada objeto a un sistema de coordenadas común, esto es debido a que es posible definir cada objeto en un sistema de coordenadas propio; esta transformación es llamada **transformación de modelado**. El hecho de que cada objeto pueda ser definido de esta manera es muy conveniente para objetos que poseen una naturaleza jerárquica; un ejemplo de una jerarquía de sistemas coordenados puede ser un brazo robótico, el cual puede dividirse en: brazo, antebrazo, mano y dedos. Así el crear un brazo robótico puede hacerse con base en objetos semejantes pero con diferente tamaño y posición; con esto, puede definirse el brazo en el origen y para crear el antebrazo sería suficiente mover el origen y análogamente se podrán construir la mano y los dedos, todo esto sin que se pierda el origen inicial como se aprecia en la figura 1.3. Esta transformación se limita a operaciones como rotaciones, translaciones, escalamientos, etc., y de ésta manera se asegura de que los triángulos no se deformen o retuerzan. La representación de cada vértice con estas coordenadas hace que el sistema gráfico pueda ejecutar todas las transformaciones simultáneamente con un único producto entre una matriz y un vector que representan la transformación y la posición del vértice, respectivamente. Así, al salir de esta etapa se obtiene un flujo de triángulos expresados en un sistema coordenado común donde el observador se sitúa en el origen y la dirección en la que se observa se alinea con el eje Z.

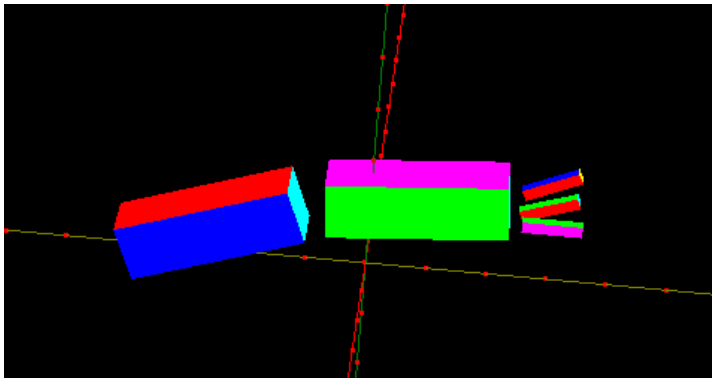

Figura 1.3: Brazo robótico modelado con ayuda de más de un sistema coordenado.

Después de llevar a todos los objetos a un sistema de coordenadas común se calcula su color considerando la luz que pudiera haber en la escena, ya que puede haber diversas fuentes de luz y por ello se agrega la contribución de cada luz individualmente. Existen muchas maneras de calcular la luz en una escena; entre ellas se encuentra el método de Phong, el cual considera la difusión del color, la normal a la superficie, el vector de la luz, el vector de luz reflejado, el vector de la cámara, etc.

Es necesario remarcar el hecho de que en donde se calcula el color es en los vértices de los objetos, ya que en como se menciono anteriormente todo objeto se divide en triángulos para luego almacenar los vértices de todos estos triángulos. Y como solamente conocemos los vértices de las figuras aun no podemos dar color al interior de la figura puesto que desconocemos los pixeles que se encuentran en el interior de la figura.

En la siguiente etapa se define un **cubo de visibilidad**, la cual será la región donde la escena tridimensional podrá ser apreciada y se ejecuta la **transformación de proyección** la cual proyecta cada triángulo 3D sobre una película plana de la cámara virtual, haciendo que los objetos puedan apreciarse en 2D con diferentes proyecciones tales como la proyección ortogonal y la proyección en perspectiva que se pueden apreciar en las figuras 1.4 y 1.5.

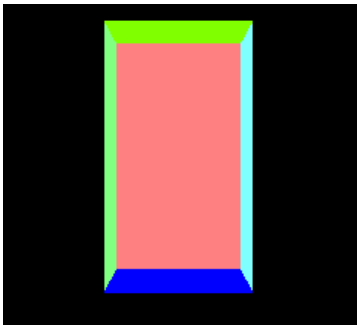

Figura 1.4: Ejemplo de proyección en perspectiva.

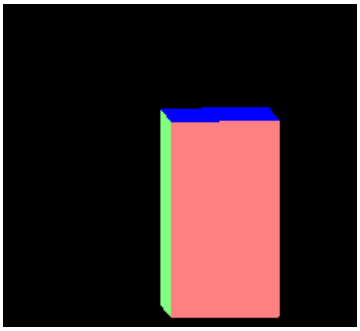

Figura 1.5: Ejemplo de proyeccion ortogonal.

Al igual que en la transformación de modelado, esta transformación se realiza con un simple producto de una matriz y un vector. El resultado de esta operación es un flujo de triángulos en coordenadas de la pantalla.

Después de esta transformación se halla el recorte de los objetos que llegan a salir total o parcialmente del cubo de visibilidad, los objetos cortados pueden agregar vértices

dependiendo de como es que se cortó el objeto. En algunos casos esto es seguido por un reacomodo de la perspectiva, lo cual hace que los objetos más cercanos se vean más grandes.

Rasterización

Al terminar los procesos aplicados a los vértices se determinan los pixeles del frame-buffer que estarán cubiertos por cada objeto visible en la pantalla, a la manera de determinar éstos pixeles se le llama **Rasterización**; a cada pixel cubierto por un objeto de la pantalla se le llama **Fragmento**, un ejemplo de esto es cuando un cuadrado cubre nueve pixeles, la rasterización convierte el cuadrado en nueve fragmentos. Ésta explicación se aprecia mejor en la figura 1.6.

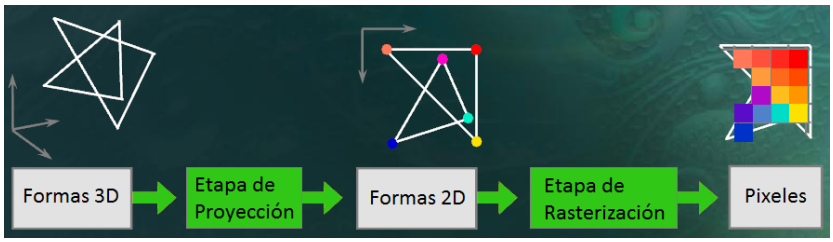

Figura 1.6: Proceso de Rasterización o Rasterization.

Procesamiento de pixeles

Una vez que se conocen los fragmentos que constituyen el objeto, se realizan operaciones sobre estos fragmentos, tales operaciones son, en algunos casos:

Mapeo de texturas Esta operación no siempre se ejecuta, ya que el mapeo de texturas consiste en "pegarüna imagen sobre un objeto con el fin de darle mayor realismo a dicho objeto

Niebla Esta operación ha sido explotada particularmente por videojuegos y es una función que se encuentra por default en muchas GPUs, las cuales brindan realismo a muchas escenas.

Suma de colores

Sin embargo existen otras operaciones las cuales son realizadas para todo tipo de objetos, entre estas operaciones esta el coloreado de cada fragmento lo cual se hace a través de la interpolación de los colores de los vértices.

Una consecuencia importante de éstas transformaciones es la independencia de los pixeles por ello la máquina puede manejar todos los pixeles en paralelo, lo cual ha llevado a los diseñadores de GPUs a un incremento en el conjunto de tuberías en paralelo.

En la actualidad las GPUs almacenan en una región de la memoria la distancia entre cada pixel y el observador, donde antes de mostrar algo en la pantalla la GPU compara la distancia entre los pixeles y los pixeles que ya se encuentran en la pantalla, la memoria sólo se llega a actualizar cuando los nuevos pixeles están cerca.

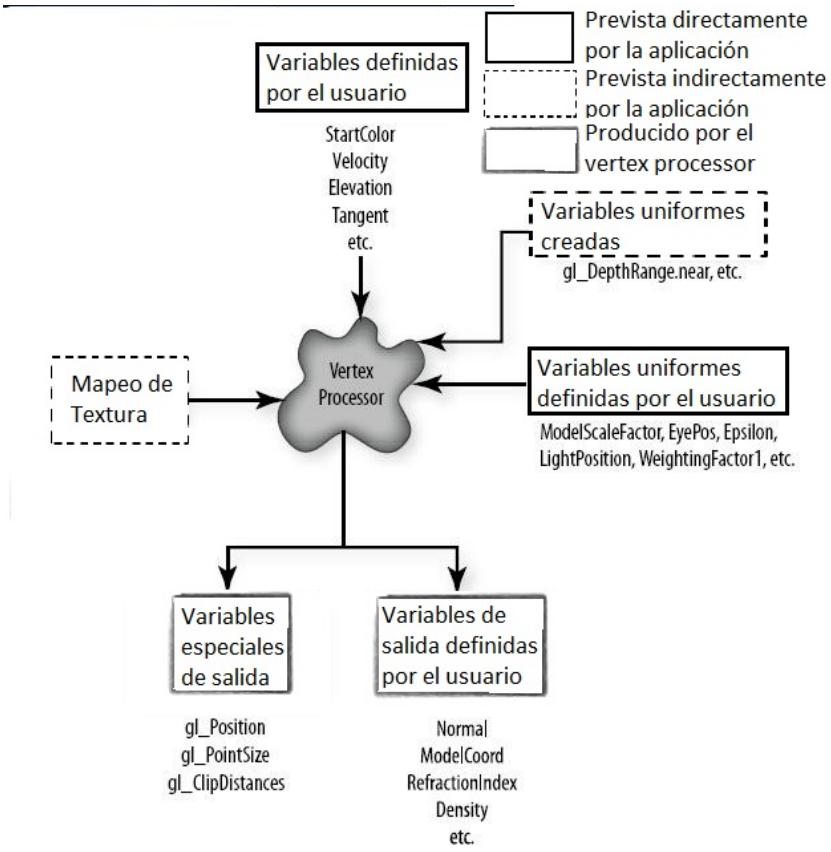

Figura 1.7: Esquema del procesamiento de vértices.

I.2.2 Las GPUs de la actualidad

La estructura que se ha descrito anteriormente era bastante buena para realizar diversos programas que utilizaban una gran cantidad de cálculos y producían muy buenos gráficos, sin embargo tenían un pequeño defecto. Debido a que se separaban los procesos realizados a los pixeles y a los vértices era posible saturar éstos procesos; un ejemplo de esto es cuando un programa despliega una imagen donde se aprecie la aurora boreal, los procesos aplicados a los vértices no son muchos sin embargo los procesos aplicados a los pixeles son demasiados ya que los colores en la aurora boreal cambian constantemente y para apreciar el realismo es necesario una gran cantidad de detalles en la imagen. Por otro lado si se desea hacer una animación de una persecución, es necesario una gran cantidad de objetos con el fin de que se pueda apreciar la velocidad entre otros detalles característicos en una persecución y por ello el procesamiento de vértices es muy importante, aunque los objetos no necesariamente requerirán gran atención a los pixeles.

Éste tipo de problemas llevaron a una unificación en los procesos, tal y como se muestra en la figura 1.9. Es decir, en lugar de manejar de manera independiente el proceso de pixeles y el proceso de vértices, éstos procesos se anteponen uno de otro para poder evitar saturaciones en los procesos como los que se acaban de describir. Más precisamente, el primer proceso en ejecutarse es el procesamiento de vértices para luego continuar con la rasterización y concluyendo con el procesamiento de fragmentos para luego desplegar los datos

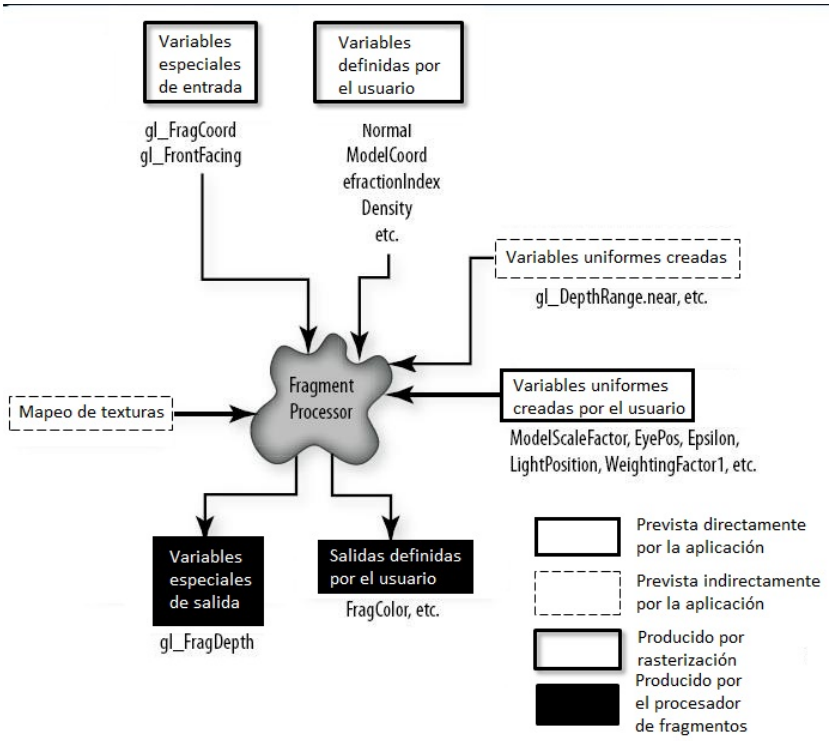

Figura 1.8: Esquema del procesamiento de pixeles.

modificados y obtenidos en el buffer. Esto ayuda a que por un lado se incremente la capacidad de procesamiento de información debido a la unificación de ambos procesos y al mismo tiempo permite que cualquier proceso que utilice más recursos, los tenga a su disposición.

Desde la primera GPU, la iSBX 275 creada en 1983 por Intel hasta las actuales, las GPUs han evolucionado de una tubería gráfica 3D para funciones fijas a un motor computacional de propósito general. Los **shaders** son programas de computadora que corren sobre la GPU, estos de manera automática, se dedican a producir niveles apropiados de luz y oscuridad sobre una imagen además de producir otros muchos tipos de efectos sobre las formas e imagenes que se trabajan. Los shaders, por si fuera poco pueden ejecutar diversos cálculos sobre los pixeles de manera independiente, cálculos como productos de matrices, exponentes, etc., los cuales han sido utilizados para crear gráficos cada vez más complejos y con mayor realismo. La capacidad de programar shaders fue posible en el 2004 lo cual nos habla de lo joven de ésta capacidad; éstos shaders son destinados para alterar las operaciones que se realizan en cada uno de los procesos y programarlos al gusto, los pasos descritos en el procesamiento de vértices y procesamiento de fragmento pueden ser hecho a través del **vertex shader** y del **fragment shader** respectivamente. Para más información acerca de los shaders vease 2.2.4.

El rápido incremento en la potencia y precisión de las GPUs orillo a programadores y científicos a investigar como aprovechar las capacidades de cómputo de las GPUs, capacidades tales como cálculos rápidos; a la divulgación de ésta información se le conoce como **GPGPU**.

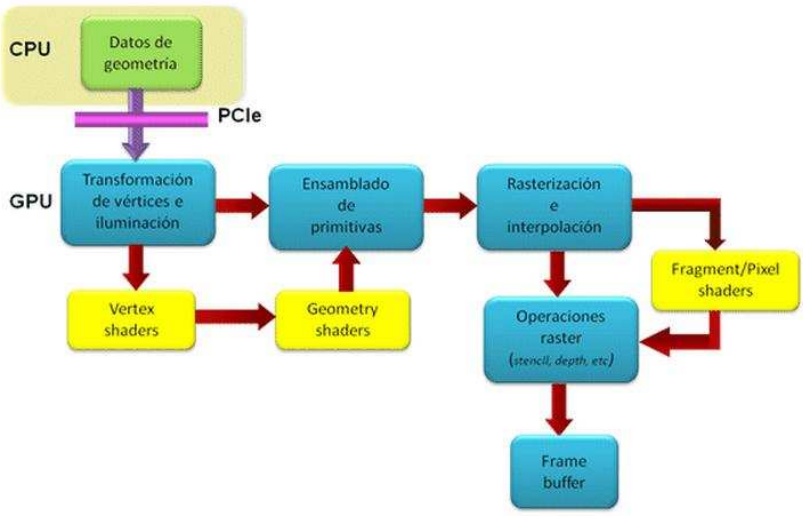

Figura 1.9: Esquema de la unificación de procesos.

La GPGPU ayudó a que cualquier programador pudiera aprovechar todos los beneficios que las GPUs pueden aportar, sin embargo ahora existen muchas maneras de programarlas. éstas maneras se explicaran en el siguiente capitulo.

Es necesario mencionar que desde el 2010 se ha incorporado un nuevo proceso, éste afecta a la geometría del objeto el cual es llamado **Geometry Shader**. El geometry shader es un proceso que entra en acción cuando el programador crea dicho shader y es utilizado para crear o eliminar vértices o primitivas, por lo que es ejecutado después del vertex shader y antes del rastreador.

II. ESTADO DEL ARTE

II.1. Programando una GPU

Como se acaba de mencionar, existen muchas maneras de programar una GPU, estas maneras son a través de diversas bibliotecas apoyadas en diferentes lenguajes que aportan múltiples herramientas que facilitan la interacción entre el programador, el cual manda ordenes a través de la CPU, y la GPU. Algunos de estos lenguajes son GLSL, CUDA, Cg, HLSL, DirectX, OpenCL y Close to Metal.

II.1.1 Lenguajes para GPUs

Debido a que hay una gran variedad de lenguajes, y puesto que el presente trabajo está basado en el lenguaje GLSL; con el fin de apreciar los beneficios que posee el lenguaje que se utilizó para programar la biblioteca, a continuación se describirán de forma breve algunas características y rasgos que contiene cada uno de los lenguajes.

Cg

Cg es la abreviación de C para gráficos (C for graphics), es un lenguaje de nivel alto creado por Nvidia en colaboración con Microsoft para la programación de vertex shaders y fragment shaders, es decir, es un lenguaje diseñado para la interacción directa y programación de la GPU. Cg está basado en el lenguaje de programación C, pero aunque comparten la misma sintaxis, Cg contiene algunas características que fueron modificadas, lo cual lo hace parcialmente diferente a C, una de estas características es la incorporación de tipos de datos para hacer más cómoda la programación en la GPU, algunos de estos tipos de datos son:

- float - representa un número de punto flotante de 32 bit
- half - representa un número de punto flotante de 16 bit
- int - representa un número entero de 32 bit
- bool - representa una variable booleana
- sampler - representa un objeto textura, etc.

Nota: Debido a que éste lenguaje fue creado por Nvidia, es necesario detallar que los programas aplicables a la GPU, son exclusivos para las tarjetas gráficas diseñadas por Nvidia y en un sistema operativo de Windows. Y que no es un lenguaje diseñado para el propósito general.

CTM

CTM es la abreviación de Close to Metal. CTM es una biblioteca creada por AMD para propósito general, el cual sin embargo sólo opera en las GPU de AMD, es decir, ATI.

A pesar de que ésta biblioteca fue presentada en el 2006, fue en el 2008 que se terminó su desarrollo.

CUDA

CUDA al igual que Cg, fue diseñado por Nvidia y al igual que en Cg, los programas diseñados para implementarse en la GPU sólo pueden ser implementados en GPUs de Nvidia, en particular de la serie G8X en adelante. La primera versión de éste lenguaje, el cual salió en el 2007 funcionaba en Windows y Linux; fue hasta su versión 2.0 que también funcionó en Mac OS.

Un beneficio adicional a CUDA es el hecho de usa los beneficios de la GPU para propósito general, es decir, es de género GPGPU.

HLSL

Abreviación que significa High Level Shader Language, HLSL es un lenguaje desarrollado por Microsoft para uso en las aplicaciones de Direct3D. HLSL es un lenguaje semejante a Cg ya que fue desarrollado junto a éste.

GLSL

GLSL significa OpenGL Shading Language, éste lenguaje de alto nivel está basado en la sintaxis del lenguaje de programación C además de que soporta bucles y bifurcaciones incluyendo: if-else, for, do-while, break, continue, etc. GLSL fue creado por OpenGL[1] para tener un control directo de la tubería de gráficos, es decir, la GPU.

Algunos destacados beneficios de GLSL, algunos de los cuales son heredados por OpenGL se presentan a continuación:

- La compatibilidad multiplataforma en los sistemas operativos: GNU/Linux, Mac OS X y Windows
- Los programas empleados para manipular GPUs pueden ser usados en cualquier tarjeta gráfica, como Nvidia y ATI
- El compilador GLSL se encuentra instalado en cualquier hardware
- Los lenguajes de programación con los cuales se puede utilizar GLSL además de C son: C++, C#, Delphi, Java, etc.

OpenCL

OpenCL significa Open Computing Language, además de ser un lenguaje de programación también posee una interfaz de programación de aplicaciones. Una de las características bastante impresionantes de OpenCL es que permite crear aplicaciones que pueden

[1] OpenGL es un estándar creado en 1992 por Silicon Graphics Inc. En la siguiente subsección se hablará con más detalle acerca de dicho estándar

ejecutarse tanto en la unidad de procesamiento central como en la GPU; el lenguaje usado en OpenCL está basado en C99.

El creador de OpenCL es Apple y el grupo Khronos fue quien lo convirtió en un estándar abierto y libre; en la actualidad OpenCL es apoyado por AMD en lugar del lenguaje desarrollado por ellos mismos, CTM.

Debido a que el objetivo de ésta tesis además de desarrollar una biblioteca con funciones matematicas útilies es la aceptación por parte de los usuarios para la utilización de ésta misma biblioteca, se decidió desarrollar la misma con ayuda del lenguaje GLSL, ya que éste los programas desarrollados por éste lenguaje puede ser utilizados en diferentes sistemas operativos y sin dependencia a algúna marca de GPU.

Además de que la sintaxis que utiliza esta basado en el lenguaje de programación C, el cual es bastante conocido y dominado por los autores de éste trabajo lo cual aporto un beneficio adicional bastante agradable.

Ahora que se ha decidido el lenguaje que se utilizará para operar sobre la GPU, comenzaremos a introducir al lector en éste lenguaje para darle las herramientas iniciales para, en un futuro, realizar algunas funciones que la GPU pueda interpretar y realizar.

Como se ha dicho antes, GLSL fue creado por OpenGL y por ello se hablará un poco sobre ésta biblioteca.

II.2. OpenGL

OpenGL significa Open Graphics Library y es una especificación estandar multiplataforma para producir aplicaciones 2D y 3D; en escencia OpenGL trata de cumplir los siguientes propositos:

- Ocultar las capacidades de las diferentes plataformas, con el único requisito de que las implementaciones soporten la funcionalidad de OpenGL
- Abstraer la complejidad de las diversas interfacez que contienen las tarjetas gráficas

OpenGL contiene primitivas para puntos, líneas, polígonos, etc., primitivas que son procesadas a través de los datos introducidos, los vértices, y que luego son convertidos en pixeles. Los comandos de OpenGL generalmente se limitaban a emitir primitvas o específicar como se procesarían dichas primitivas; éste proceso de introducir vértices o especificar la manera en como se procesarían las primitivas se ejecutaba de manera prefijada, lo que hacía que fuese poco manipulable. ésto dejó de se así en el año 2004, cuando se publicó la versión 2.0 cuando se introdujo la capacidad de configurar la manera en como se procesaban lós vértices y cómo se moldeaban las primitivas, ésto claro en algunas etapas.

A continuación presentamos el ejemplo por default con que se comienza a programar en OpenGL.

```
1
2   #include <windows.h>
3   #include <gl/gl.h>
4   #include <stdio.h>
5
6   LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);
7   void EnableOpenGL(HWND hwnd, HDC*, HGLRC*);
8   void DisableOpenGL(HWND, HDC, HGLRC);
9
10
11  int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow){
12      WNDCLASSEX wcex;
13      HWND hwnd;
```

```
14      HDC hDC;
15      HGLRC hRC;
16      MSG msg;
17      BOOL bQuit = FALSE;
18      float theta = 0.0f;
19
20      /* register window class */
21      wcex.cbSize = sizeof(WNDCLASSEX);
22      wcex.style = CS_OWNDC;
23      wcex.lpfnWndProc = WindowProc;
24      wcex.cbClsExtra = 0;
25      wcex.cbWndExtra = 0;
26      wcex.hInstance = hInstance;
27      wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
28      wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
29      wcex.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
30      wcex.lpszMenuName = NULL;
31      wcex.lpszClassName = "GLSample";
32      wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);;
33
34
35      if (!RegisterClassEx(&wcex))
36          return 0;
37
38      /* create main window */
39      hwnd = CreateWindowEx(0, "GLSample", "OpenGL Sample", WS_OVERLAPPEDWINDOW,
40                            CW_USEDEFAULT, CW_USEDEFAULT, 256, 256,
41                            NULL, NULL, hInstance, NULL);
42
43      ShowWindow(hwnd, nCmdShow);
44
45      /* enable OpenGL for the window */
46      EnableOpenGL(hwnd, &hDC, &hRC);
47
48      /* program main loop */
49      while (!bQuit)
50      {
51          /* check for messages */
52          if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
53          {
54              /* handle or dispatch messages */
55              if (msg.message == WM_QUIT)
56              {
57                  bQuit = TRUE;
58              }
59              else
60              {
61                  TranslateMessage(&msg);
62                  DispatchMessage(&msg);
63              }
64          }
65          else
66          {
67              /* OpenGL animation code goes here */
68
69              glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
70              glClear(GL_COLOR_BUFFER_BIT);
71
72              glPushMatrix();
73              glRotatef(theta, 0.0f, 0.0f, 1.0f);
74
75              glBegin(GL_TRIANGLES);
76
77                  glColor3f(1.0f, 0.0f, 0.0f);   glVertex2f(0.0f,   1.0f);
78                  glColor3f(0.0f, 1.0f, 0.0f);   glVertex2f(0.87f,  -0.5f);
79                  glColor3f(0.0f, 0.0f, 1.0f);   glVertex2f(-0.87f, -0.5f);
80
81              glEnd();
82
83              glPopMatrix();
84
85              SwapBuffers(hDC);
86
87              theta += 1.0f;
88              Sleep (1);
89          }
90      }
91      /* shutdown OpenGL */
92      DisableOpenGL(hwnd, hDC, hRC);
93
94      /* destroy the window explicitly */
95      DestroyWindow(hwnd);
96
97      return msg.wParam;
98  }
99  LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam){
100     switch (uMsg)
101     {
102         case WM_CLOSE:
103             PostQuitMessage(0);
104         break;
105
```

```
106             case WM_DESTROY:
107                 return 0;
108
109             case WM_KEYDOWN:
110             {
111                 switch (wParam)
112                 {
113                     case VK_ESCAPE:
114                         PostQuitMessage(0);
115                     break;
116                 }
117             }
118             break;
119
120             default:
121                 return DefWindowProc(hwnd, uMsg, wParam, lParam);
122         }
123
124         return 0;
125     }
126     void EnableOpenGL(HWND hwnd, HDC* hDC, HGLRC* hRC){
127         PIXELFORMATDESCRIPTOR pfd;
128
129         int iFormat;
130
131         /* get the device context (DC) */
132         *hDC = GetDC(hwnd);
133
134         /* set the pixel format for the DC */
135         ZeroMemory(&pfd, sizeof(pfd));
136
137         pfd.nSize = sizeof(pfd);
138         pfd.nVersion = 1;
139         pfd.dwFlags = PFD_DRAW_TO_WINDOW |
140                       PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
141         pfd.iPixelType = PFD_TYPE_RGBA;
142         pfd.cColorBits = 24;
143         pfd.cDepthBits = 16;
144         pfd.iLayerType = PFD_MAIN_PLANE;
145
146         iFormat = ChoosePixelFormat(*hDC, &pfd);
147
148         SetPixelFormat(*hDC, iFormat, &pfd);
149
150         /* create and enable the render context (RC) */
151         *hRC = wglCreateContext(*hDC);
152
153         wglMakeCurrent(*hDC, *hRC);
154     }
155     void DisableOpenGL (HWND hwnd, HDC hDC, HGLRC hRC){
156         wglMakeCurrent(NULL, NULL);
157         wglDeleteContext(hRC);
158         ReleaseDC(hwnd, hDC);
159     }
```

Figura 2.1: Ejemplo de un programa en OpenGL.

A pesar de lo extenso que pudiera parecer el ejemplo anterior, es en las líneas 67 - 88 donde se encuentra el código relevante para OpenGL, los cuales son validados por la biblioteca gl/gl.h la cual es referenciada en la línea 3; por lo demás debemos decir que las demás líneas de código son necesarias pues el algoritmo se está realizando para plataforma windows, por ello es que se utiliza la biblioteca windows.h, la cual se encuentra en la línea 2.

El algoritmo anterior ejecuta un triángulo cuyos vértices tienen los colores: rojo, verde y azul como se muestra en la figura (2.2) , y el cual rota constantemente hasta que el programa finalize.

Las capacidades introducidas por la versión 2.0 utilizaban el lenguaje de programación llamado **GLSL**.

Ahora bien, antes de continuar con ésta desarrollo que nos llevará a la manera en cómo se programa la GPU con el lenguaje GLSL hablaremos de un tipo de dato utilizado para hacer gráficos más realistas contenido en la biblioteca de OpenGL, es decir LAS **Texturas**. Aunque pareciera que ésta subsección carece de sentido para conocer la manera de programar

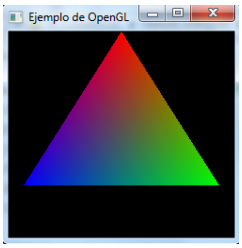

Figura 2.2: Triángulo ejecutado por el ejemplo del programa de OpenGL.

una GPU, ciertamente es de vital importancia abordarlo ya que el medio de comunicación, es decir, la manera de enviar información de la CPU a la GPU y viceversa es através de texturas.

II.2.1 Texturas

Las texturas permiten pegar una imagen sobre polígonos y de ésta manera hacer que el gráfico parecesca más reliasta luciendo por ejemplo mármol, madera, tela, etc.; además de que el mapeo de textura asegura que las imagenes sufrirán las mismas transformaciones de los polígonos a los que se encuentran pegados.

A pesar de ésta introducción las texturas son simples arreglos rectangulares de datos. Cada valor en un arreglo textura es llamado **texel** y aunque es común pensar que una textura es un arreglo bidimensional, una textura también puede ser unidimensional o tridimensional.

Nota: Desde el punto de vista gráfico, la belleza de las texturas se encuentra en que a pesar de ser arreglos rectangulares, las texturas pueden incrustarse en objetos no rectangulares como lo muestra la figura (2.3)

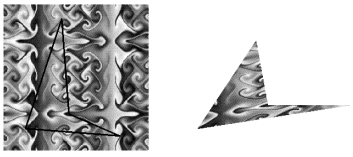

Figura 2.3: Ejemplo de una textura incrustada en una forma no rectangular.

Para definir una textura bidimensional es necesario utilizar el siguiente comando:

void glTexImage2D(GLenum *target*, GLint *level*, GLint *internalFormat*, GLsizei *width*, GLsizei *height*, GLint *border*, GLenum *format*, GLenum *type*, const GLvoid **texels*);
El comando anterior define una textura bidimensional. El parámetro *target* es fijado por una de las siguientes constantes: GL_TEXTURE_2D, GL_PROXY_TEXTURE_2D, GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z o GL_PROXY_TEXTURE_CUBE_MAP para definir texturas bidimensionales y GL_TEXTURE_1D_ARRAY y GL_PROXY_TEXTURE_1D_ARRAY para definir un arreglo de textura unidimensional, o GL_TEXTURE_RECTANGLE y GL_PROXY_TEXTURE_RECTANGLE.
El parámetro *level* es utilizado si se proporcionan múltiples resoluciones al mapeo de la textura, si se utiliza sólo una resolución *level* es 0.
El parámetro *internalFormat* los componentes de los texeles de la imagen que se seleccionaran (éstos pueden ser RGBA, profundidad, luminancia o intencidad). Existen tres grupos de formatos internos; las constantes simbólicas del primer grupo especifican los valores de los texeles que deben ser normalizados al rango $[0, 1]$ y almacenados en una representación de punto fijo, es decir, el número de bits necesarios para cada valor: GL_ALPHA, GL_ALPHA4, GL_ALPHA8, GL_ALPHA12, GL_ALPHA16, GL_COMPRESSED_ALPHA, GL_COMPRESSED_LUMINANCE, GL_COMPRESSED_LUMINANCE_ALPHA, GL_COMPRESSED_INTENSITY, GL_COMPRESSED_RGB, GL_COMPRESSED_RGBA, GL_DEPTH_COMPONENT, GL_DEPTH_COMPONENT16, GL_DEPTH_COMPONENT24, GL_DEPTH_COMPONENT32, GL_DEPTH_STENCIL, GL_INTENSITY, GL_INTENSITY4, GL_INTENSITY8, GL_INTENSITY12, GL_INTENSITY16, GL_LUMINANCE, GL_LUMINANCE4, GL_LUMINANCE8, GL_LUMINANCE12, GL_LUMINANCE16, GL_LUMINANCE_ALPHA, GL_LUMINANCE4_ALPHA4, GL_LUMINANCE6_ALPHA2, GL_LUMINANCE8_ALPHA8, GL_LUMINANCE12_ALPHA4, GL_LUMINANCE12_ALPHA12, GL_LUMINANCE16_ALPHA16, GL_RED, GL_R8, GL_R16, GL_RG, GL_RG8, GL_RG16, GL_RGB, GL_R3_G3_B2, GL_RGB4, GL_RGB5, GL_RGB8, GL_RGB10, GL_RGB12, GL_RGB16, GL_RGBA, GL_RGBA2, GL_RGBA4, GL_RGB5_A1, GL_RGBA8, GL_RGB10_A2, GL_RGBA12, GL_RGBA16, GL_SRGB, GL_SRGB8, GL_SRGB_ALPHA, GL_SRGB8_ALPHA8, GL_SLUMINANCE_ALPHA, GL_SLUMINANCE8_ALPHA8, GL_SLUMINANCE, GL_SLUMINANCE8, GL_COMPRESSED_SRGB, GL_COMPRESSED_SRGB_ALPHA, GL_COMPRESSED_SLUMINANCE, or GL_COMPRESSED_SLUMINANCE_ALPHA.

Un **objeto textura** es aquel que almacena los datos de la textura y hace que este disponible con mayor facilidad. La manera de llenar un objeto textura es cargando una imagen en ésta, ésto si se desea usar la textura para gráficos, o darle valores por nuestra cuenta a cada texel, que es lo que se desea; los datos que describen una textura pueden consistir de uno,

El segundo grupo de constantes simbólicas fueron agregadas en el 2008 y especifican formatos de pixeles de punto flotante, los cuales no necesitan ser normalizados: GL_R16F, GL_R32F, GL_RG16F, GL_RG32F, GL_RGB16F, GL_RGB32F, GL_RGBA16F, GL_RGBA32F, GL_R11F_G11F_B10F y GL_RGB9_E5.
En el 2009 se agregó el soporte para valores con signo normalizados, es decir los que son mapeados al rango$[-1, 1]$ y son especificos para el *internalFormat* que posean las siguientes muestras: GL_R8_SNORM, GL_R16_SNORM, GL_RG8_SNORM, GL_RG16_SNORM, GL_RGB8_SNORM, GL_RGB16_SNORM, GL_RGBA8_SNORM, GL_RGBA16_SNORM.
Los parámetros *width* y *height* nos indican las dimensiones de la textura imagen; el parámetro *border* especifica el ancho del borde, el cual debe ser 0 si no hay borde o 1 en caso contrario.
Nota: Cualquier GPU anterior al 2004 debe expresar a los parámetros *width* y *height* para que tengan la forma $2^m + 2b$, con m un entero no negativo y b debe ser el valor del borde.
Los parámetros *format* y *type* describen el formato y tipo de dato de la imagen textura respectivamente. Los valores del parámetro *format* pueden ser: GL_COLOR_INDEX, GL_DEPTH_COMPONENT, GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_LUMINANCE o GL_LUMINANCE_ALPHA; desde el 2009 también se pueden agregar los siguientes valores: GL_DEPTH_STENCIL, GL_RG, GL_RED_INTEGER, GL_GREEN_INTEGER, GL_BLUE_INTEGER, GL_ALPHA_INTEGER, GL_RG_INTEGER, GL_RGB_INTEGER, GL_RGBA_INTEGER, GL_BGR_INTEGER, and GL_BGRA_INTEGER.
El parámetro *type* puede tener los valores: GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT o GL_BITMAP.
Y por último, el parámetro *texels* contiene los datos de la textura imagen

Figura 2.4: Función glTexSubImage2D.

dos, tres o cuatro elementos por texel y generalmente se representa por la cuatro upla (R, G, B, A).

Para darle valores a un objeto textura es necesario seguir los siguientes pasos:

- Generar el nombre de la textura
- Enlazar el objeto textura con los datos de la textura
- Enlazar los objetos textura repetidamente para que los datos de cada textura esten continuamente disponibles

Para generar el nombre del objeto textura y crear el mismo para enlazarlo con los datos de la textura se usan los siguientes comandos:

void glGenTextures(GLsizei *n*, GLuint **textureNames*);
El comando anterior regresa *n* nombres sin usar para los objetos textura en el arreglo **textureNames*

void glBindTexture(GLenum *target*, GLuint *textureNames*);
Cuando enlazamos a un objeto textura ya creado, el objeto textura se activa. El parámetro *target* puede tener los valores GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D, GL_TEXTURE_CUBE_MAP, GL_TEXTURE_1D_ARRAY, GL_TEXTURE_2D_ARRAY, GL_TEXTURE_RECTANGLE, GL_TEXTURE_BUFFER

Una vez terminado de usar el objeto textura es necesario utilizar el siguiente comando, ya que los datos de los objetos texturas se quedan.

void glDeleteTextures(GLsizei *n*, const GLuint **textureNames*);
éste comando borra *n* objetos texturas, llamados por los elementos del arreglo *textureNames*

Cuando se implementa una textura sobre una escena, se deben de proporcionar las coordenadas del objeto y las coordenadas de la textura para cada vértice. Las coordenadas de la textura determinan cual texel en el mapeo de textura es asignado a cual vértice. Las coordenadas de la textura pueden comprimirse en uno, dos, tres o cuatro coordenadas. referidos como coordenadas *s, t, r, y q* generalmente. El comando para especificar las coordenadas de la textura es el siguiente:

void glTexCoord1234sifd(TYPE *coords*);
void glTexCoord1234sifdv(const TYPE **coords*);
Con ayuda de éste comando se fijan las coordenadas de la textura (s, t, r, q). Para luego especificar la coordenada en el entorno gráfico (en el entorno de OpenGL, ésto es con ayuda del comando glVertex*()) en cuyos vértices se asignará las coordenadas de la textura actual. Se usa el sufijo (s, i, f o d) apropiado que corresponde al valor de tipo GLshort, GLint, GLfloat o GLdouble, el cual depende del tipo de dato de las coordenadas en la parte gráfica. La diferencia entre el primer comando y el segundo es en como se especifiquen las coordenadas, las cuales pueden ser expresadas individualmente o expresadas en la versión vectorial

Cuando se llegan a usar texturas con bordes o tienen especificado un color en el borde de la textura es necesario especificar el modo de envoltura (wrapping mode) y el método de filtrado, ya que éstos influencian el cómo y el dónde se usará la información del borde. Por ello abordaremos éstos temas enseguida.

Métodos de filtrado

Uno de éstos métodos, el cual tiene una gran importancia para nosotros por la manera en que se usará, es el método GL_NEAREST, el cual toma el texel más cercano para usarlo.

Otro método es el G_LINEAR, el cual usa una combinación pesada (weighted) en un arreglo de 2x2(para el caso de texturas bidimensionales) de datos de colores es usado para aplicarlo en la textura.

Modo de envoltura

El modo de envoltura GL_CLAMP_TO_BORDER usa sólo los texeles del borde para la aplicación de la textura si las texturas coordenadas estan fuera del rango $[0, 1]$ (si no hay bordes usa un color de borde constante); si además está combinado con el método de filtrado GL_NEAREST entonces escoje los texeles del borde más cercano.

El modo GL_CLAMP_TO_EDGE al igual que el modo GL_REPEAT ignora el borde, sin embargo en el caso del modo GL_CLAMP_TO_EDGE los texeles del borde(edge) o cercanos a éste si son usados para los cálculos de la texturización pero no los del borde(border). Mientras que en el modo GL_REPEAT los texeles de los bordes del arreglo de 2 x 2 son mezclados, es decir, los texeles del borde derecho son promediados con los del borde izquierdo al igual que los texeles del borde superior son promediados con los del borde inferior.

Para el modo GL_CLAMP al igual que el modo GL_TEXTURE_BORDER_COLOR, los texeles del borde son usados en un arreglo de 2 x 2 de texeles pesados(weighted).

La manera en como se usan éstos parámetros es con el siguiente parámetro:

void glTexParameterif(GLenum *target*, GLenum *pname*, TYPE *param*);
void glTexParameterifv(GLenum *target*, GLenum *pname*, const TYPE **param*);
void glTexParameterIi uiv(GLenum *target*, GLenum *pname*, const TYPE **param*);
Como ya se dijo previamente, éstos comandos especifican los parámetros de control que determinarán la manera en como la textura será tratada y aplicada a un fragmento o almacenada en un objeto textura. El parámetro *target* puede ser GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D, GL_TEXTURE_1D_ARRAY, GL_TEXTURE_2D_ARRAY, GL_TEXTURE_CUBE_MAP o GL_TEXTURE_RECTANGLE para que enccaje con la textura pensada.

Figura 2.5: Función glTexParameter.

Los valores respectivos para los parámetros *pnam* y *param* se muestran en la siguiente tabla.

Puesto que es posible utilizar más de una textura, es necesario saber como asignar información a cada una da las unidades de texturas, el siguiente comando selecciona la actual unidad de textura para ser modificada.

Cualquier comando que sea llamado despues de **glActiveTexture()**, tal como **glBindTexture()**, afectará sólo a la unidad de textura actual.

Cuando se usen objetos textura, podremos enlazar una textura a una unidad de textura. De ésta forma la unidad de textura tendrá los calores del estado de la textura contenida dentro del objeto textura

Parámetro *pname*	Valores para *param*
GL_TEXTURE_WRAP_S	GL_CLAMP, GL_CLAMP_TO_EDGE, GL_CLAMP_TO_BORDER, GL_REPEAT, GL_MIRRORED_REPEAT
GL_TEXTURE_WRAP_T	GL_CLAMP, GL_CLAMP_TO_EDGE, GL_CLAMP_TO_BORDER, GL_REPEAT, GL_MIRRORED_REPEAT
GL_TEXTURE_WRAP_R	GL_CLAMP, GL_CLAMP_TO_EDGE, GL_CLAMP_TO_BORDER, GL_REPEAT, GL_MIRRORED_REPEAT
GL_TEXTURE_MAG_FILTER	GL_NEAREST, GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_BORDER_COLOR	cualquier valor en $[0.0, 1.0]$ o valores enteros con o sin signo
GL_TEXTURE_PRIORITY	$[0.0, 1.0]$ para el objeto textura actual
GL_TEXTURE_MIN_LOD	cualquier valor de punto flotante
GL_TEXTURE_MAX_LOD	cualquier valor de punto flotante
GL_TEXTURE_BASE_LEVEL	cualquier entero no negativo
GL_TEXTURE_MAX_LEVEL	cualquier entero no negativo
GL_TEXTURE_LOD_BIAS	cualquier entero no negativo
GL_DEPTH_TEXTURE_MODE	GL_RED, GL_LUMINANCE, GL_INTENSITY, GL_ALPHA
GL_TEXTURE_COMPARE_MODE	GL_NONE, GL_COMPARE_REF_TO_TEXTURE (que soporten la versión 3.0 en adelante), o GL_COMPARE_R_TO_TEXTURE (para versiones que soporten 2.1 en adelante incluyendo 2.1)
GL_TEXTURE_COMPARE_FUNC	GL_LEQUAL, GL_GEQUAL, GL_LESS, GL_GREATER, GL_EQUAL, GL_NOTEQUAL, GL_ALWAYS, GL_NEVER
GL_GENERATE_MIPMAP	GL_TRUE, GL_FALSE

Figura 2.6: Tabla de opciones para los parámetros *pnam* y *param*.

void glActiveTexture(GLenum *texUnit*);
Selecciona la unidad de textura que está actualmente modificada por rutinas de texturización. El parámetro *texUnit* es una constante simbólica de la forma GL_TEXTURE*i*, donde *i* puede tomar el valor de 0 a $n-1$, donde n es el máximo número de unidades de textura.

Figura 2.7: Función glActiveTexture.

Para más información acerca de éstos comandos o más vease **?**

Ahora daremos terminada ésta parte teórica para continuar con la manera en como enviaremos información desde la CPU, el cual es el lugar donde introducimos los datos, a la GPU, el cual será el lugar donde se realizen los cálculos.

II.2.2 Envío de datos de la CPU a la GPU

Previo a ver éste tema abordaremos otro el cual es de vital importancia para continuar.

Objetos Framebuffer

Un framebuffer en escencia es una categoría de dispositivos gráficos, éstos representan a los diferentes pixeles de la pantalla como ubicaciones en la memoria de acceso aleatorio. La manera para asignar una aplicación por un objeto framebuffer es llamando al siguiente comando, el cual asignará un identificador no usado para el objeto framebuffer.

void glGenFramebuffers(GLsize *n*, GLuint **ids*);
Con éste comando se asignan *n* objetos framebuffer no usados y se retornan sus nombres en *ids*

Figura 2.8: Función glGenFramebuffers.

Es necesario remarcar que el asignar un nombre de un objetos framebuffer no crea el objeto framebuffer ni mucho menos almacena algo en éste; éstas tareas se manejan a través del siguiente comando.

void glBindFramebufferEXT(GLenum *target*, GLuint *framebuffer*);
Con éste comando se especifica si en el framebuffer se escribirán datos o se leerán. *framebuffer* especifica el destino framebuffer para la entrega de datos. Cuando *target* sea GL_FRAMEBUFFER se fijará al framebuffer de lectura y escritura para enlazarlo con *framebuffer*.

Figura 2.9: Función glBindFramebufferEXT.

Para marcar el nombre de algun objeto framebuffer como no usado y liberar alguna aplicación almacenada se usa el siguiente comando.

El tipo de dato base con el que trabajaremos será el arreglo ya que es el tipo de dato nativo en las CPUs, un **arreglo** es un grupo de ubicaciones de memoria consecutivas, las cuales poseen el mismo tipo. La manera de hacer referencia a una ubicación o elemento

void glDeleteFramebuffers(GLsize *n*, GLuint **ids*);
Gracias a éste comando se desalojan *n* objetos framebuffer que estén asociados a los nombres aportados en *ids*.

Figura 2.10: Función glDeleteFramebuffers.

particular del arreglo, es especificando el nombre y número de posición del elemento en el arreglo.

Para hacer referencia a algún elemento particular de un arreglo en un programa, se proporciona el nombre del arreglo seguido del número de la posición en la que se encuentra dicho elemento entre corchetes ([ ]). Al número de la posición se le conoce como **índice** o **subíndice**. El primer elemento de todo arreglo posee el subíndice 0.

Ejemplo, si identificamos algún arreglo con el nombre c, el cual posee 10 elementos y deseamos hacer referencia al 4^{to} elemento, sólo es necesario expresarlo como c[3], ya que como el primer elemento tiene el subíndice 0, el segundo elemento tiene el subíndice 1, el tercer elemento tiene el subíndice 2, entonces el 4^{to} elemento poseerá el subíndice 3.

Puesto que la dimensión de un arreglo puede variar, es necesario conocer la manera de como accesar a ellos a través de un arreglo unidimensional. Puesto que el explicar en detalle lo anterior no es el tema principal de éste trabajo, solo daremos un ejemplo.

Ejemplo, si poseemos un arreglo bidimensional llamado 'a' de dimensiones M y N, el cual puede expresarse como *a[i][j]*; la manera de accesar a los elementos del anterior arreglo sería a traves del arreglo unidimensional *a[M * i + j]*.

Por otro lado, el tipo de datos nativo en las GPUs es el arreglo bidimensional, éstos arreglos son llamadas **texturas** en la memoria de la GPU; aunque las dimensiones de las texturas son limitadas, éstas pueden superar los valores de 2048 ó 4096 por dimensión.

Nota: Cuando en la CPU nos referimos al índice del arreglo, en la GPU estaremos accesando a los valores de las coordenadas de la textura almacenadas en la textura.

Puesto que la manera de enviar datos de la CPU a la GPU varia un poco debido a la marca de la misma, se detallara el como enviar información para cada tipo de GPU.

La primera cosa en la que se tiene que ser cuidadoso es en la manera en como se mandara el arreglo bidimendional a la GPU, ya que ésto afectara a las dimensiones de la textura y por consecuencia a la manera en como se introduciran las coordenadas de la textura.

Las opciones que tenemos para solucionar ésto es con los siguientes parámetros:

GL_TEXTURE_2D - éste parámetro nos pide que las dimensiones de la textura sean potencia de dos; además de que nos pide que las coordenadas de la textura sean normalizadas al rango $[0, 1]$ por $[0, 1]$, independientemente de las dimensiones $[0, M]$ por $[0, N]$ de la textura.

GL_TEXTURE_RECTANGLE - éste parámetro nos permite que las dimensiones sean arbitrarias; además de que las dimensiones no necesitan ser normalizadas

En la siguiente cosa en la que hay que pensar, es en el **formato de la textura**. Las GPUs pueden procesar escalares, duplas, triuplas o cuatrouplas de manera simultánea, es decir, podemos guardar desde un único valor de punto flotante en cada texel hasta cuatro.

Para almacenar un valor escalar usamos el parámetro **GL_LUMINANCE**, **GL_RED**, **GL_GREEN**, **GL_BLUE** o **GL_ALPHA**; para almacenar una dupla podemos usar el parámtero **GL_RG**; para almacenar una triupla usamos el parámereo **GL_RGB**; y por último, para almacenar una cuatroupla usamos el parámetro **GL_RGBA**.

El siguiente parámetro a considerar es el **formato interno**, en éste parámetro se utilizan diferentes extensiones que son muy importantes, ya que dependiento de la extensión que se use, se nos permitira programar en GPUs de marca nVIDIA o ATI además de que indicarán el valor de presición de punto flotante que se usarán en las texturas. A continuación se presenta una tabla donde se describen los detalles de cada extención:

Extensión	Valores por texel	Presición por valor	Marca de GPU
GL_RGBA_FLOAT16_ATI	4	16 bites	ATI
GL_RGBA_FLOAT32_ATI	4	32 bites	ATI
GL_RGB_FLOAT16_ATI	3	16 bites	ATI
GL_RGB_FLOAT32_ATI	3	32 bites	ATI
GL_FLOAT_R16_NV	1	16 bites	nVIDIA
GL_FLOAT_R32_NV	2	32 bites	nVIDIA
GL_FLOAT_RG16_NV	2	16 bites	nVIDIA
GL_FLOAT_RG32_NV	3	32 bites	nVIDIA
GL_FLOAT_RGB16_NV	3	16 bites	nVIDIA
GL_FLOAT_RGB32_NV	4	32 bites	nVIDIA
GL_FLOAT_RGBA16_NV	4	16 bites	nVIDIA
GL_FLOAT_RGBA32_NV	4	32 bites	nVIDIA
GL_FLOAT_RGBA_NV	4	32 bites	nVIDIA
GL_FLOAT_RGBA_MODE_NV	4	32 bites	nVIDIA

Figura 2.11: Tabla de formato interno.

El último parámetro que se necesita considerar es el que nos indicará las dimensiones de la textura en la GPU en base a la longitud del vector en la CPU. La manera en que se obtienen estas dimensiones es la siguiente: si la longitud del vector es *N* un cuadrado perfecto y se utiliza un comando como GL_FLOAT_R16_NV, es decir que almacena un sólo valor de punto flotante por texel, entonces las dimenciones de la textura serán $\sqrt{N}$ por $\sqrt{N}$; por otro lado, si la longitud del vector es *N* un cuadrado perfecto que además es multiplo de cuatro y se usa un comando como GL_RGB_FLOAT32_ATI, entonces las dimensiones de la textura serán $\sqrt{N/4}$ por $\sqrt{N/4}$.

Ahora que conocemos la manera de enviar información de la CPU a la GPU, necesitamos se capaces de controlar los datos sobre los cuales hacemos los calculos. Para ello usaremos los siguientes comando:

```
1 glMatrixMode(GL\_PROJECTION);
2 glLoadIdentity();
3 gluOrtho2D(0.0, texSize1, 0.0, texSize2);
4 glMatrixMode(GL\_MODELVIEW);
5 glLoadIdentity();
6 glViewport(0, 0, texSize1, texSize2);
```

éstos comando son los encargados de, primeramente, proyectar los objetos tridimencionales (con coordenadas espaciales) a la pantalla bidimensional y , en segundo lugar, mapear inyectivamente los pixeles y texeles.

Nota: Los valores de los flotantes *texSize1* y *texSize2* representan las dimensiones de la textura.

Nota: Una textura sólo se puede usar para escribir datos o leer datos (pero no para ambos).

Para usar una textura como objetivo a entregar los datos, tenemos que unir la textura a un objeto framebuffer (FBO), el siguiente comando nos permite hacerlo:

void glFramebufferTexture2DEXT(GLenum *target*, GLenum *attachment*, GLenum *texturetarget*, GLuint *texture*, GLint *level*);
éste comando fija un objeto de textura como una representación de apego para un objeto framebuffer; *target* puede ser una de las siguientes expresiones, GL_READ_FRAMEBUFFER, GL_DRAW_FRAMEBUFFER o GL_FRAMEBUFFER; *attachment* debe ser uno de los puntos de fijación del framebuffer GL_COLOR_ATTACHMENTi, GL_DEPTH_ATTACHMENT, GL_STENCIL_ATTACHMENT, or GL_DEPTH_STENCIL_ATTACHMENT; *texturetarget* debe ser GL_TEXTURE_2D, GL_TEXTURE_RECTANGLE, GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_NEGATIVE_Z;
texture puede ser cero o diferente de cero; en el primer caso se indica que cualquier textura que limite *attachment* es descargado, subsecuentemente ninguna atadura a *attachment* es hecha, al mismo tiempo que los parámetros *texturetarget*, y *level* son ignorados.
En el segundo caso, *texture* debe ser el nombre de un objeto textura con *texturetarget* como tipo de la textura asociada al objeto textura. El parámetro *level* representa el nivel de mipmap asociada a la imagen textura. Si *texturetarget* es GL_TEXTURE_RECTANGLE, entonces el parámetro *level* debe ser cero.

Figura 2.12: Función glFramebufferTexture2DEXT.

Hasta ahora hemos adquirido las herramientas para crear texturas, que serán las enviadas a la GPU y objetos framebuffer. Lo único que falta para completar ésta sección es, como el nombre de la sección lo indica, enviar la información a la GPU. ésto se hace insertando los datos en la textura, lo cual es posible a través de dos maneras, la primera es con los siguientes comandos:

El primer comando ya se ha visto, sin embargo el segundo no, por ello detallaremos sus parémetros.

éstos datos son de manera general pero generalmente llenaremos éstos comandos de la manera en que se coloco anteriormente. La segunda manera de insertar los datos en la textura es a través de los siguientes comandos:

```
glBindTexture(texture_target, texID);
glTexSubImage2D(texture_target, 0, 0, 0, texSize, texSize, texture_format, GL_FLOAT, data);
```

Figura 2.13: Comandos para enviar información a la GPU.

void glTexSubImage2D(GLenum *target*, GLint *level*, GLint *xoffset*,GLint *yoffset*, GLsizei *width*, GLsizei *height*, GLenum *format*, GLenum *type*, const GLvoid **texels*); éste comando define una textura bidimensional que reemplaza toda o una parte de una subregion en 2D de la actual textura bidimensional existente. El parámetro *target* puede ser uno de las opciones del parámetro *level* del comando glTexImage2D. El parámetro *level* es el número que detalla el nivel de mipmap; *format* y *type* especifican el formato y tipo de datos de la textura; *texels* contiene los datos de la textura para la subimagen; los parámetros *width* y *height* son las dimensiones de la subregion que es reemplazada en la actual textura. *xoffset* y *yoffset* especifícan las compensaciones de los texeles en la dirección x y y, donde $(0, 0)$ se coloca en la esquina inferior izquierda, además de especificar donde se colocara la subimagen en la textura

Figura 2.14: Función glTexSubImage2D.

Después de conocer todos éstos parámetros, veremos la manera de enviar información a la GPU, para ello se necesitan seguir los siguientes pasos:

- Llamamos a los comandos que proyectan los objetos y mapean los pixeles con los texeles
- Crear un objeto framebuffer
- Generar el nombre de la textura
- Enlazamos el objeto textura con la textura objetivo, la textura objetivo especifíca el tipo de textura que se usará
- Apagamos el filtrado de datos (el cual es sólo util para los gráficos, el cual no es el caso) y especificamos el modo wrap, con el fin de especificar los parámetros de la textura
- Se reserva espacio en la memoria gráfica

```
glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT);
glRasterPos2i(0,0);
glDrawPixels(texSize,texSize,texture_format,GL_FLOAT,data);
```
El primer comando redirige la salida, el segundo comando usamos el origen como punto de referencia debido a que descargamos los datos completos en la textura, ésto con el tercer comando

Figura 2.15: Función glDrawBuffer.

- Adjuntamos la textura al objeto framebuffer
- Por último transferimos los datos a la textura

A continuación daremos un ejemplo, el cual utilizará los comando que hemos visto en las secciones pasadas.

```
1
2  GLuint FBO;
3  GLuint textura;
4  int texSize1 = 4, texSize2 = 4;
5  float datos[ 16 ];
6  glMatrixMode(GL_PROJECTION);
7  glLoadIdentity();
8  gluOrtho2D(0, texSize1, 0, texSize2);
9  glMatrixMode(GL_MODELVIEW);
10 glLoadIdentity();
11 glViewport(0, 0, texSize1, texSize2);
12 // creamos el FBO (off-screen framebuffer) y generamos el nombre de la textura
13 glGenFramebuffersEXT(1, &FBO);
14 glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, FBO);
15 //creamos texturas
16 glGenTextures(1, &textura);
17 //atamos la textura a un texture target
18 glBindTexture(GL_TEXTURE_RECTANGLE, textura);
19 //apagamos el filtrado y fijamos el modo wrap adecuado
20 glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
21 glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
22 glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_WRAP_S, GL_CLAMP);
23 glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_WRAP_T, GL_CLAMP);
24 //reservamos memoria grafica
25 glTexImage2D(GL_TEXTURE_RECTANGLE, 0, GL_RGBA_FLOAT32_ATI, texSize1, texSize2, 0, GL_RGBA, GL_FLOAT, datos);
26 //adjuntamos la textura al objeto framebuffer
27 glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_RECTANGLE,
28 *textura, 0);
29 //transefrimos los datos a la textura
30 glBindTexture(GL_TEXTURE_RECTANGLE, &texure);
31 glTexSubImage2D(GL_TEXTURE_RECTANGLE, 0, 0, 0, texSize1, texSize2, GL_RGBA, GL_FLOAT, datos);
32 glDeleteFramebuffersEXT(1, &FBO);
33 glDeleteTextures(1, &textura);
```

Figura 2.16: Ejemplo de un programa que envia datos a la GPU.

Como se puede apreciar en el ejemplo anterior, se enviaron 16 datos a la GPU, por ello es que las dimensiones de la textura se definieron como cuatro cada una. Por otro lado se puso GL_TEXTURE_RECTANGLE en el parámetro texture_target, ya que es la manera como se decidio enviar el arreglo a la textura (veáse pag 25). En el comando glTexImage2D se escribió GL_RGBA_FLOAT32_ATI en el parámetro *internalformat* ya que la tarjeta gráfica donde se programo éste ejemplo era de marca ATI. Por lo demár deben ser claros los parámetros expresados en el ejemplo.

II.2.3 Recepción de datos de la GPU al CPU

Es necesario remarcar que el ejemplo anterior sólo envía datos a la GPU, ahora necesitamos saber como leer datos que se encuentren en las texturas de la GPU. Existen dos maneras de leer los datos de la GPU, la primera manera es:

II.2.4 Shaders

Como ya se ha dicho anteriormente los shaders son programas que pueden ejecutar diversos cálculos sobre los pixeles de manera independiente. Para poder hacer uso eficiente de éstos shaders nos adentraremos en un poco de teoría, la cual nos dira como hacer uso de la información en la GPU para hacer cálculos y por supuesto, la manera de programar los shaders.

```
glBindTexture(texture_target,texID);
glGetTexImage(texture_target, 0, texture_format, GL_FLOAT, datos);
```

y la segunda manera es:

```
glReadBuffer(GL_COLOR_ATTACHMENT0_EXT);
glReadPixels(0, 0, texSize1, texSize2, texture_format, GL_FLOAT, datos);
```

Figura 2.17: Función glBindTexture.

El lenguaje para shaders de OpenGL, es decir la GLSL, está basada en la sintaxis del lenguaje de programación ANSI C; además de que la estructura básica de los programas escritos en GLSL es la misma que para los programas escritos en C. Sin emargo se han agregado muchas cosas, algunas de éstas son:

- Variables tipo vector para valores de punto flotante, entero y boleano, referidos como vec2, vec3, vec4; para dos, tres o cuatro valores de punto flotante.
- Variables tipo matriz
- Un tipo básico es el SAMPLER, el cual fue agregado para crear la manera en la cual el shader accesa a la memoria de la textura, la variable de tipo sampler2D es usada para accesar a una textura 2D, mientras que la variable de tipo sampler2DRect es usada para accesar a una textura rectangular bidimensional.
- Los calificadores de variables globales, **in, out** y **uniform** especifican para lo que sirve el tipo de variable, de entrada o salida. Las variables **in** cominican el valor cuando cambia la variable, las variables **uniform** comunican los cambios en los valores de las variables de la aplicación a algun shader, las variables **out** comunican frecuentemente los cambios en los valores de un shader a una etapa subsecuente de procesamiento.

Con esto, es posible entender el siguiente ejemplo de fragment shader.

```
1
2  uniform sampler2DRect TextureX;
3  uniform sampler2DRect TextureY;
4  uniform float escalar;
5  void main(void){
6  vec4 x = texture2DRect(TextureX, gl_TexCoord[0].st);
7  vec4 y = texture2DRect(TextureY, gl_TexCoord[0].st);
8  float alpha = escalar;
9  gl_FragColor = x + y * alpha;
10 }
```

Figura 2.18: Ejemplo de Shader simple.

Como se puede apreciar en el ejemplo anterior, las variables TextureX y Texture Y son de tipo uniform sampler2DRect, es decir, éstas variables son las que leen los datos de la memoria de la textura y los comunican al shader; igualmente la variable escalar comunica un valor flotante al shader. Las variables x, y son de tipo vec4 debido al formato que se le dio a la imagen textura, el cual fue GL_RGBA. La variable x se igualo al comando

GLuint glCreateProgram();
éste comando crea un programa para un shader vacío. El valor de retorno es un entero diferente de cero, o cero si algun error ocurrio.

Figura 2.19: Función glCreateProgram.

GLuint glCreateShader(GLenum *type*);
Asigna un objeto shader, *type* debe ser GL_VERTEX_SHADER o GL_FRAGMENT_SHADER. El valor de retorno es un entero diferente de cero, o cero si algun error ocurrio.

Figura 2.20: Función glCreateShader.

texture2DRect(TextureX, gl_TexCoord[0].st) con el fin de tomar la muestra del valore de la textura, al igual que y.

Nota: Es necesario recordar que los shader son programas que se ejecutan dentro de la GPU y los calculos se realizan en cada pixel, es decir que si enviamos dos arreglos de dimension 16, dentro de la GPU necesitamos obtener el n- esimo valor de cada arreglo para hacer la operacion deseada, es por ello que utilizamos el comando texture2DRect(*nameTexture*, gl_TexCoord[0].st), el cual llama al comando gl_TexCoord[n] que regresa el n-esimo valor coordenado de la textura.

Por último para poder escribir en la salida necesitamos escribir en la variable incorporada gl_FragColor de la forma en que se muestra en el ejemplo anterior.

A pesar de que ya conocemos la manera de enviar información de la CPU a la GPU, es también necesario conocer la manera de poder atar el nombre de una variable en la CPU para poder usarla en la GPU, para ello se utilizarán los siguientes comandos en el mismo orden.

Realizados éstos comandos, es necesario fijar los arreglos de entrada a las texturas, ésto se hace atando nuestras texturas a diferentes unidades de textura para luego pasar éstas unidades a nuestros parámetros uniform; todo ésto a traves de la siguiente serie de comandos.

En el caso de que se quiera enviar una constante a la GPU, en lugar de utilizar los comandos anteriores sólo es necesario utilizar el siguiente comando:

void glShaderSource(GLuint *shader*, GLsizei *count*, const GLchar ***string*, const GLint **length*);
Asocia la fuente de un shader con el objeto shader *shader*; *string* es un arreglo de *count* arreglos GLchar que componen la fuente del shader;*length* puede ser uno de tres valores. Si *length* es NULL, entonces se supone que cada arreglo provisto es terminado como null. De otra manera, *length* tiene *count* elementos, cada uno de los cuales especifican la longitud de la correspondiente entrada en *string*

Figura 2.21: Función glShaderSource.

void glCompileShader(GLuint *shader*);
éste comando compila el código fuente para *shader*. El resultado de la compilación puede ser preguntado a través del comando glGetShaderiv() con argumento GL_COMPILE_STATUS.

Figura 2.22: Función glCompileShader.

void glAttachShader(GLuint *program*, GLuint *shader*);
Asocia el objeto shader, *shader*, con el programa del shader, *program*. Un objeto shader puede ser adjuntado a un programa shader en cualquier momento, aunque su funcionalidad será posible sólo después de un enlaze exitoso del programa del shader. Un objeto shader puede ser adjuntado a múltiples programas de shaders simultáneamente.

Figura 2.23: Función glAttachShader.

void glLinkProgram(GLuint *program*);
Procesa todos los objetos shader adjuntados a *program* para generar un programa shader completo. El resultado de la operación de enlaze puede ser obtenido con el comando glGet-Programiv() con el parámetro GL_LING_STATUS. Se regresara GL_TRUE en caso de un enlazado exitoso y GL_FALSE se retornara en caso contrario.

Figura 2.24: Función glLinkProgram.

glActiveTexture(GLenum *texUnit*);
glBindTexture(*texture_target*, *texID*);
glUniform1iARB(GLint *location*, TYPE *value*);
Recuerde que el parámetro *texUnit* del comando glActiveTexture es una consante simbólica de la forma GL_TEXTURE*i*.
Por otro lado el comando glUniform1iARB fija los valores de la variable uniforme asociada con el indice *location* y *value* tendra el mismo valor que *i* en el comando glActiveTexture

Figura 2.25: Función glActiveTexture.

glUniform1fARB(GLint *location*, TYPE *value*);
Donde *value* será el nombre de la constante a enviar

Figura 2.26: Función glUniform1fARB.

GLint glGetUniformLocation(GLuint *program*, const char **name*)
Regresa el índice de la variable uniforme *name* asociada con el shader *program*. *name* puede ser el nombre de una variable, un elemento de un arreglo. Un valor de -1 es regresado si *name* no corresponde a una variable uniforme en el shader activo *program*

Figura 2.27: Función glGetUniformLocation.

Realizados éstos comandos para cada uno de los arreglos y constantes que se quieran enviar, se utilizan los siguientes comandos:

Para comprender mejor los comandos anteriores damos un ejemplo, donde se utiliza el shader anteriormente expuesto, el cual se mostrará nuevamente.

```
1
2  const GLchar FragmetShader[] = {
3  uniform sampler2DRect TextureX;
4  uniform sampler2DRect TextureY;
5  uniform float escalar;
6  void main(void){
7  vec4 x = texture2DRect(TextureX, gl_TexCoord[0].st);
8  vec4 y = texture2DRect(TextureY, gl_TexCoord[0].st);
9  float alpha = escalar;
10 gl_FragColor = x + escalar * y;
11 }"};
12 GLuint shader, program;
13 program = glCreateProgram();
14 shader = glCreateShader(GL_FRAGMENT_SHADER);
15 glShaderSource(shader, 1, &vShader, NULL);
16 glCompileShader(shader);
17 glAttachShader(program, shader);
18 glLinkProgram(program);
19 glActiveTexture(GL_TEXTURE1);
20 glBindTexture(GL_TEXTURE_RECTANGLE, *XTexture);
21 glUniform1iARB(ParamX, 1);
22 glActiveTexture(GL_TEXTURE2);
23 glBindTexture(GL_TEXTURE_RECTANGLE, *YTexture);
24 glUniform1iARB(ParamY, 2);
25 glUniform1fARB(Alpha, escalar);
26 ParamX = glGetUniformLocation(program,"TextureX");
27 glUniform1i(ParamX, 1);
28 ParamY = glGetUniformLocation(program,"TextureY");
29 glUniform1i(ParamY, 2);
30 Alpha = glGetUniformLocation(program,.escalar");
31 glUniform1f(Alpha, escalar);
```

Figura 2.28: Ejemplo de un programa que incorpora un Shader que realiza la operacion x + αy con vectores.

II.2.5 Forzando los calculos

Debido a la proyeccion y punto de vista fijados, lo único que tenemos que hacer es rellenar un rectangulo que cubra la ventana completa por la que estamos observando. Para definir un rectangulo usamos las instrucciones usuales de OpenGL (vease en bibliografia). A través de ésto especificamos directamente los cuatro vértices del rectángulo. Al mismo tiempo que asignamos las coordenadas de la textura como atributos de los vértices a las cuatro esquinas.

Los cuatro vértices serán transformados al espacio de la pantalla por la estapa del vértice de función fija la cual no programamos. La rasterización, una parte de la función fija de la tubería gráfica localizada entre la etapa del vertex process y fragment process, ejecutará una interpolación bilineal para cada pixel que está cubierto por el rectangulo, interpolando la posición de los pixeles y los atributos de los vertices. ésto generara un fragmento para

cada pixel cubierto por el rectangulo. Los valores interpolados pasarán automáticamente al fragment shader de bido a que usamos la unión cuando escribimos el shader. En forma breve, el digujar un rectangulo relleno sirve como generador de flujo de datos para el fragmento del programa.

Si utilizamos texturas rectangulares, usamos los siguientes comandos:

```
1 glPolygonMode(GL\_FRONT,GL\_FILL);
2 // and render quad
3 glBegin(GL\_QUADS);
4
5 glTexCoord2f(0.0, 0.0);                  glVertex2f(0.0, 0.0);
6 glTexCoord2f(texSize1, 0.0);             glVertex2f(texSize1, 0.0);
7 glTexCoord2f(texSize1, texSize2);        glVertex2f(texSize1, texSize2);
8 glTexCoord2f(0.0, texSize2);             glVertex2f(0.0, texSize2);
9 glEnd();
```

Donde texSize1 y texSize2 son las dimensiones de la textura

Ejecutando el shader y mostrando resultados

Una vez hechos los cálculos deseados, necesitamos establecer los shaders que queremos que se ejecuten, para luego regresar los datos y utilizarlos; la manera de establecer los shaders a usar, es a través del siguiente comando.

void glUseProgram(GLuint *program*);
Usa *program* para el procesamiento de vértices o de fragmentos, dependiendo del tipo de shader creado con el comando glCreateShader(). Si *program* es cero, ningun shader es usado para procesarse.
Mientras un programa esta en usa, éste puede tener un nuevo shader adjuntado, compilar shaders adjuntados o separar shaders adjuntados. Este puede reenlazarse; si el enlaze es exitoso, el nuevo shader enlazado reemplaza al shader anterior; si el enlaze falla, el shader permanece activo.

Figura 2.29: Función glUseProgram.

II.2.6 Ejemplo de shader

A continuación mostraremos un ejemplo de todo el proceso para enviar datos, ejecutar el shader y recibir los datos tratados por el shader. Para el lector poco observador destacamos el hecho de que el código del shader se encuentra en las líneas 166 a 174, las cuales se encuentran entre comillas.

```
1
2  // linkers glew32s.lib, glew32.lib, libglu.a, libopengl32.a
3  #include <windows.h>
4  #define GLEW_STATIC
5  #include <gl/glew.h>
6  #include <stdio.h>
7  #include <conio.h>
8  LRESULT CALLBACK WindowProc1(HWND, UINT, WPARAM, LPARAM);
9  LRESULT CALLBACK WindowProc2(HWND, UINT, WPARAM, LPARAM);
10 int WindowClass(HINSTANCE, HINSTANCE, LPSTR, int);
11 void EnableOpenGL(HWND hwnd, HDC*, HGLRC*);
12 void DisableOpenGL(HWND, HDC, HGLRC);
13 void ActiveGPU();
14 void MapeoProyeccion(int, int);
15 void ProcesoAtarLaTextura(GLuint *, float *, int, int);
16 void iniciandoGLSL();
17 GLuint Load(const char*);
18 void Usar(GLuint);
19 void Computo(int, int);
20
21 HDC hDC;
22 HANDLE mutex;
```

```
GLint ParamX, ParamY, Alpha;
GLint program;
GLuint *XTexture, *YTexture,*ZTexture;
float escalar = 2.0;

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow){
    HWND hwnd;
    MSG msg;

    //mutex = CreateMutex(0,FALSE,0);

    if (!WindowClass(hInstance, hPrevInstance, lpCmdLine, nCmdShow))
        return 0;

    /* create main window */
    hwnd = CreateWindowEx(0,
                          "GLconGPU",
                          "OpenGL Sample",
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT,
                          CW_USEDEFAULT,
                          512,
                          512,
                          NULL,
                          NULL,
                          hInstance,
                          NULL);

    ShowWindow(hwnd, nCmdShow);

    /* program main loop */
    while (GetMessage (&msg, NULL, 0, 0))
     {
          TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;
     }
    /* destroy the window explicitly */
    DestroyWindow(hwnd);
    //CloseHandle(mutex);
    system("pause");
    return msg.wParam;
}
void    ActiveGPU(){
    //definimos los vectores x[], y[] y un escalar que enviaremos a la GPU
    float x[16], y[16], result[16];
    int i, texSize1 = 4, texSize2 = 4;  //definimos las dimensiones de la textura
    //definimos las texturas
    GLuint texturas[3], FBO;

    for (i = 0; i < 16; i++)     //le damos valores a las variables
    {
        x[i] = 0.01 * (i + 1);
        y[i] = 0.01 * (16 - i);
    }

    MapeoProyeccion(texSize1, texSize2);
    glFinish();

    // create FBO (off-screen framebuffer)
    glGenFramebuffersEXT(1, &FBO);
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, FBO);

    //creamos texturas
    glGenTextures(3, texturas);
    ZTexture = &texturas[0];
    XTexture = &texturas[1];
    YTexture = &texturas[2];

    ProcesoAtarLaTextura(ZTexture, 0, texSize1, texSize2);
    ProcesoAtarLaTextura(XTexture, y, texSize1, texSize2);
    ProcesoAtarLaTextura(YTexture, x, texSize1, texSize2);

    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_RECTANGLE, *ZTexture, 0);
    glFinish();

    iniciandoGLSL();

    Computo(texSize1, texSize2);

    glFinish();

    glReadBuffer(GL_COLOR_ATTACHMENT0_EXT);
    glReadPixels(0, 0, texSize1, texSize2, GL_RED, GL_FLOAT, result);

    for(i = 0; i < 16; i++)
       printf("Datos obtenidos %f\n", result[i]);
    //getch();

    glDeleteFramebuffersEXT(1, &FBO);
    glDeleteTextures(3, texturas);
    glFinish();
}
```

```
115 void    MapeoProyeccion(int texSize1, int texSize2){
116     glMatrixMode(GL_PROJECTION);
117     glLoadIdentity();
118     gluOrtho2D(0, texSize1, 0, texSize2);
119     glMatrixMode(GL_MODELVIEW);
120     glLoadIdentity();
121     glViewport(0, 0, texSize1, texSize2);
122 }
123 void    ProcesoAtarLaTextura(GLuint *textura, float *datos, int texSize1, int texSize2){
124     //atamos la textura a un texture target
125     glBindTexture(GL_TEXTURE_RECTANGLE, *textura);
126     //apagamos el filtrado y fijamos el modo wrap adecuado
127     glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
128     glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
129     glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_WRAP_S, GL_CLAMP);
130     glTexParameteri(GL_TEXTURE_RECTANGLE, GL_TEXTURE_WRAP_T, GL_CLAMP);
131     //reservamos memoria grafica
132     glTexImage2D(GL_TEXTURE_RECTANGLE, 0, GL_RGBA_FLOAT32_ATI, texSize1, texSize2, 0, GL_RED, GL_FLOAT, datos);
133 }
134 GLuint  Load(const char* vShader){
135     GLuint shader, program;
136     GLint succes;
137 
138     program = glCreateProgram();
139         if (program == 0)
140             printf("Un error ocurrio al crear un programa vacio\n\n");
141     if (vShader != NULL) {
142         shader = glCreateShader(GL_FRAGMENT_SHADER);
143             if (shader == 0)
144                 printf("Un error ocurrio al almacenar el objeto shader\n\n");
145         glShaderSource(shader, 1, &vShader, NULL);
146         glCompileShader(shader);
147             glGetShaderiv(shader, GL_COMPILE_STATUS, &succes );
148             if (succes == FALSE)
149                 printf("Error al compilar\n\n");
150         glAttachShader(program, shader);
151     }
152     else
153         printf("cShader es NULL\n\n");
154 
155     glLinkProgram(program);
156         glGetProgramiv(program, GL_LINK_STATUS, &succes );
157         if (succes == GL_FALSE)
158             printf("Error al enlazar el programa\n\n");
159     return program;
160 }
161 void    Usar(GLuint program){
162     glUseProgram(program);
163 }
164 void    iniciandoGLSL(){
165     const GLchar FragmetShader[] = {
166         "uniform sampler2DRect TextureX;\
167         uniform sampler2DRect TextureY;\
168         uniform float escalar;\
169         void main(void){\
170             vec4 x = texture2DRect(TextureX, gl_TexCoord[0].st);\
171             vec4 y = texture2DRect(TextureY, gl_TexCoord[0].st);\
172             float alpha = escalar;\
173             gl_FragColor = x;\
174         }"
175         };
176     program = Load(FragmetShader);
177     Usar(program);
178     glFinish();
179 
180     glActiveTexture(GL_TEXTURE1);
181     glBindTexture(GL_TEXTURE_RECTANGLE, *XTexture);
182     glUniform1iARB(ParamX, 1);
183 
184     glActiveTexture(GL_TEXTURE2);
185     glBindTexture(GL_TEXTURE_RECTANGLE, *YTexture);
186     glUniform1iARB(ParamY, 2);
187 
188     glUniform1fARB(Alpha, escalar);
189     glFinish();
190 
191     ParamX = glGetUniformLocation(program,"TextureX");
192         glUniform1i(ParamX, 1);
193     ParamY = glGetUniformLocation(program,"TextureY");
194         glUniform1i(ParamY, 2);
195     Alpha = glGetUniformLocation(program,"escalar");
196         glUniform1f(Alpha, escalar);
197     glFinish();
198 }
199 void    Computo(int texSize1, int texSize2){
200     // make quad filled to hit every pixel/texel
201     glPolygonMode(GL_FRONT,GL_FILL);
202     // and render quad
203     glColor3f(1.0, 0.0, 1.0);
204     glBegin(GL_QUADS);
205         glTexCoord2f(0.0, 0.0);             glVertex2f(0.0, 0.0);
206         glTexCoord2f(texSize1, 0.0);        glVertex2f(texSize1, 0.0);
```

```
207          glTexCoord2f(texSize1, texSize2);     glVertex2f(texSize1, texSize2);
208          glTexCoord2f(0.0, texSize2);         glVertex2f(0.0, texSize2);
209      glEnd();
210      SwapBuffers(hDC);
211      glFinish();
212  }
213  LRESULT CALLBACK WindowProc2(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam){
214      static HGLRC hRC;
215      switch (uMsg)
216      {
217          case WM_CREATE:
218              WaitForSingleObject(mutex, INFINITE);
219                  EnableOpenGL (hwnd,&hDC,&hRC);
220                  wglMakeCurrent(hDC,hRC);
221                      glewInit();
222
223
224                      glFinish();
225                  wglMakeCurrent(NULL,NULL);
226              ReleaseMutex(mutex);
227              return 0;
228
229          case WM_PAINT:
230              wglMakeCurrent(hDC,hRC);
231              ActiveGPU();
232              wglMakeCurrent(NULL,NULL);
233              PostQuitMessage(0);
234              //Computo(4, 4);
235              return 0;
236
237          case WM_CLOSE:
238              PostQuitMessage(0);
239              return 0;
240
241          case WM_DESTROY:
242              return 0;
243
244          case WM_KEYDOWN:
245          {
246              switch (wParam)
247              {
248                  case VK_ESCAPE:
249                      PostQuitMessage(0);
250                  return 0;
251              }
252          }
253          return 0;
254
255          default:
256              return DefWindowProc(hwnd, uMsg, wParam, lParam);
257      }
258
259      return 0;
260  }
261  LRESULT CALLBACK WindowProc1(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam){
262      switch (uMsg)
263      {
264          case WM_CREATE:
265              CreateWindow (TEXT("GPUcalculation"), NULL, WS_CHILDWINDOW | WS_BORDER | WS_VISIBLE ,
266                                      0, 0, 250, 250, hwnd, (HMENU) 10001, ((LPCREATESTRUCT)lParam)->hInstance, NULL) ;
267              return 0;
268
269          case WM_CLOSE:
270              PostQuitMessage(0);
271              return 0;
272
273          case WM_DESTROY:
274              return 0;
275
276          case WM_KEYDOWN:
277          {
278              switch (wParam)
279              {
280                  case VK_ESCAPE:
281                      PostQuitMessage(0);
282                  return 0;
283              }
284          }
285          return 0;
286
287          default:
288              return DefWindowProc(hwnd, uMsg, wParam, lParam);
289      }
290
291      return 0;
292  }
293  int     WindowClass(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow){
294      WNDCLASSEX wcex;
295
296      /* register window class */
297      wcex.cbSize = sizeof(WNDCLASSEX);
298      wcex.style = CS_OWNDC;
```

```
299     wcex.lpfnWndProc = WindowProc1;
300     wcex.cbClsExtra = 0;
301     wcex.cbWndExtra = 0;
302     wcex.hInstance = hInstance;
303     wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
304     wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
305     wcex.hbrBackground = (HBRUSH) COLOR_BACKGROUND;
306     wcex.lpszMenuName = NULL;
307     wcex.lpszClassName = TEXT("GLconGPU");
308     wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
309
310     if (!RegisterClassEx(&wcex))
311         return 0;
312     /* register window class */
313     wcex.cbSize = sizeof(WNDCLASSEX);
314     wcex.style = CS_OWNDC;
315     wcex.lpfnWndProc = WindowProc2;
316     wcex.cbClsExtra = 0;
317     wcex.cbWndExtra = 0;
318     wcex.hInstance = hInstance;
319     wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
320     wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
321     wcex.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
322     wcex.lpszMenuName = NULL;
323     wcex.lpszClassName = TEXT("GPUcalculation");
324     wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);;
325
326     if (!RegisterClassEx(&wcex))
327         return 0;
328     return 1;
329 }
330 void    EnableOpenGL(HWND hwnd, HDC* hDC, HGLRC* hRC){
331     PIXELFORMATDESCRIPTOR pfd;
332
333     int iFormat;
334
335     /* get the device context (DC) */
336     *hDC = GetDC(hwnd);
337
338     /* set the pixel format for the DC */
339     ZeroMemory(&pfd, sizeof(pfd));
340
341     pfd.nSize = sizeof(pfd);
342     pfd.nVersion = 1;
343     pfd.dwFlags = PFD_DRAW_TO_WINDOW |
344                   PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
345     pfd.iPixelType = PFD_TYPE_RGBA;
346     pfd.cColorBits = 24;
347     pfd.cDepthBits = 16;
348     pfd.iLayerType = PFD_MAIN_PLANE;
349
350     iFormat = ChoosePixelFormat(*hDC, &pfd);
351
352     SetPixelFormat(*hDC, iFormat, &pfd);
353
354     /* create and enable the render context (RC) */
355     *hRC = wglCreateContext(*hDC);
356
357     wglMakeCurrent(*hDC, *hRC);
358 }
359 void    DisableOpenGL (HWND hwnd, HDC hDC, HGLRC hRC){
360     wglMakeCurrent(NULL, NULL);
361     wglDeleteContext(hRC);
362     ReleaseDC(hwnd, hDC);
363 }
```

Figura 2.30: Ejemplo de un Programa que envía y recibe datos, además de ejecutar un shader.

II.3. GLEW

En esta sección del capítulo se hablará acerca de **GLEW** (OpenGL Extension Wrangler Library), GLEW es una biblioteca miltiplataforma escrita en C/C++ creada con la finalidad de ayudar a OpenGL para determinar. GLEW consta de de diversos métodos que ayudan a mejorar la rapidez de ejecución en algoritmos que utilizan OpenGL, pues en ocaciones es mejor utilizar la biblioteca GLEW.H en comparación con la biblioteca GL.H.

En la biblioteca GLEW se definen las extensiones, las cuales son apuntadores a funciones que se ejecutaran sobre la GPU.

Cabe aclarar que cuando un algoritmo requiere extensiones o funciones que se encuentran en versiones de OpenGL inferiores o iguales a la 2.0, la biblioteca GLEW manda a ejecutar a la biblioteca GL.

Como nota particular se resalta el hecho de que en ocaciones, cuando se agrega la biblioteca GLEW es necesario utilizar algunos enlces, tales como: glew32.s y glew32 así como los que generalmente se colocan para utilizar OpenGL, lo cuales son: OpenGL32 y glu32. La necesidad de agregar dichos enlaces viene de la necesidad de utilizar funciones o extensiones que superen la versión 1.1 de OpenGL.

A continuación se presenta un ejemplo de un algoritmo que utiliza la biblioteca GLEW, seguido de una imagen que muestra el resultado de la ejecución de dicho algoritmo, es decir un toro.

```
1  #include <windows.h>
2  #include <gl/glew.h>
3  #include <math.h>
4  LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);
5  void EnableOpenGL(HWND hwnd, HDC*, HGLRC*);
6  void DisableOpenGL(HWND, HDC, HGLRC);
7  void opengl1();
8  void opengl2();
9  void init();
10 void display();
11 void vista();
12 void recta();
13 static void torus(int numc, int numt);
14 float alpha = 0.0, betha = 2.0, mas = 2.0;
15 float theta = 0.0f;
16 HDC hDC;
17
18 int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow){
19     WNDCLASSEX wcex;
20     HWND hwnd;
21     HGLRC hRC;
22     MSG msg;
23     BOOL bQuit = FALSE;
24
25     /* register window class */
26     wcex.cbSize = sizeof(WNDCLASSEX);
27     wcex.style = CS_OWNDC;
28     wcex.lpfnWndProc = WindowProc;
29     wcex.cbClsExtra = 0;
30     wcex.cbWndExtra = 0;
31     wcex.hInstance = hInstance;
32     wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
33     wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
34     wcex.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
35     wcex.lpszMenuName = NULL;
36     wcex.lpszClassName = "GLSample";
37     wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);;
38
39
40     if (!RegisterClassEx(&wcex))
41         return 0;
42
43     /* create main window */
44     hwnd = CreateWindowEx(0,
45                           "GLSample",
46                           "OpenGL Sample",
47                           WS_OVERLAPPEDWINDOW,
48                           CW_USEDEFAULT,
49                           CW_USEDEFAULT,
50                           512,
51                           512,
52                           NULL,
53                           NULL,
54                           hInstance,
55                           NULL);
56
57     ShowWindow(hwnd, nCmdShow);
58
59     /* enable OpenGL for the window */
60     EnableOpenGL(hwnd, &hDC, &hRC);
61
62     /* program main loop */
63     while (!bQuit)
64     {
65         /* check for messages */
66         if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
67         {
68             /* handle or dispatch messages */
69             if (msg.message == WM_QUIT)
```

```
            {
                bQuit = TRUE;
            }
            else
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }
        else
        {
            /* OpenGL animation code goes here */
            //opengl();
        }
    }

    /* shutdown OpenGL */
    DisableOpenGL(hwnd, hDC, hRC);

    /* destroy the window explicitly */
    DestroyWindow(hwnd);

    return msg.wParam;
}

void opengl2(){
    init();
    display();
}
void init(){
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glOrtho(-2.5, 2.5, -2.5, 2.5, -2.5, 2.5);
}
void display(){
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glDepthFunc(GL_LEQUAL);  //se dibujan los pixeles si su coordenada Z estan mas cercanos
     glEnable(GL_DEPTH_TEST); //habilitamos la comprobacion de la profundad

     glColor3f(1.0, 1.0, 1.0);
    glPushMatrix();

    vista();
    recta();
    torus(25, 50);
    glPopMatrix();

     glFlush();
     SwapBuffers(hDC);
}
void vista(){
     glRotatef (alpha, 1.0f, 0.0f, 0.0f);
     glRotatef (-betha, 0.0f, 1.0f, 0.0f);
     //glTranslatef(0.0f, 0.0f, mas);
     glScalef(mas, mas, mas);        //es mejor que el glTranslatef pues mueve los ejes
}
void recta(){
     float i;
     glColor3f(0.5, 0.5, 0.0);
     glBegin(GL_LINES);
         glVertex3f(-100.0, 0.0, 0.0);
         glVertex3f( 100.0, 0.0, 0.0);
     glEnd();
     glColor3f(0.0, 0.5, 0.0);
     glBegin(GL_LINES);
         glVertex3f(0.0, -100.0, 0.0);
         glVertex3f(0.0,  100.0, 0.0);
     glEnd();
     glColor3f(1.0, 0.0, 0.0);
     glBegin(GL_LINES);
         glVertex3f(0.0, 0.0, -100.0);
         glVertex3f(0.0, 0.0,  100.0);
     glEnd();
     for(i = -50.0; i < 50.0; i = i + 0.1)          //puntos del eje X
     {
           glPointSize(3.0);
           glBegin(GL_POINTS);
                glVertex3f(i, 0.0, 0.0);
           glEnd();
     }
     for(i = -100; i < 100; i = i + 0.1)          //puntos del eje Y
     {
           glPointSize(3.0);
           glBegin(GL_POINTS);
                glVertex3f(0.0, i, 0.0);
           glEnd();
     }
     for(i = -50.0; i < 50.0; i = i + 0.1)          //puntos del eje z
     {
           glPointSize(3.0);
           glBegin(GL_POINTS);
```

```
                glVertex3f(0.0, 0.0, i);
            glEnd();
      }
}
static void torus(int numc, int numt){
   int i, j, k;
   double s, t, x, y, z, twopi;
   float a = 0.1, b = 0.1;

   twopi = 2 * M_PI;

   for (i = 0; i < numc; i++) {
      glBegin(GL_TRIANGLE_STRIP);//);//GL_QUAD_STRIP);
      for (j = 0; j <= numt; j++) {
         glColor3f (a, b, 1.0);
         for (k = 1; k >= 0; k--) {
            s = (i + k)  % numc + 0.5;
            t = j  % numt;

            x = (1+.1*cos(s*twopi/numc))*cos(t*twopi/numt);
            y = (1+.1*cos(s*twopi/numc))*sin(t*twopi/numt);
            z = .1 * sin(s * twopi / numc);
            glVertex3f(x, y, z);
         }
         a = a + 0.1;     b = b - 0.1;
         if (a > 0.9)
         {
            a = 0.1;      b = 0.9;
         }
      }
      glEnd();
   }
}
void opengl1(){
     glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
            glClear(GL_COLOR_BUFFER_BIT);

            glPushMatrix();
            glRotatef(theta, 0.0f, 0.0f, 1.0f);

            glBegin(GL_TRIANGLES);

                glColor3f(1.0f, 0.0f, 0.0f);   glVertex2f(0.0f,   1.0f);
                glColor3f(0.0f, 1.0f, 0.0f);   glVertex2f(0.87f,  -0.5f);
                glColor3f(0.0f, 0.0f, 1.0f);   glVertex2f(-0.87f, -0.5f);

            glEnd();

            glPopMatrix();
            glFlush();
            SwapBuffers(hDC);

            theta += 1.0f;
            Sleep (1);
}
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam){
    switch (uMsg)
    {
        case WM_PAINT:
            opengl2();
            return 0;

        case WM_CLOSE:
            PostQuitMessage(0);
        break;

        case WM_MOUSEMOVE:
        if (LOWORD (lParam) > 0)
            alpha = HIWORD (lParam);
        if (HIWORD (lParam) > 0)
            betha = LOWORD (lParam);
        return 0;

        case WM_DESTROY:
            return 0;

        case WM_KEYDOWN:
        {
            switch (wParam)
            {
                case 0xBB :
                    mas = mas + 0.05;
                    return 0;
                case 0xBD :
                    mas = mas - 0.05;
                    return 0;
                case VK_ESCAPE:
                    PostQuitMessage(0);
                break;
            }
        }
        break;
```

```
254
255         default:
256             return DefWindowProc(hwnd, uMsg, wParam, lParam);
257     }
258
259     return 0;
260 }
261
262 void EnableOpenGL(HWND hwnd, HDC* hDC, HGLRC* hRC){
263     PIXELFORMATDESCRIPTOR pfd;
264
265     int iFormat;
266
267     /* get the device context (DC) */
268     *hDC = GetDC(hwnd);
269
270     /* set the pixel format for the DC */
271     ZeroMemory(&pfd, sizeof(pfd));
272
273     pfd.nSize = sizeof(pfd);
274     pfd.nVersion = 1;
275     pfd.dwFlags = PFD_DRAW_TO_WINDOW |
276                   PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
277     pfd.iPixelType = PFD_TYPE_RGBA;
278     pfd.cColorBits = 24;
279     pfd.cDepthBits = 16;
280     pfd.iLayerType = PFD_MAIN_PLANE;
281
282     iFormat = ChoosePixelFormat(*hDC, &pfd);
283
284     SetPixelFormat(*hDC, iFormat, &pfd);
285
286     /* create and enable the render context (RC) */
287     *hRC = wglCreateContext(*hDC);
288
289     wglMakeCurrent(*hDC, *hRC);
290 }
291
292 void DisableOpenGL (HWND hwnd, HDC hDC, HGLRC hRC){
293     wglMakeCurrent(NULL, NULL);
294     wglDeleteContext(hRC);
295     ReleaseDC(hwnd, hDC);
296 }
```

Figura 2.31: Ejemplo de un algoritmo que utiliza la biblioteca GLEW.

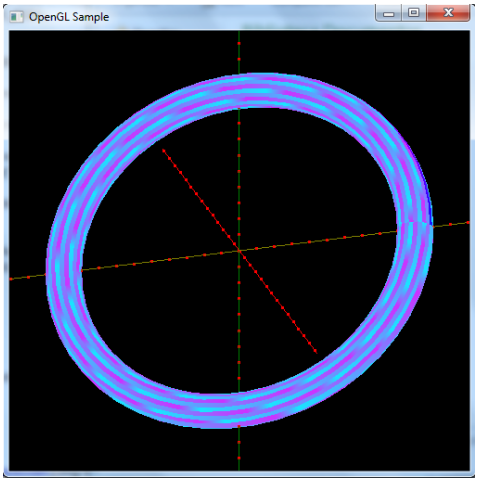

Figura 2.32: Gráfica de un toro resultado de ejecutar el algoritmo de la figura 2.31.

III. METODOLOGÍA

Como recuento diremos que hasta ahora hemos conocido las razones que motivaron a la creación del proyecto sobre el cual se basa la presente tesis, además de las diferentes formas que se pueden seguir para la creación de la biblioteca matemática GPU Math y sobre todo la manera que se utilizó para crearlo.

En éste capítulo al fin podremos hablar de GPU Math desde un punto de vista más enfocado a la programación, es decir, hablaremos de como funciona, como es que hace lo que hace y por supuesto, lo que puede hacer para más tarde hablar de los resultados que nos trajo ésta biblioteca, es decir los beneficios que nos trajo el programar éstos algoritmos en la GPU en lugar de programarlos en la CPU.

III.1. Motor de la biblioteca

La biblioteca GPU Math consta de una serie de diversos algoritmos matemáticos que pueden ser utilizados por una amplia variedad de personas como por ejemplo: estudiantes de licenciatura o posgrado de alguna carrera de ingeniería, matemáticas, física o con alguna afinidad a las matemáticas y que posean alguna necesidades de una obtención rápida de soluciones para diversos problemas; empresas que posean una gran cantidad de datos por ser analizados y que necesiten de una rápida obtención de información y que no deseen desperdiciar demasiados recursos para ello.

La manera en como se implementaron todos éstos algoritmos fue a través del "motor de la biblioteca"; cabe aclarar que en realidad son varias biblioteca las que se crearon, una por cada familia de algoritmos pertenecientes a una misma rama de las matemáticas, es decir, una biblioteca para Álgebra Lineal, una biblioteca para Cálculo, una biblioteca para Procesamiento de Imágenes, una biblioteca para Inteligencia Artificial . Puesto que cada biblioteca posee varios algoritmos implementados en la GPU se optó por programarlos en diferentes archivos y así dejarlos como archivos ejecutables, esto con el fin de que en el momento de la ejecución de la biblioteca sólo se mande una llamada al archivo ejecutable que represente el algoritmo que se requiere.

Sin embargo para poder realizar lo anterior fue necesario utilizar procesos y tuberías (pipes).

III.1.1 Procesos.

Un proceso es simplemente la ejecución de un programa. La utilización de estos procesos, entre otras cosas, es más destacable cuando al ejecutar un algoritmo o programa mandas llamar a un archivo ejecutable independiente. Ésta fue la manera en cómo se utilizaron los procesos para el desarrollo de GPU Math.

Es necesario mencionar que cada proceso provee el recurso necesario para ejecutar un programa, siendo más precisos, cada proceso tiene una dirección virtual, un código eje-

cutable, un contexto de seguridad, un único identificador del proceso, al menos un hilo de ejecución, etc.

Hilos de Ejecución.

Sin ahondar mucho en el tema, un hilo de ejecución es la entidad dentro del proceso que puede ser programada para la ejecución; como sabrán los que tengan algún conocimiento en programación, sólo es posible hacer que la computadora realice una acción a la vez, esto claro cuando no se utilizan herramientas especiales como los hilos. Por ello al utilizar un hilo de ejecución tenemos la posibilidad de mandar la orden para que se ejecute algún otro archivo ejecutable sin afectarlo y sin afectar el demás funcionamiento del programa. Los hilos también pueden tener su propio contexto de seguridad, el cual puede ser usado para imitadores de usuarios.

Además, Microsoft Windows soporta la **preemptive multitarea**, la cual da la impresión de una ejecución simultánea para múltiples hilos de múltiples procesos. En una computadora con varios procesadores, el sistema puede ejecutar simultáneamente tantos hilos como procesadores tenga la computadora.

Un **proceso hijo** es un proceso que es creado por otro proceso, el cual es llamado **proceso padre**, es decir, el proceso de donde se hace la llamada es el proceso padre y el programa ejecutable es el proceso hijo.

Para poder crear un proceso en algún algoritmo es posible utilizar la siguiente función:

Nota: Un manejador o *HANDLE* en inglés, es una estructura que contiene datos del objeto que se quiere manejar, en otras palabras es un apuntador que es controlado de manera independiente por el sistema.

A continuación se presenta un ejemplo de un proceso padre.

```
1  #include <windows.h>
2  #include <stdio.h>
3  int main(){
4     bool bFuncRetn = false;
5     // Especificamos la direccion donde se encuentra el proceso hijo
6     TCHAR direccion[ ] = TEXT(''C:''Users'''..''ProcesosHijo'');
7
8     // Preparamos la estructura STARTUPINFO para el proceso
9     STARTUPINFO si = { sizeof(si) };
10    SECURITY_ATTRIBUTES saProcess, saThread;
11    PROCESS_INFORMATION piProcessB;
12
13    // Preparamos el proceso B del proceso A.
14    // El manejador identificando el nuevo proceso que podra ser heredado
15    saProcess.nLength = sizeof(saProcess);
16    saProcess.lpSecurityDescriptor = NULL;
17    saProcess.bInheritHandle = TRUE;
18
19    // El manejador identificando al nuevo hilo
20    saThread.nLength = sizeof(saThread);
21    saThread.lpSecurityDescriptor = NULL;
22    saThread.bInheritHandle = FALSE;
23    si.dwFlags = STARTF_USEPOSITION;
24    bFuncRetn = CreateProcess(NULL, direccion, & saProcess, & saThread,
25       FALSE, 0, NULL, NULL, & si, & piProcessB);
26
27     if (!bFuncRetn){
28       printf("Error al crear el proceso \n");
29       return 0;
30     }
31     printf("Proceso creado");
32     printf("\n Llego al final de la funcion \n");
33     return 0;
34 }
```

Donde ProcesoHijo es un archivo ejecutable cualquiera.

BOOL WINAPI CreateProcess(LPCTSTR lpApplicationName, LPTSTR lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes, LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles, DWORD dwCreationFlags, LPVOID lpEnvironment, LPCTSTR lpCurrentDirectory, LPSTARTUPINFO lpStartupInfo, LPPROCESS_INFORMATION lpProcessInformation);
El comando anterior crea un nuevo proceso y su hilo primario.
lpApplicationName es el nombre del módulo a ser ejecutado; La cadena puede especificar la dirección completa y nombre del módulo a ser ejecutado; este parametro puede ser NULL, en cuyo caso significa que el módulo a ser ejecutado tiene la misma dirección del módulo que crea el proceso.
lpCommandLine representa la línea de comando a ser ejecutado; el parámetro puede ser NULL, en tal caso la función usa la cadena dada en lpApplicationName.
lpProcessAttributes representa un apuntador a una estructura SECURITY_ATTRIBUTES que determina si el manejador retornado por el nuevo proceso podrá ser heredado por el proceso hijo. Si lpProcessAttributes es NULL, el proceso obtiene un descriptor de seguridad por default.
lpThreadAttributes representa un apuntador a una estructura SECURITY_ATTRIBUTES que determina si el manejador retornado por el nuevo hilo podrá ser heredado por el proceso hijo. Si lpThreadAttributes es NULL, el manejador no podrá ser heredado.
bInheritHandles es un booleano, en caso de que éste parámetro sea TRUE, cada manejador heredado por parte del proceso que esta llamando es heredado por el nuevo proceso. En el caso de que el parámetro sea FALSE, los manejadores no son heredados.
dwCreationFlags representa las banderas que controlan la prioridad de la clase y la creación del proceso, para más detalles sobre los valores que puede tomar este parámetro ver **?**.
lpEnvironment es un apuntador a un bloque de entorno para el nuevo proceso. Si éste parámetro es NULL, el nuevo proceso usará el entorno del proceso que esta llamando.
lpCurrentDirectory es la dirección completa del directorio actual para el proceso. Si este parámetro es NULL, el nuevo proceso tendrá el mismo directorio y drive que el que esta llamando el proceso.
lpStartupInfo es un apuntador a una estructura STARTUPINFO o STARTUPINFOEX.
lpProcessInformation es un apuntador a una estructura PROCESS_INFORMATION que recibe la información de identificación acerca del nuevo proceso. Los manejadores en PROCESS_INFORMATION deben ser cerrados con la función CloseHandle cuando ya no se necesiten.
En caso de que la función sea exitosa regresará un valor diferente de cero, de lo contrario, si la función falla regresará el valor de cero.

Figura 3.1: Función CreateProcess.

Figura 3.2: Ejemplo de como llamar a un proceso hijo.

III.1.2 Tuberías (Pipes)

Ahora que ya sabemos como mandar una llamada para que se ejecute un programa desde otro programa llega a nosotros un pequeño problema, el cual es, como enviarle los datos de los argumentos de las funciones a los archivos ejecutables que realizaran dicha función.

La manera de solucionar dicho problema es con ayuda de tuberías. Las tuberias son un mecanismo que nos provee Microsoft Windows para facilitar la comunicación y el compartir datos entre aplicaciones. De manera general, las actividades que permiten éste tipo de mecanismos son llamados *comunicación entre procesos* o IPC por sus siglas en inglés.

De manera típica, las aplicaciones que usan IPC se clasifican en *clientes* y *servidores*. Un cliente es una aplicación o proceso que requiere un servicio de alguna otra aplicación o proceso, mientras que un servidor es una aplicación o proceso que atiende a la solicitud del cliente.

Existen dos tipos de tuberías para una comunicación de dos vías: tuberías anónimas y tuberías nombradas. Las **tuberías anónimas** permiten la transferencia de información entre procesos relacionados. Generalmente, las tuberías anónimas son utilizadas para redirigir información de entrada y salida de un proceso hijo para intercambiar datos con su proceso padre. Para que una tubería anónima intercambie información en ambas direcciones, se deben crear dos tuberías anónimas; el proceso padre escribe datos a una tubería usando su manejador de escritura, mientras que el proceso hijo lee los datos de ésa tubería con su manejador de lectura. De manera similar, el proceso hijo escribe datos en otra tubería y el proceso padre los lee de ésta.

NOTA: Las tuberías anónimas no pueden ser utilizados en una red de computadoras, ni en procesos no relacionados.

Las **tuberías nombradas** son utilizadas para transferir información entre procesos no relacionados y entre procesos de diferentes computadoras. Comúnmente, un proceso servidor de una tubería nombrada crea la tubería nombrada con un nombre conocido o con un nombre que deberá comunicar a su cliente. Una vez que tanto el servidor como el cliente se han conectado a la tubería, es posible intercambiar información ejecutando las operaciones de lectura y escritura en la tubería.

Como nota, hay que agregar que las tuberías anónimas utilizan menos espacio que las tuberías nombradas, sin embargo ofrecen servicios limitados.

Como se ha mencionado previamente, en el presente trabajo se utilizan procesos y es por eso que el tipo de tuberías usadas para la comunicación entre éstos seran las tuberías anónimas.

Tuberías anónimas

Las tuberías anónimas son tuberías de un solo sentido utilizadas para transferir información entre un proceso padre y un proceso hijo. Para crear una tubería anonima se utiliza el siguiente comando:

Un servidor puede enviar el manejador de escritura o el manejador de lectura, dependiendo de si se quiere utilizar a la tubería anónima para enviar o recibir información. Para leer información de la tubería, se utiliza el manejador de lectura cuando se llame a la función

BOOL WINAPI **CreatePipe**(PHANDLE *hReadPipe*, PHANDLE *hWritePipe*, LPSECURITY_ATTRIBUTES *lpPipeAttributes*, DWORD *nSize*);
hReadPipe es un apuntador a una variable que recibe el manejador de lectura de la tubería.
hWritePipe es un apuntador a una variable que recibe el manejador de escritura de la tubería
lpPipeAttributes es una apuntador a una estructura de tipo SECURITY_ATTRIBUTES que determina si el manejador retornado puede ser heredado por el proceso hijo. Si lpPipeAttributes es NULL, la tubería tendrá una descripción de seguridad por default.
nSize es el tamaño en bytes del buffer. Si éste parámetro es cero, el sistema usa el tamaño del buffer por default.
Si la función fue exitosa entonces regresará un valor diferente a cero, en caso contrario el valor regresado será cero.
Ésta función además crea dos manejadores, los cuales serán los manejadores de lectura y escritura de la tubería; el manejador de lectura tendrá acceso de sólo lectura a la tubería y el manejador de escritura sólo podrá escribir en la tubería, lo cual difiere con las tuberías nombradas. Para comunicarse utilizando la tubería, el servidor debe pasar un manejador de la tubería al proceso hijo; usualmente esto se hace a través del apuntador a la estructura STARTUPINFO en los parámetros requeridos para crear el proceso.

Figura 3.3: Función CreatePipe.

ReafFile; para escribir información en la tubería, se utiliza el manejador de escritura en la llamada a la función **WriteFile**. Dichas funciones se describen a continuación.

BOOL WINAPI **ReadFile**(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);
hFile es un manejador al archivo a ser leído.
lpBuffer es un apuntador al buffer que recibe los datos leídos del archivo
nNumberOfBytesToRead es el número máximo de bytes que pueden ser leídos
lpNumberOfBytesRead es un apuntador a una variable que recibe el número de bytes leídos
lpOverlapped es una apuntador a una estructura de tipo OVERLAPPED
Cuando la función fue ejecutada exitosa, ésta regresa un valor diferente de cero; en caso contrario regresará el valor de cero. Para conocer la causa del error se puede utilizar la función GetLastError.

Figura 3.4: Función ReadFile.

Previo a introducir un ejemplo de tuberías anónimas, se explicarán algunas funciones que son de utilidad al momento de crear tuberías.

La siguiente función recupera un manejador para el dispositivo estándar especificado(standard input, standard output o standard error)

BOOL WINAPI **WriteFile**(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);
hFile es un manejador al archivo a ser leído.
lpBuffer es un apuntador al buffer que contiene los datos que serán escritos al archivo
nNumberOfBytesToWrite es el número máximo de bytes que serán escritos al archivo
lpNumberOfBytesWritten es un apuntador a una variable que recibe el número de bytes escritos
lpOverlapped es una apuntador a una estructura de tipo OVERLAPPED
Cuando la función fue ejecutada exitosa, ésta regresa un valor diferente de cero; en caso contrario regresará el valor de cero. Para conocer la causa del error se puede utilizar la función GetLastError.

Figura 3.5: Función WriteFile.

HANDLE WINAPI **GetStdHandle**(DWORD nStdHandle);
nStdHandle representa el dispositivo estándar, este parámetro puede tomar uno de los siguientes valores.

Valor	Significado
STD_INPUT_HANDLE	El dispositivo estándar de entrada
STD_OUTPUT_HANDLE	El dispositivo estándar de salida
STD_ERROR_HANDLE	El dispositivo estándar de error

Figura 3.6: Función GetStdHandle.

Ahora que ya describimos todas las herramientas a utilizar podemos pasar a dar un ejemplo con el cual podremos entender con mayor facilidad éste tema. A continuación se presenta un ejemplo del proceso hijo, el cual primeramente recibe una frase para después enviar otra. Para después mostrar un ejemplo de un código que crea una tubería y que funciona como servidor para entablar comunicación con otro programa.

```
1  #include <windows.h>
2  #include <stdio.h>
3  #define BUFSIZE 4096
4
5  int main(){
6     CHAR chBuf[BUFSIZE];
7     DWORD dwRead, dwWritten;
8     HANDLE hStdin, hStdout;
9     BOOL fSuccess;
10    hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);
11    hStdin = GetStdHandle(STD\_INPUT\_HANDLE);
12    if ((hStdout == INVALID\_HANDLE\_VALUE) ||
13       (hStdin == INVALID\_HANDLE\_VALUE))
14       ExitProcess(1);
15    for (;;){
16
17   // leemos la frase de la tuberia
18
19 fSuccess = ReadFile(hStdin, chBuf, BUFSIZE, &dwRead, NULL);
20       if (! fSuccess || dwRead == 0)
21          break;
22
23       printf(" \nEl proceso hijo recibio el mensaje: %s", chBuf);
24    // escribimos una frase a la tuberia
25    strcpy(chBuf, "Mensaje del proceso hijo \n");
26       fSuccess = WriteFile(hStdout, chBuf, dwRead, &dwWritten, NULL);
27       if (! fSuccess)
28          break;
```

BOOL WINAPI **SetHandleInformation**(HANDLE hObject, DWORD dwMask, DWORD dwFlags);
Esta función fija ciertas propiedades de un manejador.
hObject es el manejador del objeto al cual se le va a establecer la información.
dwMask es la máscara que especifica la bandera que será cambiada. Dichas banderas son las mismas que se mostrarán para *dwFlags*.
dwFlags fija la bandera que especifica las propiedades del manejador del objeto. Este parámetro puede ser 0 ó uno de los siguientes valores.

Valor	Significado
HANDLE_FLAG_INHERIT	Si se fija ésta bandera, un proceso hijo creado con el parámetro *bInheritHandles* en **CreateProcess** como TRUE heredará el manejador
HANDLE_FLAG_PROTECT_FROM_CLOSE	Si se fija ésta bandera, cuando se llama a la función **CloseHandle**, ésta no cerrará el manejador

Si la función fue exitosa, entonces regresará un valor diferente de cero; por el contrario, si la función falla, ésta regresará el valor de cero

Figura 3.7: Función SetHandleInformation.

```
29          printf("El proceso hijo envio el mensaje: %s", chBuf);
30       }
31       return 0;
32    }
```

Figura 3.8: Ejemplo de una tubería anónima que también actua como cliente.

```
1     #include <windows.h>
2     #include <tchar.h>
3     #include <stdio.h>
4     #define BUFSIZE 4096
5
6     HANDLE hChildStdinRd, hChildStdinWr,
7        hChildStdoutRd, hChildStdoutWr,
8        hInputFile, hStdout;
9
10    BOOL CreateChildProcess();
11    void WriteToPipe();
12    void ReadFromPipe();
13    void ErrorExit(LPSTR);
14
15    int main()\{
16       SECURITY_ATTRIBUTES saAttr;
17       BOOL fSuccess;
18    // Fijamos al parametro {\em bInheritHandle} para que se puedan heredad los manejadores
19       saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
20       saAttr.bInheritHandle = TRUE;
21       saAttr.lpSecurityDescriptor = NULL;
22
23    // Fijamos el manejador al actual STDOUT. Notese que esto es para que podamos leer datos de la tuberia
24        hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
25
26    // Creamos la tuberia que nos servira para que el proceso hijo pueda leer y escribir datos
27       if (! CreatePipe(&hChildStdoutRd, &hChildStdoutWr, &saAttr, 0))
28          ErrorExit("Stdout pipe creation failed \n");
```

```
29
30  // Nos aseguramos que el manejador de lectura para el proceso hijo no sea heredado
31     SetHandleInformation( hChildStdoutRd, HANDLE_FLAG_INHERIT, 0);
32
33  // Creamos una tuberia para poderle enviar y recibir informacion del proceso hijo
34     if (! CreatePipe(&hChildStdinRd, &hChildStdinWr, &saAttr, 0))
35        ErrorExit("Stdin pipe creation failed \n");
36
37  // Nos aseguramos que el manejador de escritura para el proceso hijo no sea heredado
38     SetHandleInformation( hChildStdinWr, HANDLE_FLAG_INHERIT, 0);
39
40  // Creamos el proceso hijo
41     fSuccess = CreateChildProcess();
42     if (! fSuccess)
43        ErrorExit("Create process failed with");
44     WriteToPipe();
45     ReadFromPipe();
46     return 0;
47  }
48
49  BOOL CreateChildProcess(){
50     TCHAR szCmdline[ ]=TEXT("C:\\Users\\...\\AnonymousPipesChild");
51     PROCESS_INFORMATION piProcInfo;
52     STARTUPINFO siStartInfo;
53     BOOL bFuncRetn = FALSE;
54     ZeroMemory( &piProcInfo, sizeof(PROCESS_INFORMATION) );
55     ZeroMemory( &siStartInfo, sizeof(STARTUPINFO) );
56     siStartInfo.cb = sizeof(STARTUPINFO);
57
58  //Notese que es en estas lineas donde le mandamos los manejadores al proceso hijo
59     siStartInfo.hStdError = hChildStdoutWr;
60     siStartInfo.hStdOutput = hChildStdoutWr;
61     siStartInfo.hStdInput = hChildStdinRd;
62     siStartInfo.dwFlags |= STARTF_USESTDHANDLES;
63
64  // Creamos el proceso hijo
65     bFuncRetn = CreateProcess(NULL,
66        szCmdline,
67        NULL,
68        NULL,
69        TRUE,
70        0,
71        NULL,
72        NULL,
73        &siStartInfo,
74        &piProcInfo);
75
76     if (bFuncRetn == 0)
77        ErrorExit("CreateProcess failed \n");
78     else {
79        CloseHandle(piProcInfo.hProcess);
80        CloseHandle(piProcInfo.hThread);
81        return bFuncRetn;
82     }
83
84  }
85
86  void WriteToPipe() {
87     DWORD dwRead, dwWritten;
88     CHAR chBuf[BUFSIZE] = {"Mensaje del proceso padre"};
89  // Escribirmos en la tuberia
90
91     for (;;){
92       if (! WriteFile(hChildStdinWr, chBuf, dwRead,
93           &dwWritten, NULL)) break;
94        if (! ReadFile(hInputFile, chBuf, BUFSIZE, &dwRead, NULL) ||
95           dwRead == 0) break;
96    }
97
98  // Cerramos la tuberia para que el proceso hijo pare de leer
99     if (! CloseHandle(hChildStdinWr))
100       ErrorExit("Close pipe failed \n");
101 }
102 void ReadFromPipe() {
103    DWORD dwRead, dwWritten;
104    CHAR chBuf[BUFSIZE];
105
106 // Cerramos la terminal de escritura de la tuberia antes de leer los datos de la terminal de lectura
107 Close the write end of the pipe before reading from the
108 printf(" \n En ReadFromPipe \n");
109    if (!CloseHandle(hChildStdoutWr))
110       ErrorExit("Closing handle failed");
111
112 // Read output from the child process, and write to parent's STDOUT.
113    for (;;){
114       if( !ReadFile( hChildStdoutRd, chBuf, BUFSIZE, &dwRead,
115          NULL) || dwRead == 0) break;
116      printf(" \n El buffer de ReadFile dice %s", chBuf);
117       strcpy(chBuf, "Mensaje del proceso padre");
118       if (! WriteFile(hStdout, chBuf, dwRead, &dwWritten, NULL))
119          break;
120      printf(" \n El buffer de WriteFile dice %s", chBuf);
```

```
121     }
122  printf(" \n Salio de ReadFromPipe \n");
123  }
124
125  void ErrorExit (LPSTR lpszMessage) {
126     fprintf(stderr, "%s \n", lpszMessage);
127     ExitProcess(0);
128  }
```

Figura 3.9: Ejemplo de una tubería anónima que también actua como servidor.

III.2. Funciones implementadas

En matemáticas existen funciones que por su naturaleza (iterativa u otra) no pueden ser paralelizadas, un ejemplo de tales funciones es $f(n) = n-simonmerodelasucesindefibonacci$. La función $f(x)$ es iterativa, es decir, para obtener $f(n)$ se requiere haber hecho el cómputo para $f(n-1)$ y así sucesivamente. Es fácil ver que f(n) solo requiere de un procesador, y por lo tanto no es paralelizable.

Un ejemplo de una función paralelizable es la operación de suma de matrices (y vectores). En la figura (3.10) se puede observar que para cada A_{ij}es igual a $B_{ij} + C_{ij}$, es decir, la operación no depende de algún cómputo anterior, lo que se traduce en que cada A_{ij} puede calcularse por separado. Muchas operaciones importantes del Algebra Lineal son paralelizables, como son: el producto de matrices, la reducción gaussiana, entre otras.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} b_{11}+c_{11} & b_{12}+c_{12} & \cdots & b_{1n}+c_{1n} \\ b_{21}+c_{21} & b_{22}+c_{22} & \cdots & b_{2n}++c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1}+c_{n1} & b_{n2}++c_{n2} & \cdots & b_{nn}+c_{nn} \end{bmatrix} =$$
$$\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} + \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = B + C$$

Figura 3.10: Manera en que se realiza una suma de matrices.

Aunque la operación suma de matrices, descrita en el párrafo anterior es paralelizable, en la práctica hay muchas cosas a tomar en cuenta. Hay que notar que no se está trabajando diréctamente con el CPU, en realidad se está trabajando en la tarjeta gráfica de la computadora. La tarjeta gráfica no esta conectada directamente con la memoria principal, y como en esta se guardan todos los datos que ocupan los programas entonces el GPU no tiene acceso directo a los datos de las matrices B,C. Aunque existen muchas funciones para la comunicación entre el CPU y el GPU, siempre hay que tomar en cuenta que lo que hacen dichas funciones es trasladar cada byte de las matrices B y C desde la memoria principal hasta la memoria de video, y eso se traduce en una disminución de rendimiento. A continuación se explicará con más detalle.

Supóngase que un proceso P se puede realizar en n tareas independientes que llamaremos $tarea_i$, a su vez cada tarea va a requerir cierto número de operaciones ops_{tarea}, y cada operación va a requerir $tclock_{tarea,operacion}$ ciclos de reloj. Si solo se cuenta con un procesador de frecuencia f, el número de ciclos de reloj será:

$$totalclock = \sum_{i=1}^{n} \left(\sum_{j=1}^{ops_i} tclock_{i,j} \right)$$

y en el tiempo en segundos será:

$$t = \frac{totalclock}{frec}$$

En el ejemplo anterior solo se disponía de un solo procesador, por lo que estuvo muy sencillo saber en cuanto tiempo se iba a completar alguna tarea. Ahora supóngase que

se dispone de $k > 1$ procesadores, en adelante núcleos. Como cada $tarea_i$ es independiente, entonces el conjunto de tareas denotado por:

$$Tareas = \{tarea_1, tarea_2, ..., tarea_m\} \, donde \, m = min(k, n)$$

se pueden empezar a realizar al mismo tiempo.

La pregunta ahora es: ¿Cuanto tiempo va a requerir el procesador para ejecutar P?. A primera vista parece que la respuesta es: $\frac{t}{k}$, lo cual es muy buena aproximación si se cumple que: $tclock_{i,j} = tclock_{i+1,j} = tclock_{i,j+1}$. Obsérvese que existen dos respuestas posibles y estas dependen de k y n. Si $k >= m$ entonces todas las tareas van a ejecutarse en paralelo y el número total de ciclos de reloj para ejecutar la tarea es:

$$totalclock = max(\sum_{i=0}^{ops_1} tclock_{1,i}, \sum_{i=0}^{ops_2} tclock_{2,i}), .. \sum_{i=0}^{ops_n} tclock_{n,i}$$

y se puede ver que el tiempo para ejecutar P se dividió entre n. Para $m > k$ hay que tener en cuenta que el procesador no sabe cuanto tiempo requiere cada tarea. El procesador escoge las primeras k tareas, y si el i-esimo núcleo acabó primero que los demás, entonces la tarea $tarea_{k+1}$ se le asigna al i-esimo núcleo. Para obtener el número de iteraciones se requiere de una variable más y de un pseudocódigo para mayor claridad.

Sea $totalclock_i$ el número de ciclos de reloj que se requieren para que termine ciertas tareas el núcleo i. Primero, el procesador le va a asignar a cada núcleo una tarea, por lo anterior se infiere que $tarea_1, ..., tarea_k$ se empiezan a ejecutar al mismo tiempo. Cuando algún nucleo i termine de ejecutar $tarea_i$, entonces se le va a asignar $tarea_{k+1}$ para ejecutar, cuando $nucleo_j$ termine de ejecutar su tarea se le va a asignar $tarea_{k+2}$, y así sucesivamente.

El párrafo anterior es un bosquejo general de como opera un procesador con múltiples núcleos. Con dicho procedimiento se infiere que para calcular el número total de ciclos de reloj solo hay que acumular los tiempos de cada tarea al nucleo que tenga que realizar una menor cantidad de operaciones. Cundo el proceso de asignación termine solo hay que tomar el máximo de los tiempos (Ver figura):

Sea $totalclock_1 = 0, ..., totalclock_k = 0$

Para i=0 hasta n $min(totalclock_1, ..., totalclock_k) = min(totalclock_1, ..., totalclock_k) + tclock_i \;\; totalclock = max(totalclock_1, ..., totalclock_k)$;

Como ya se explicó en capítulos anteriores, la memoria de la tarjeta gráfica es compartida por todos los núcleos del GPU. Funciona casi de la misma manera que los núcleos de un CPU. Lo anterior quiere decir que para modelar un proceso en el cual cada núcleo necesita mucho acceso a memoria, este debe tratarse de forma independiente. A continuación se darán algunos algoritmos de funciones paralelizables.

III.3. Funciones implementadas de Álgebra Lineal

III.3.1 Producto de Matrices

La multiplicación de matrices es una operación muy usada, tanto en el álgebra lineal como en las ecuaciones diferenciales, etc. Al igual que en la integración numérica, para la multiplicación de matrices existen varios algoritmos.

Sean las matrices reales: A_{mxn}, B_{pxm} y $C_{pxn} = A \cdot B$. Como $C = A \cdot B$, entonces:

$$C_{i,j} = \sum_{k=0}^{m} A(k,j) \cdot B(i,k)$$

La operación anterior puede paralelizarse si se le asigna a cada núcleo del GPU la operación $C_{i,j}$.

Algo muy importante del cómputo de $C_{i,j}$ es que no podemos saber exactamente su tiempo de ejecución, depende de cada chip GPU. Hay que notar que las matrices A y B se guardan en la memoria de la tarjeta gráfica, de tal manera que cada núcleo comporte los elementos $\{a_{i,j}\}$ y $\{b_{i,j}\}$. La razón por la que no podemos determinar con exactitud el tiempo de ejecución del algoritmo es por que los núcleos que hacen los cómputos $\{C(1,j), ..., C(p,j)\}$ dependen al mismo tiempo del valor $A(1,j)$, y como ya se explicó anteriormente, dos o más núcleos no pueden acceder a un mismo recurso de memoria a la vez. Lo anterior hace parecer al producto de matrices una operación secuencial: si el núcleo $C(1,j)$ está ocupando $A(1,j)$, entonces el núcleo $C(2,j)$ tendrá que esperar. Pero no hay de que preocuparse, los resultados muestran que de todos modos sigue siendo óptimo hacer el producto de matrices en el GPU.

Si t_1 es el tiempo requerido para hacer el producto entre dos números flotantes y t_2 para la suma de dos números flotantes, entonces, para hacer el producto de matrices en el CPU el tiempo requerido es:

$$\begin{aligned} &(m \cdot t_1 + (m-1) \cdot t_2) \cdot p \cdot n \\ &= m \cdot p \cdot n \cdot \left(t_1 + \left(1 - \frac{1}{m} \right) \cdot t_2 \right) \end{aligned}$$

el cómputo (teórico) de la misma operación en el GPU es:

$$\frac{m \cdot p \cdot n \cdot \left(t_1 + \left(1 - \frac{1}{m} \right) \cdot t_2 \right)}{numnucleos}$$

donde $numnucleos$ es la cantidad de núcleos que tiene el GPU.

III.4. Funciones implementadas de Cálculo

III.4.1 Integración numérica

Existen muchos algoritmos para la integración numérica de varias variables. La idea de casi todos los algoritmos de integración es hacer una partición del dominio en regiones $R_1, ...R_n$ y después hacer algún cómputo para cada R_k.

Sumas de Riemann:

Supóngase que se quiere integrar la función $f(x) = a_n \cdot x^n + ... + a_1 \cdot x + a_0 \cdot$ en el intervalo $[a,b]$ donde a_k son constantes conocidas. Una forma muy sencilla de integrar $f(x)$ es haciendo la siguiente suma:

$$\int_a^b \approx \sum_{k=1}^{N} f(a + k \cdot \frac{b-a}{n}) \cdot \frac{b-a}{N}$$

Como ya se explicó antes, la suma no es paralelizable en el GPU(por razones de optimalidad), así que no hay que fijarse en la suma. Lo que si se puede paralelizar (en muchos

casos) es la evaluación de la función $f(x)$. Si $f(x)$ es una función muy complicada, entonces se le puede asignar a cada núcleo del GPU la operación $f(a + k \cdot \frac{b-a}{n}) \cdot \frac{b-a}{N}$ para que después el CPU haga la suma y arroje un resultado.

La explicación anterior nos da una aplicación mas, a saber: "la evaluación de funciones". Es claro ver que solo es aplicable cuando se requiere evaluar en muchos puntos a la vez.

III.4.2 Series de Fourier

Las series de Fourier[1] son utilizadas para el análisis de Fourier, el cual trata de la extracción de información de señales convertidas en una serie de números.

La series de Fourier, como su nombre lo indica, consta de una seria, la cual puede aproximar (converger puntualmente[2]) a una gran cantidad de funciones que cumplan la hipótesis de ser continuas a trozos.

La idea detrás de las series de Fourier es la aproximación de la función con ayuda de funciones cosenoidales, es decir data una función $f(x)$, aproximar esta a través de la serie mostrada en la ecuación 3.1.

$$\frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n cos\left(\frac{n * \pi * x}{b-a}\right) + b_n sin\left(\frac{n * \pi * x}{b-a}\right) \right] \tag{3.1}$$

Como se puede apreciar en la ecuación 3.1 los valores a encontrar son los a_i y los b_i, ya que los valores n, a, b y x son provistos por la ecuación y por la información referente a esta.

Recordando que los valores a_i y b_i se obtienen a partir de integrales, estas integrales se muestran en las ecuaciones 3.2, 3.3 y 3.4.

$$a_0 = \frac{2}{b-a} \int_0^{b-a} f(x)dx \tag{3.2}$$

$$a_n = \frac{2}{b-a} \int_0^{b-a} cos\left(\frac{n\pi x}{b-a}\right) f(x)dx \tag{3.3}$$

$$b_n = \frac{2}{b-a} \int_0^{b-a} sin\left(\frac{n\pi x}{b-a}\right) f(x)dx \tag{3.4}$$

Donde el valor n en las ecuaciones 3.3 y 3.4 va desde 1 hasta el límite superior de la suma.

Como se menciono previamente, las series de Fourier son utilizadas para el análisis de señales, además de esto, también son utilizadas fuertemente para el reforzamiento de señales y por supuesto para dar soluciones a algunas ecuaciones diferenciales parciales.

Como puede observar en las ecuaciones anteriores, la paralelización en la GPU de los cálculos para obtener los coeficientes de Fourier se debe principalmente a las integrales que se utilizan.

[1] Para información detallada acerca de las series de Fourier, véase Weinberger

[2] Para la definición de convergencia puntual, véase Spivak

III.5. Procesamiento de Imágenes

Una imagen esta constituida por una arreglo rectangular de pixeles. Cada pixel tiene tres valores (enteros) de 8 bits, correspondientes a las identidades de rojo, verde y azul (RGB por sus siglas en inglés); de esta manera se pueden generar poco mas de 16M de colores.

En la práctica es muy difícil trabajar con las tres componentes de cada pixel. El estudio de imágenes tomando en cuenta las tres componentes RGB es muy reciente y aún está en fase teórica. Por lo anterior se vuelve necesario simplificar la imagen, para ello se transforma a su respectiva escala de grises, con lo cual nos queda una imagen cuyas componentes RGB en cada pixel poseen el mismo valor L para al menos dos componentes. Así es como se obtiene una función del tipo: $f(x, y) = L_{x,y}$ donde (x, y) es la posición del pixel y $L_{x,y}$ es su respectivo valor.

Naturalmente un borde es aquella región donde el pixel (x, y) tiene un valor $L_{x,y}$ muy diferente a sus vecinos $(x + 1.y + 1), (x - 1, y + 1), ...$ Pero con la imagen vista como una función de $R^2 \rightarrow R$ el cálculo de un borde se reduce a encontrar la derivada de $L_{x,y}$.

III.5.1 Detección de bordes

Como se acaba de mencionar un borde es la región donde el valor $L_{x,y}$ del pixel del borde es diferente a los que se encuentran a su alrededor, para encontrar dichos pixeles se implementan diferentes operadores, algunos de estos se mencionan a continuación:

Sobel

El operador Sobel es un operador diferencial discreto que calcula el gradiente de intensidad de una imagen, es decir, el vector de derivadas en cada pixel de la imagen, el cual indica la dirección en el que la intensidad varía con mayor velocidad; esto muestra que tan suavemente cambia la intensidad de una imagen y por ello la probabilidad de que exista un borde.

El operador Sobel al igual que otros operadores utiliza dos matrices también llamadas máscaras para aplicar una convolución a la imagen y de esta manera calcular aproximaciones a las derivadas, la primer máscara es para detectar los cambios horizontales y la segunda para los cambios verticales, dichas máscaras se muestran a continuación:

$$M_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Figura 3.11: Máscara Sobel para los cambios horizontales.

$$M_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Figura 3.12: Máscara Sobel para los cambios verticales.

De esta manera, sí **A** representa la parte de la imagen original de la cual se quiere conocer la intensidad, entonces las imagenes que representan las aproximaciones a las derivadas

horizontales y verticales vienen dadas por las siguientes expresiones:

$$G_x = M_x * A \quad y \quad G_y = M_y * A \tag{3.5}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.6}$$

Detallando la explicación anterior podemos decir, sí **A** es como se muestra en la imagen 3.13, es decir **A** representa un cuadro de pixeles de la imagen a la cual se le desea aplicar el proceso de detección de bordes. Al multiplicar las máscaras por **A** y obtener el valor de G obtendremos el gradiente de intensidad del cuadro coloreado de gris en la imagen 3.13.

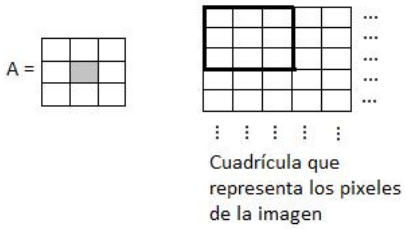

Figura 3.13: Ejemplo de la obtención de intensidad luminosa de un pixel.

PREWITT

En la actualidad existe una amplia gama de operadores con los cuales es posible detectar los bordes en una imagen, uno de ellos es el operador Sobel del cual se acaba de mencionar; otro operador muy útil para realizar esta labor es el operador **Prewitt**.

La diferencia entre los diferentes operadores radica en las máscaras que se usan, el operador Prewitt utiliza las siguientes máscaras.

$$M_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

Figura 3.14: Máscara Prewitt para los cambios horizontales.

$$M_y = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

Figura 3.15: Máscara Prewitt para los cambios verticales.

$$M_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$$

Figura 3.16: Máscara Roberts para los cambios horizontales.

$$M_y = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Figura 3.17: Máscara Roberts para los cambios verticales.

ROBERTS

El operador **Roberts** por otra parte tiene las máscaras mostradas en las imágenes 3.16 y 3.17.

En las imágenes 4.6 y 4.7 se muestran los resultados de aplicar los operadores Sobel y Prewits respectivamente a la imagen 4.4. Con el fin de hacer un poco más entendible de la razón de ser de la ecuación 3.6 se presentan en las imágenes 3.18, 3.19, 3.20 y 3.20 las máscaras G_x y G_y de los operadores Sobel y Prewits.

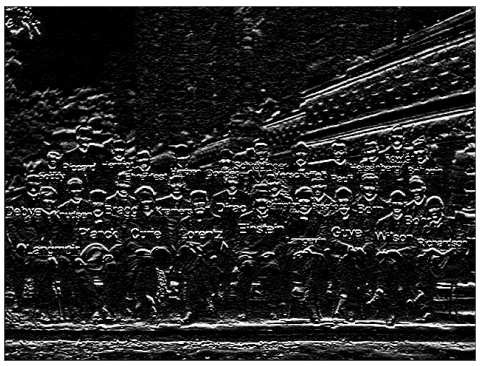

Figura 3.18: Máscara G_x del operador Sobel aplicado a la imagen 4.4.

III.5.2 Filtros - Canny

El operador canny[3], al igual que los operadores antes mencionados, es un operador de detección de bordes, el cual tiene ventajas adicionales a los operadores antes mencionados.

Una de estas ventajas es que es menos susceptible al ruido que se encuentra en una imagen, es decir a líneas que no perteneces a la imagen y que fueron capturados por el tipo de cámara con que se captó la imagen o por algún otro factor.

El operador Canny usa un algoritmo de varias etapas para detectar los bordes contenidos en la imagen. Cabe destacar que el señor John Canny además de crear el algoritmo

[3]Debido al computólogo John F. Canny, el cual contribuyó además en áreas como la inteligencia artificial, la robótica, la seguridad computacional, computación algebraica, geometría computacional, entre otras.

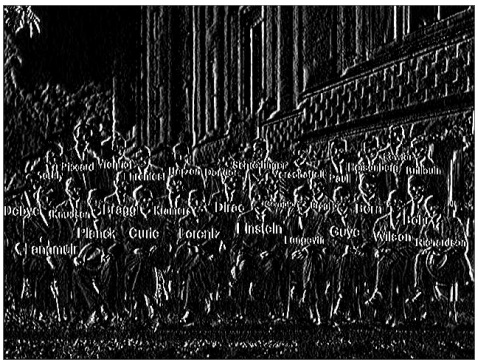

Figura 3.19: Máscara G_y del operador Sobel aplicado a la imagen 4.4.

Figura 3.20: Máscara G_x del operador Prewitts aplicado a la imagen 4.4.

que lleva su nombre creó, una teoría computacional de detección de borde donde explica el por qué es que su algorimo es el más óptimo.

A grandes rasgos el algoritmo Canny está dividido en dos grandes etapas, la primera de estas constituye en un filtro para disminuir el ruido, este filtro está basado en una Gaussiana, esto quiere decir que una porción de la imagen es convolucionada, multiplicada, por dicho filtro, de la misma manera que se hace con los operadores antes mencionados. El resultado de aplicar la imagen a este filtro es una versión de la imagen más borrosa, por lo cual los bordes en la imagen sobresalen más y el ruido existente en la imagen disminuye.

Un ejemplo de un filtro para disminuir el ruido se presente en la imagen 3.22, donde la desviación estandar (σ) es 1.4.

La manera de obtener la forma del filtro es a través de la función Gaussiana bidimensional, la cual tiene la forma de la ecuación 3.7.

$$f(x,y) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{x^2+y2}{2\sigma^2}\right) \tag{3.7}$$

Es decir, el valor a_{ij} de la matriz filtro se obtiene evaluando el punto $i-3$, $j-3$ en la ecuación 3.7, es decir $a_{ij} = f(i-3, j-3)$.

La segunda etapa constituye en aplicar un segundo filtro de deteccion de bordes a la imagen, este segudo filtro puede ser alguno de los que se mencionaron anterioremente, es decir, Soble, Prewits o Roberts.

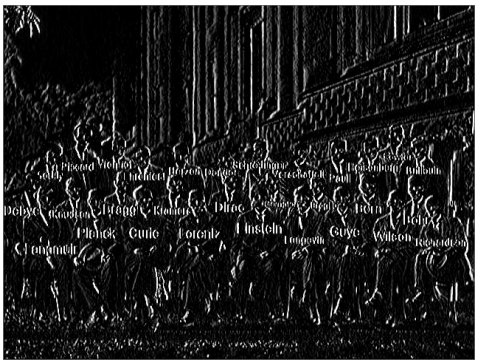

Figura 3.21: Máscara G_y del operador Prewitts aplicado a la imagen 4.4.

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figura 3.22: Máscara para disminuir el ruido en el operador Canny.

III.5.3 Transformada Hough para círculos y líneas

La transformada de Hough es un algoritmo utilizado en la detección de patrones de imágenes, el cual permite la detección de diversas formas que se encuentran dentro de una imagen.

Debido a la gran cantidad de información que contiene una imagen convertir la imagen a escala de grises y aplicarle algún método de procesamiento de imágenes para detectar todos los bordes que contenga dicha imagen.

Cabe resaltar que estas operaciones son necesarias antes de poder aplicar el algoritmo que corresponde a la transformada de Hough.

Es necesario destacar que el algoritmo de Hough es tan general que puede aplicarse para detectar cualquier tipo de forma dentro de una imagen, tal como un corazón una fruta o cualquier figura tan complicada como se quiera; sin embargo, para poder detectar cada forma es necesario modificar el algoritmo para que este pueda detectar la forma deseada. En la presente tesis se incorporaron dos códigos que implementan el algoritmo de Hough para detectar dos formas básicas: círculos y líneas.

A continuación se describirá la manera de aplicar el algoritmo de la transformada de Hough para de detectar líneas a través de un ejemplo.

Consideremos la ecuación de la recta $ax + by = c$; lo primero que se necesita es tener la ecuación de la línea parametrizada, la forma paramétrica de la recta, como es sabido, es expresada por la ecuación 3.8.

$$xcos(\theta) + ysin(\theta) = \rho \tag{3.8}$$

La manera en como se detectarán los lugares por donde pasan las líneas es a través de una tabla que se llenará con unos que se irán acumulando por diferentes criterios, así, los valores más grandes indicarán los lugares por donde pasan las rectas.

Dicha tabla tendrá a sus lados los valores discretizados de los parámetros de la ecuaciones, es decir de θ y de ρ, tal y como se muestra a continuación en la imagen 3.23.

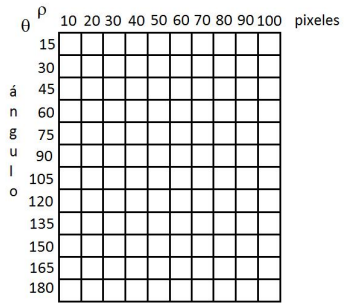

Figura 3.23: Ejemplo de una tabla con los parámetros ρ y θ discretizados.

En la imagen 3.23 se puede observar la discretización que se utilizó como ejemplo, cabe resaltar que la discretización sobre los pixeles puede variar dependiendo del tamaño de la imagen; sin embargo la discretización sobre el ángulo, a pesar de que también puede variar el número de valores sobre la discretización, sólo es necesario discretizar los ángulos de 0 a 180.

La manera de llenar la tabla es considerando cada pixel ocupado; recordemos que para este momento ya se convirtió la imagen a escala de grises y se le aplicó una máscara para detectar los bordes, por lo que un pixel es considerado .ºcupadoçuando un borde pasa sobre este.

Para una explicación con mayor detalle describiremos el proceso con ayuda de la imagen 3.24, la cual supondremos que es una imagen la cual ya fue convertida a escala de grises y tratada para detectar los bordes en ella.

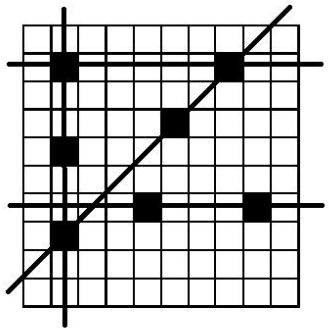

Figura 3.24: Ejemplo de una imagen convertida a escala de grises y con una máscara aplicada para detectar los bordes en ella.

Supondremos entonces que la imagen 3.24 es una imagen de 10x10 pixeles, donde cada recuadro representa un pixel y los recuadros de color negro son aquellos que están

ocupados, lo que se describirá a continuación es la manera de como llenar la tabla que tendrá una forma semejante a la de la imagen 3.23 para detectar las diferentes líneas que se pueden encontrar en la imagen 3.24.

Tomando a cada pixel de forma individual, se fijará un ángulo entre 0 y 180, fijando así el valor θ en la ecuación 3.8, el siguiente paso a realizar es obtener el valor ρ donde los valores x y y es la posición del pixel. Así para la imagen 3.24 la tabla con los parámetros ρ y θ queda como en la imagen 3.25.

distancia en pixeles

ángulo θ \ ρ	1	2	3	4	5	6	7	8	9	10
0			1	2		1	1		2	
45	4				1	1		1		
90	3				1	1		1	1	
135				1			1	1		1

Figura 3.25: Ejemplo de una tabla llena con los valores de la imagen 3.24.

Como se puede apreciar en la imagen 3.25, existen cuatro valores mayores a uno, los cuales corresponden a las líneas resaltadas sobre la imagen 3.24; cabe resaltar que no todo valor mayor a uno representa una línea. Además es necesario mencionar que el hecho de que la tabla a llenar correspondiera a una bidimensional es debido a que el número de parámetros que posee una línea es dos; por otro lado el número de parámetros que posee un círculo es tres, esto debido a que la ecuación paramétrica de un círculo es: $(x - x_o)^2 + (y - y_0)^2 = r^2$ y por ello la forma que adquiriría la tabla es un poco mas compleja, ya que esta tabla tendría en sus lados los dos valores del centro del círculo y el radio de este, fuera de este detalle la manera de llenar la tabla y por tanto de encontrar los posibles círculos en una imagen es la misma.

IV. RESULTADOS

En el presente capítulo se exhibirán los beneficios y ventajas obtenidos al utilizar GPU Math, además de algunos ejemplos que ayudarán a entender la forma en que se pueden emplear las funciones que se realizaron en esta biblioteca.

IV.1. Mejoras a las funciones implementadas

Presentamos las mejoras obtenidas de las funciones con el mismo orden que en el anterior capítulo.

IV.2. Mejoras a las funciones de Álgebra Lineal

En el álgebra lineal los tipos de datos más básicos con los que se trabaja son: matrices, vectores y escalares; por ello, se definen en GPU Math dichos tipos de datos.

Detallando la manera en como se definen estos datos, cabe mencionar que cada uno de estos datos se crearon con ayuda de estructuras (struct [1]), las cuales contienen como componentes los tres valores que se consideran necesarios para poder definirlos. Dos de estos valores necesitan ser enteros (int) ya que serán los que definan las dimensiones de estos tipos de datos, mientras que el tercer valor dará a cada tipo la capacidad de ser predefinido con un valor constante flotante (float) si así se requiere, de otro modo podrá definirse como nulo (NULL) para que se definan los valores de cada tipo de dato de forma manual.

Como es bien sabido, en C/C++ es necesario definir cada variable antes de poder usarse, del mismo modo es necesario definir cada vector o matriz antes de que esta sea utilizada.

Para poder definir una matriz se utiliza el comando **mathgpumy_MatrixRf** seguido por el nombre de la variable que se desee, esta variable debe ser un apuntador ya que la manera en como se devuelven los valores y los resultados de las operaciones es a través de la técnica de programación conocida como pase por referencia (struct [2]); un ejemplo de esto se muestra en la siguiente línea.

mathgpumy_MatrixRf *A;

Para poder definir un vector se utiliza el comando **mathgpumy_VectorRf** seguido por el nombre de la variable, esta variable debe ser un apuntador ya que la manera en como se reserven los valores de entrada del vector y los resultados de las operaciones es, al igual que con las matrices, con ayuda de la técnica de programación llamada: pase por referencia; un ejemplo de esto se muestra en la siguiente línea.

mathgpumy_VectorRf *x;

Para crear una matriz A de dimensión $m * n$ se utiliza el comando **mathgpumy_CreateMatrixRf1**. Dicho comando requiere de tres parámetros, los dos primeros

[1] para más información acerca de este tipo de dato vease (C++ cómo programar. DEITEL)

[2] para más información acerca de este tipo de dato véase (C++ cómo programar. DEITEL)

representan las dimensiones de la matriz que se crea, el primer parámetro m representa el número de renglones que tendrá la matriz A mientras que el segundo parámetro representa el número de columnas que tendrá la matriz A, cabe mencionar que las dimensiones de la matriz A deben ser enteras, para lo cual se utiliza el tipo de dato int; el tercer parámetro nos brinda la facilidad de poder definir matrices constantes, esto es, matrices cuyas entradas (a_{ij}) son iguales para todo i, j y por ello el tipo de dato para este tercer parámetro es un flotante (float), si no se desea definir una matriz constante es posible dar este parámetro como nulo (NULL). A continuación se muestra una manera de poder crear una matriz de manera general.

mathgpumy_CreateMatrixRf1(int m,int n,float *vals)

Para crear un vector se requiere hacer una distinción, ya que existen dos tipos de vectores: vectores renglón y vectores columna. Para crear un vector columna es necesario utilizar el comando **mathgpumy_CreateColumnVectorRf** mientras que para crear un vector renglón es necesario utilizar el comando **mathgpumy_CreateRowVectorRf**. Estos comandos son parecidos en la manera en como se definen ya que ambos requieren de dos parámetros de entrada, el primero de ellos nos brinda la misma facilidad que el tercer parámetro necesario para definir una matriz, es decir podemos definir un vector constante; el segundo parámetro representa el número de elementos que tendrán los vectores por lo cual el tipo de dato necesario para este parámetro es un entero (int). A continuación se muestra una manera de definir un vector columna y un vector renglón de manera general.

mathgpumy_CreateColumnVectorRf(float *data,int size)

mathgpumy_CreateRowVectorRf(float *data,int size)

Para crear un escalar se hace uso de un tipo de dato común en el lenguaje C, el cual es el flotante (float).

A continuación se presenta un ejemplo donde se aplica el uso de GPU Math con su aplicación de Álgebra Lineal para realizar el producto entre una matriz y un vector, además de la descripción de los comandos que se presentan con la finalidad de que resulten ser una guia útil para aquel que desee aplicar la biblioteca.

```
1  //linkers libmathgpu_LA.dll.a
2  #include <stdio.h>
3  #include <conio.h>
4  #include <C:\Users\...\mathgl\mathGLmy\Linear Algebra\dlls\mathgpu_LA\main.h>
5
6  int main(){
7      mathgpumy_MatrixRf *A;
8      mathgpumy_VectorRf *x,*y;
9      int i1,i2;
10
11     A = mathgpumy_CreateMatrixRf1(1000,10,NULL);
12     x = mathgpumy_CreateColumnVectorRf(NULL,1000);
13     y = mathgpumy_CreateColumnVectorRf(NULL,10);
14
15     for(i1=0;i1<A->n_;i1++){
16         for(i2=0;i2<A->m_;i2++){
17             *(A->vals_+(A->m_)*i1+i2)=1;
18         }
19     }
20     for(i1=0;i1<x->n_;i1++){
21         *(x->vals_+i1)=1;
22     }
23     mathgpumy_MatrixRfVectorRf_Ax(y,A,x);
24
25     for(i1=0;i1<y->n_;i1++){
26         printf("\n  %f",*(y->vals_+i1));
27     }
28     system("pause");
29     return 1;
30 }
```

El algoritmo anterior realiza el producto entre una matriz A y un vector x, donde el resultado obtenido por el uso de GPU Math es guardado en el vector y. Es en las líneas

7 y 8 donde se definen la matriz A y los vectores columna x y y respectivamente a través de los tipos de datos **mathgpumy_CreateMatrixRf1** y **mathgpumy_CreateColumnVectorRf**, los cuales son creados por la biblioteca GPU Math.

En la línea 11 se definen las características de la matriz A a través del comando **mathgpumy_CreateMatrixRf1**; donde el primero parámetro representa el número de columnas que tendrá dicha matriz, el segundo parámetro representa el número de renglones que tendrá dicha matriz, mientras que el tercer parámetro representan los datos iniciales con los que se puede llenar la matriz, en este caso, como a la matriz se le dan los valores en las líneas 15-19 se declara dicho parámetro como nulo (NULL).

En las líneas 12 y 13 se definen las características de los vectores x y y a través del comando **mathgpumy_CreateColumnVectorRf**; donde el primer parámetro representa el vector de datos de dichos vectores y puesto que se realiza el llenado de los vectores en las líneas 20 a 22, se declara dicho parámetro como nulo (NULL), mientras que el segundo parámetro representa el número de renglones que tendrán dichos vectores.

En las líneas 15 a 22 se definen los valores de la matriz A y del vector x, mientras que en la línea 23 se realiza el producto entre la matriz A y el vector x y el resultado se almacena en el vector y a través del comando **mathgpumy_MatrixRfVectorRf_Ax**.

Un vector, como es bien sabido, puede ser de dos tipos: vector columna o vector renglón; en el ejemplo anterior se utilizaron y definieron vectores columna con ayuda del comando que se describe en la línea 12, sin embargo en GPU Math es tambien posible definir un vector renglón, esto con ayuda del comando **mathgpumy_CreateRowVectorRf**, el cual requiere de dos parámetros al igual que el comando **mathgpumy_CreateColumnVectorRf** el primero de los cuales permite introducir los valores que tendrá dicho vector y el segundo de los mismos definirá el número de elementos que tendrá el vector columna.

IV.2.1 Producto de matrices y otras operaciones

En el tema de álgebra lineal GPU Math contiene varias operaciones, una de las cuales es el producto de matrices, como se acaba de ver en el ejemplo anterior. A continuación se hace un listado de las operaciones incorporadas a GPU Math seguidas por los comandos que realizan dichas operaciones y los argumentos necesarios por tales comandos.

Transpuesta de un vector - Esta operación se hace a través del comando **mathgpumy_VectorRf_transpose** el cual solo requiere como argumento el nombre de dicho vector, es decir, el argumento del comando es un tipo de dato mathgpumy_VectorRf. El vector transpuesto se almacenará en el dato enviado.

mathgpumy_VectorRf_transpose(mathgpumy_VectorRf *x)

Suma de vectores - Esta operación se hace a través del comando **mathgpumy_VectorRf_sum** el cual requiere de dos vectores, es decir de dos argumentos de tipo **mathgpumy_VectorRf** y el resultado de dicha suma se almacenará en el primer vector enviado.

mathgpumy_VectorRf_sum(mathgpumy_VectorRf *vec1,mathgpumy_VectorRf *vec2)

Producto de un escalar con un vector - Esta operación se hace a través del comando **mathgpumy_VectorRf_ax** y requiere de dos argumentos, el primero de ellos un escalar, tipo de dato float, y el segundo argumento un vector, tipo de dato (mathgpumy_VectorRf); el resultado de la operación se almacena en el vector enviado.

mathgpumy_VectorRf_ax(float a,mathgpumy_VectorRf *x)

Suma y producto de un escalar y dos vectores - Esta operación se hace a través del comando **mathgpumy_VectorRf_axpy** y requiere de un escalar (float) y dos vectores (mathgpumy_VectorRf) y el resultado de esta operación es primeramente el producto del escalar con el primer vector cuyo resultado es sumado al segundo vector, el resultado de estas operaciones es almacenada en el parámetro que ocupe el primer vector. Esta operación puede verse como una forma resumida entre los comandos mathgpumy_VectorRf_ax y mathgpumy_VectorRf_sum.

mathgpumy_VectorRf_axpy(float a,mathgpumy_VectorRf *x,mathgpumy_VectorRf *y)

Copia de un vector - Esta operación se hace a través del comando **mathgpumy_VectorRf_copy** y el resultado de dicho comando es el copiado de un vector, por ello se requiere de dos parámetros de entrada, los cuales son vectores (mathgpumy_VectorRf), el primero de los parámetros será el vector donde se almacene la copia y el segundo parámetro será el vector a copiar.

mathgpumy_VectorRf_copy(mathgpumy_VectorRf *dst,mathgpumy_VectorRf *src)

Intercambio de vectores - Esta operación se hace a través del comando **mathgpumy_VectorRf_swap** por lo que requiere de dos parámetros de entradas, los cuales son vectores (mathgpumy_VectorRf).

mathgpumy_VectorRf_swap(mathgpumy_VectorRf *vec1,mathgpumy_VectorRf *vec2)

Producto interior de vectores - Esta operación se hace a través del comando **mathgpumy_VectorRf_dot** y requiere de dos vectores (mathgpumy_VectorRf), el resultado de dicha operación, como se sabe, es un número real (tipo de dato float); por lo que para obtener el resultado de esta operación es necesario igualarlo a un número real.

a = mathgpumy_VectorRf_dot(mathgpumy_VectorRf *x,mathgpumy_VectorRf *y)

Norma de un vector - Esta operación se hace a través del comando **mathgpumy_VectorRf_nrm2**, dicho comando requiere de un parámetro, el cual es un vector (mathgpumy_VectorRf), el resultado de dicha operación, como es sabido, es un número real positivo. Para obtener el resultado de esta operación es necesario igualarlo a un número real (tipo de dato flotante).

a = mathgpumy_VectorRf_nrm2(mathgpumy_VectorRf *x)

Suma de matrices - Esta operación se hace a través del comando **mathgpumy_MatrixRf_sum**, dicho comando requiere de dos parámetros que serán las matrices (tipos de datos mathgpumy_MatrixRf) a sumar y cuyo resultado se almacenará en el primer parámetro introducido.

mathgpumy_MatrixRf_sum(mathgpumy_MatrixRf *A,mathgpumy_MatrixRf *B)

Producto de un escalar y una matriz - Esta operación se hace a través del comando **mathgpumy_MatrixRf_aA**, dicho comando requiere de dos parámetros, el primero de los cuales es el escalar a multiplicar (float), el segundo parámetro corresponde a la matriz (tipo de dato mathgpumy_MatrixRf); el resultado del producto se almacenará en la matriz enviada.

mathgpumy_MatrixRf_aA(float a,mathgpumy_MatrixRf *A)

Copia de una matriz - Esta operación se hace a través del comando **mathgpumy_MatrixRf_copy**, dicho comando requiere de dos parámetros, el primero parámetro representa la matriz (tipo de dato mathgpumy_MatrixRf) en donde se copiará la matriz, mientras que el segundo parámetro representa la matriz a copiar.

mathgpumy_MatrixRf_copy(mathgpumy_MatrixRf *dst,mathgpumy_MatrixRf *src)

Transpuesta de una matriz - Esta operación se hace a través del comando **mathgpumy_MatrixRf_transpose**, dicho comando requiere de un parámetro el cual es la matriz (tipo de dato mathgpumy_MatrixRf) que se quiere transponer, el resultado de la operación se almacena en la misma matriz que se introduce como parámetro.

mathgpumy_MatrixRf_transpose(mathgpumy_MatrixRf *A)

Producto entre matrices - Esta operación se hace a través del comando **mathgpumy_MatrixRf_AB**, tal operación requiere de dos parámetros los cuales son las matrices (tipos de datos mathgpumy_MatrixRf) a multiplicar, el resultado se almacena en el primer parámetro introducido.

mathgpumy_MatrixRf_AB(mathgpumy_MatrixRf *A,mathgpumy_MatrixRf *B)

Producto entre un vector y una matriz - Esta operación se hace a través del comando **mathgpumy_MatrixRfVectorRf_Ax**, dicho comando requiere de tres parámetros, el segundo parámetro, llamémoslo A, representa la matriz (mathgpumy_MatrixRf) a multiplicar mientras que el tercer parámetro, llamémosle x, representa el vector columna (mathgpumy_VectorRf) que participa en el producto, el primer parámetro será el vector columna (mathgpumy) que almacenará el resultado del producto Ax.

mathgpumy_MatrixRfVectorRf_Ax(mathgpumy_VectorRf *y,mathgpumy_MatrixRf *A,mathgpumy_VectorRf *x)

A continuación se muestra el shader que implementa el producto de funciones, eñ cual fue implementada en el motor de álgebra lineal.

```
uniform sampler2DRect matrizA;
uniform sampler2DRect matrizB;
uniform sampler2DRect matdim;
uniform sampler2DRect matguiax;
uniform sampler2DRect matguiay;

float elmAxy(float x,float y){
    float px,px2;
    px=floor(x/4.0);
    px2=x-4*px;
    if(abs(px2)<=.01)
        return texture(matrizA,vec2(px,y)).x;
    if(abs(px2-1)<=.01)
        return texture(matrizA,vec2(px,y)).y;
    if(abs(px2-2)<=.01)
        return texture(matrizA,vec2(px,y)).z;
    if(abs(px2-3)<=.01)
        return texture(matrizA,vec2(px,y)).w;
    return 0.0;
}
float elmBxy(float x,float y){
    float px,px2;
    px=floor(x/4.0);
    px2=x-4*px;
    if(abs(px2)<=.01)
        return texture(matrizB,vec2(px,y)).x;
    if(abs(px2-1)<=.01)
        return texture(matrizB,vec2(px,y)).y;
    if(abs(px2-2)<=.01)
        return texture(matrizB,vec2(px,y)).z;
    if(abs(px2-3)<=.01)
        return texture(matrizB,vec2(px,y)).w;
    return 0.0;
}

void main()
{
    vec4 color1,color2;
    float f1,ac1,ac2,ac3,ac4;
    float tx1,tx2,ty1,ty2;
    tx1=texture(matdim,vec2(0,0)).x;
    ty1=texture(matdim,vec2(0,0)).y;
    tx2=texture(matdim,vec2(0,1)).x;
    ty2=texture(matdim,vec2(0,1)).y;

    color1=texture(matguiax,gl_TexCoord[0].st);
    color2=texture(matguiay,gl_TexCoord[0].st);
    ac1=0;
    for(f1=0;f1<tx1;f1=f1+1){
```

```
50          ac1=ac1+elmAxy(f1,color2.x)*elmBxy(color1.x,f1);
51      }
52      ac2=0;
53      for(f1=0;f1<tx1;f1=f1+1){
54          ac2=ac2+elmAxy(f1,color2.y)*elmBxy(color1.y,f1);
55      }
56      ac3=0;
57      for(f1=0;f1<tx1;f1=f1+1){
58          ac3=ac3+elmAxy(f1,color2.z)*elmBxy(color1.z,f1);
59      }
60      ac4=0;
61      for(f1=0;f1<tx1;f1=f1+1){
62          ac4=ac4+elmAxy(f1,color2.w)*elmBxy(color1.w,f1);
63      }
64      gl_FragColor=vec4(ac1, ac2, ac3, ac4);
65  }
```

A lo largo de esta subsección se ha descrito lo incorporado al motor de álgebra lineal, en lo que resta de esta subsección se hablará de las ventajas obtenidas al implementar las funciones incorporadas a este motor sobre la GPU y la superioridad que exhibió al compararla con la CPU.

Primero recordemos que las ventajas que una operación ejecutada en la GPU pueda aportar dependerá directamente del tipo de GPU que tenga la computadora sobre la que se ejecute la operación, por ello describiremos a continuación las características de la computadora sobre la que se realizó la comparación.

Computadora marca HP.

Memoria RAM de 4 GB.

Procesador AMD Athlon II Dual-Core P320 2.10 GHz

GPU ATI serie 3400 con 40 microprocesadores.

Versión de OpenGL: 3.2

Para explicar la manera en como se compararon las operaciones realizadas en la CPU y en la GPU, es necesario mencionar que se utilizó una función que pueda medir el tiempo en la computadora, sin embargo en el presente trabajo nos encontramos con la limitación de que las funciones encontradas para realizar esta tarea median el tiempo en segundos. Es por esta razón que para evitar valores en el tiempo muy pequeños tuvimos que multiplicar las veces que se realizaban las operaciones; explico, lo que se realizó para superar el obstáculo del tiempo fue hacer que los algoritmos tanto en la GPU como en la CPU realizarán múltiples veces la misma operación dentro de la misma ejecución, esto hacía que el tiempo de ejecución fuese más largo y de esta manera disminuir el error en la medición de tiempos.

Aclarado esto se procederá a describir los tipos de ejecuciones que se realizaron y el número de veces que se ejecutaron.

La principal forma en como se comparó la GPU y la CPU en la biblioteca de álgebra lineal fue con ayuda del algoritmo para multiplicar matrices, ya que es la más compleja. En la tabla contenida en la imagen4.1 se muestran las dimensiones de las matrices que se multiplicaron y el número de veces que se hizo que el algoritmo implementado en la CPU realizara la operación dentro de la misma ejecución, mientras que en la tabla contenida en la imagen4.2 se muestran las mismas dimensiones de las matrices que se multiplicaron y el número de veces que se hizo que el algoritmo implementado en la GPU realizara la operación dentro de la misma ejecución.

La manera aproximada de obtener el valor en que el algoritmo realizó una operación es dividiendo el tiempo de duración de dicho algoritmo sobre el número de veces en que se realizó dicha operación. Realizando esta operación y graficándola, se obtiene la imagen4.3.

Implementación en CPU		
Dimensión de las matrices	Número de veces realizado	Duración
100*100	1000	14 segundos
200*200	500	54 segundos
300*300	50	18 segundos
400*400	20	19 segundos
500*500	20	43 segundos
600*600	10	37 segundos
700*700	10	59 segundos
800*800	5	51 segundos

Figura 4.1: Tabla de dimensiones de matrices y tiempos de realización.

Implementación en GPU		
Dimensión de las matrices	Número de veces realizado	Duración
100*100	89	5 segundos
200*200	78	6 segundos
300*300	67	13 segundos
400*400	56	19 segundos
500*500	45	21 segundos
600*600	34	22 segundos
700*700	23	19 segundos
800*800	12	24 segundos

Figura 4.2: Tabla de dimensiones de matrices y tiempos de realización.

La línea superior dentro de la gráfica 4.3, como señala la flecha, corresponde a los datos aportados por la CPU, la cual puede ser aproximada por una función cúbica, esto es debido al oren que tiene el algoritmo de producto de matrices [3], mientras que la línea inferior corresponde a los datos provistos por la GPU, cabe resaltar que esta línea también puede aproximarse por una función cúbica. Como se puede apreciar en la imagen 4.3 la ventaja que proporciona el implementar el algoritmo en la GPU es considerable, esto debido principalmente al número de microprocesadores de la GPU.

IV.3. Mejoras a las funciones de cálculo

Se procederá a continuación a mostrar los resultados referentes a las funciones implementadas al motor de cálculo.

IV.3.1 Integración numérica

A continuación se muestra un ejemplo de un algoritmo donde se implementa la integración numérica con ayuda de la biblioteca GPU Math.

```
1 //linkers libmathgpu_Calculus.dll.a
2 #include <stdio.h>
```

[3]Para información referente acerca del orden de un algoritmo, véase LIBRO.

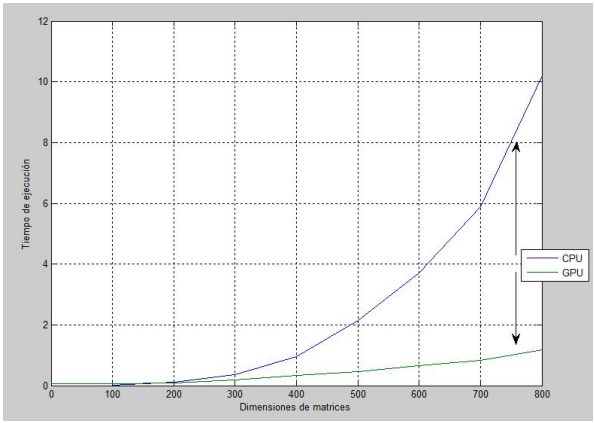

Figura 4.3: Gráfica que muestra la eficiencia de la GPU vs CPU.

```
3  #include <stdlib.h>
4  #include <C:\Users\...\Calculus\dlls\mathgpu_Calculus\main.h>
5
6  int main(){
7
8      void *fn;
9
10     fn=mathGPUwe_CreateFunctionFromString((char *)"cos(x)");
11     printf(" %f",mathGPUwe_DefiniteIntegralf(fn,0,1,.000001));
12
13     system("pause");
14     return 0;
15 }
```

Como se puede apreciar en el ejemplo de código que se acaba de mostrar, la implementación del algoritmo para realizar una integración numérica es considerablemente fácil.

Para implementar dicha función primeramente es necesario crear un apuntador a una variable que sea un tipo de dato vacío (void), lo cual se hace en la línea 8 del ejemplo, se usa este tipo de dato ya que con este es posible almacenar información que tenga un tipo de dato no específico; el tipo de dato en este caso es el creado en la parte interna del motor.

En la línea 10 es donde se especifica la función que se desea integrar con ayuda de la función **mathGPUwe_CreateFunctionFromString**, esto como se puede deducir del ejemplo tiene que ser considerando a la función como una cadena de caracteres; la información retornada por la función mathGPUwe_CreateFunctionFromString es, en la misma línea 10, almacenada por la variable de tipo vacío (void). Es en la línea 11 donde se manda a hacer la operación para la integración numérica con ayuda de la función **mathGPUwe_DefiniteIntegralf** la cual recibe cuatro parámetros y retorna un valor de punto flotante.

La función **mathGPUwe_DefiniteIntegralf** como se acaba de mencionar es una función que recibe cuatro parámetros; el primer parámetro es la función de tipo vacío (void) la cual contiene toda la información referente a la función que se integrará; el segundo parámetro es un valor de tipo flotante (float) el cual indica el límite inferior de la integral; el tercer valor, al igual que el segundo parámetro, es un valor de tipo flotante (float) el cual indica el límite superior de la integral; por último, el cuarto parámetro de entrada es un valor de tipo flotante (float) el cual indica el grosor de la partición que se ha hecho sobre el dominio a integrar [4]. A continuación se muestra de manera general la manera en como se pueden in-

[4]Para más información acerca de la integración léase Spivak

troducir los valores a la función **mathGPUwe_DefiniteIntegralf** así como el tipo de valor que este devuelve.

float mathGPUwoe_DefiniteIntegralf(void *handlefn,float a,float b,float delta)

A continuación se muestra el shader que implementa la integral numérica, el cual fue implementado en el motor de cálculo.

```
1  uniform sampler2DRect matguiavec;
2  uniform int tamtex1;
3
4  float funcion(float x){
5      return ;
6  }
7
8  void main(){
9      float x,y;
10
11     x=texture(matguiavec,gl_TexCoord[0].st).x;
12
13     y=funcion(x);
14
15     gl_FragColor=vec4(y/float(tamtex1*tamtex1),0,0,1);
16 }
```

Como se puede apreciar es un shader muy simple, es en la línea 11 donde se obtiene el valor a evaluar mientras que en la línea 13 se realiza la evaluación. Algo interesante que se puede apreciar en esta función (la cual esta constituida por las líneas 4 - 6) es que no se encuentra de manera explícita la función que se evaluará, esto es debido a que no hay una función fija.

Lo que se hace para poder hacer explícita la función a evaluar es introducir la cadena de caracteres que expresen la función antes del punto y coma (;) que se encuentra en la línea 5. Dicho lo anterior, ahora es posible entender el por qué se expresar la función como cadena de caracteres en línea 5 en el ejemplo del algoritmo que implementa la integración numérica al inicio de la sección.

Es en la línea 15 donde se regresa el valor de la integral, o expresado con mayor precisión, se regresa el valor de un rectángulo, el cual después de sumarlo con los demás rectángulo generados por la partición producirá el valor de la integral. Puede causar un poco de confusión la manera en que se regresa el valor en la línea 15, por ello se hace la aclaración de que $tamtex1 = funcionTecho(\frac{b-a}{\Delta t})$, donde a y b son los límites inferior y superior de la integral respectivamente y Δt es el grosor de la partición.

IV.3.2 Series de Fourier

A continuación se muestra un ejemplo de un algoritmo donde se obtienen los coeficientes para la serie de Fourier con ayuda de la biblioteca GPU Math.

```
1  //linkers libmathgpu_Calculus.dll.a
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #include <C:\Users\...\Calculus\dlls\mathgpu_Calculus\main.h>
6
7  int main(){
8      void *fn1;
9      float A[10000],B[10000];
10     int i1;
11
12     fn1=mathGPUwoe_CreateFunctionFromString((char *)"x");
13     mathGPUwoe_FourierCoef(A,B,10000,fn1,0,1,.00001);
14     mathGPUwoe_DestroyFunction(fn1);
15
16     for(i1=0;i1<40;i1++){
17         printf(" %f\n",A[i1]);
18     }
19
20     printf("\n\n\n\n");
21     system("pause");
22     return 0;
23 }
```

Como se mencionó en el capítulo anterior, el objetivo del algoritmo para obtener la serie de Fourier es la de obtener los coeficientes a_i y b_i, estos valores se guardarán en los vectores que se muestran en la línea 9.

Lo primero que se necesita para utilizar la función de GPU Math para las series de Fourier es definir un apuntador a una variable de tipo vacío (void), la cual almacenará la información requerida por la función que se aproximará, además de esto, es necesario utilizar la función **mathGPUwoe_CreateFunctionFromString** la cual recibe como parámetro una cadena de caracteres que contendrá la función a aproximar, esta función regresará la información de la función por lo que es necesario que esta información sea guardada en la variable de tipo vació (void).

Una vez creada la función a a proximar lo siguiente es obtener los valores de los coeficientes de la serie de Fourier, esto se hace con la función **mathGPUwoe_FourierCoef**, la cual requiere de siete parámetros, los primeros dos son vectores flotantes (float) que obtendrán los valores de los coeficientes de Fourier a_i y b_i respectivamente; el tercer valor representa el límite superior de la suma de la ecuación 3.1, la cual por obvias razones no puede ser infinito; el cuarto parámetro representa la variable que contiene la información de la función a aproximar; el quinto y sexto parámetro representan el valor a y b en la ecuación 3.1 respectivamente; el séptimo y último parámetro representa el Δt con que se efectuarán las integrales para obtener los coeficientes.

Por último antes de terminar el algoritmo es necesario liberar la información creada y almacenada por la función para crear los coeficientes de Fourier, esto se hace, como se muestra en la línea 14, con la función **mathGPUwoe_DestroyFunction** la cual requiere como parámetro la función de tipo vacío (void).

A continuación se muestra de manera general la manera en como se pueden introducir los valores a la función **mathGPUwoe_FourierCoef** así como el tipo de valor que devuelve.

mathGPUwoe_FourierCoef(float *A,float *B,int n,void *handlefn,float a,float b,float delta)

En el siguiente código se muestra el shader que implementa la obtención de coeficientes a_i para las series de Fourier, está de más mencionar que el shader para obtener los coeficientes b_i es igual a este excepto por la línea 15 la cual sustituye la función sin con cos

```
1  uniform sampler2DRect matguiavec;
2  uniform float a,b,delta;
3  float funcion(float x){
4          return ;
5  }
6
7  void main(){
8          float x,y,n,vf1,vf2;
9           float pi=3.141592654;
10
11          n=texture(matguiavec,gl_TexCoord[0].st).x;
12          vf2=0;
13          for(vf1=a;vf1<b;vf1=vf1+delta){
14                   x=vf1+delta/2;
15                 vf2=vf2+funcion(x)*cos(n*pi*x/(b-a));
16          }
17          vf2=(vf2*delta)*2/(b-a);
18
19          gl_FragColor=vec4(vf2,0,0,1);
20 }
```

IV.4. Mejoras en el procesamiento de Imagenes

IV.4.1 Detección de bordes

A continuación mostramos una imagen titulada "Genios del siglo pasado", la cual fue convertida a escala de grises (imagen 4.5) y más tarde tratada con los diferentes operadores (Sobel, Prewits, Roberts y Canny).

Figura 4.4: Imagen que será tratada por los diferentes operadores para detección de bordes.

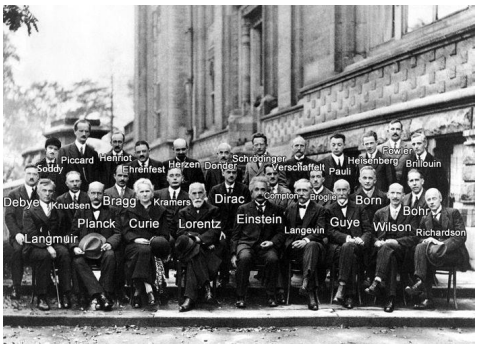

Figura 4.5: Imagen 4.4 que fue sometida a un filtro para convertirlo a escala de grises.

Una vez que obtuvimos la imagen 4.5 estamos en condiciones de aplicar las diferentes máscaras explicadas en el capítulo 3, las imágenes 4.6 y 4.7 muestran el resultado de la ecuación 3.6 con las máscaras para el operador Sobel y Prewits respectivamente.

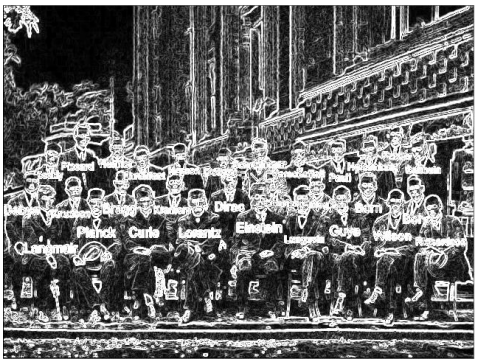

Figura 4.6: Imagen 4.4 que fue sometida a un filtro para convertirlo a escala de grises.

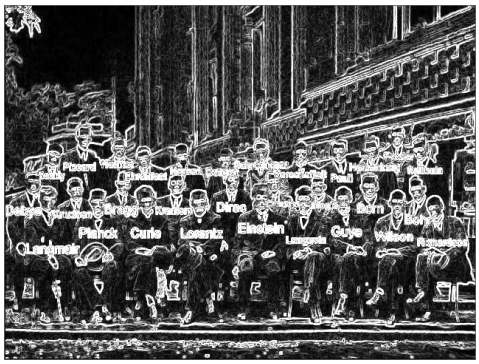

Figura 4.7: Imagen 4.4 que fue sometida a un filtro para convertirlo a escala de grises.

Como en la primera subsección, a continuación en la tabla contenida en la imagen 4.8 mostramos los datos de ejecución realizados en la CPU y en la GPU.

	Dimension de la imagen	Número de veces realizado	Duración
CPU	640*480	100	45 segundos
GPU	640*480	100	6 segundos

Figura 4.8: Tabla de dimensiones de matrices y tiempos de realización.

Con base en la información mostrada en la imagen 4.8 se puede apreciar la ventaja mostrada por la ejecución realizada en la GPU, la cual exhibe una mejora de entre 7 y 8 veces.

IV.5. Aplicación

El objetivo de la presente tesis fue la elaboración de una biblioteca matemática que puede realizar diversas operaciones a través de la GPU, dicha biblioteca fue llamada GPU Math, de esto es de lo que se ha estado hablando a lo largo de este trabajo. Además de los objetivos planteados, es decir la biblioteca, es necesario destacar que como resultado adicional se construyo un software el cual fue bautizado con el nombre: OPTICT.

El resto de este capítulo se dedica a la descripción de este software.

IV.5.1 OPTICT

Como se acaba de mencionar, además de cumplir los objetivos planteados al inicio del proyecto, se desarrollo un software llamado OPTICT como resultado de la unión de varios motores contenidos en la biblioteca GPU Math.

OPTICT es un software que tiene la finalidad de ser un *mouse* óptico. La manera de alcanzar esta finalidad es a través de los siguientes pasos:

Con ayuda de una cámara conectada a la computadora donde se encuentra el software y que se encuentre a una corta distancia del ojo de una persona, no hay restricción sobre el ojo de la persona, es decir, no importa si es el ojo izquierdo o derecho; se enfocará el ojo de la persona.

La función de esta cámara será poder captar los movimientos del ojo de la persona.

La función interna que tendrá el software será procesar las imágenes que sean captadas por la cámara, para más tarde procesar dichas imágenes con un filtro para convertirlas a escala de grises y luego aplicarle algún filtro para la detección de bordes.

Es apropiado mencionar que el objetivo de la cámara es captar solamente las imágenes del ojo.

Una vez aplicados estos filtros a las imágenes captadas por la cámara se procede a aplicar la transformada de Hough para la detección de círculos. El objetivo de esta transformación es es captar el iris del ojo; al obtener el iris del ojo, es decir, al detectar el círculo contenido en el ojo se procede a encontrar el centro del círculo.

Dicho centro será el puntero de nuestro *mouse*, la innovación al realizar este proceso es que el lugar donde se enfoca el, ahora, puntero coincide con el lugar al que la persona enfoca la mirada; de ahí que cada vez que una persona enfoque un punto en la pantalla será el punto donde se encontrará el puntero.

Como se puede apreciar, el proceso que realiza OPTICT utiliza varias funciones de GPU Math las cuales all estar programadas en la GPU hacen que todas estas operaciones sean en tiempo real.

Es necesario decir que el único requisito para utilizar OPTICT es la necesidad de calibrar este al comienzo de su uso, la calibración consite en enfocar la vista en cuatro puntos distintos de la pantalla, los cuales coinciden con las esquinas de la pantalla, esto es con el fin de restringir el dominio de visión del ojo al área de la pantalla.

Además de esto se considerará el equivalente a un click con el *mouse* como un pestañeo. Sin embargo cabe resaltar que este detalle aún se está perfeccionando.

V. CONCLUSIONES

La GPU aporta un beneficio muy necesario en la actualidad en lo que refiere al tiempo de ejecución de diferentes algoritmos, dicho beneficio es la velocidad de ejecución. La ventaja que se logra al programar sobre la GPU se acrecienta en relación al número de datos con los que se trabajen; por otro lado, si se trabaja con una pequeña cantidad de datos, es incluso posible que la ejecución en la GPU requiera de más tiempo que la ejecución en la CPU, esto se debe a que es necesario considerar el tiempo que se requiere para llevar los datos de la CPU a la GPU y viceversa. En lo sucesivo es claro que los programas se irán recargando más sobre la GPU, esto quiere decir que módulos o partes de los algoritmos se ejecutarán dentro de la GPU para aportar mayor rapidez a los tiempos de ejecución. Por otra parte, la creación de OPTICT es un gran logro ya que éste software incorpora varios algoritmos que por sí solos conllevan un gran tiempo de ejecución lo cual fue enormemente minimizado gracias a la GPU, este software, dejando de lado los beneficios técnicos que puede aportar, tiene un amplio campo de utilidad además de que presenta un camino con el que se pueden desarrollar diversos software innovadores y muy útiles. GPU Math como se ha dicho gran cantidad de veces a lo largo de este documento, es una biblioteca que incorpora diversas funciones pertenecientes a varias ramas de las matemáticas, dicha biblioteca se deja junto con el presente documento guardado en los discos entregados a las bibliotecas de Ingeniería y Central de la UAQ para que cualquiera que así lo desee pueda usarla o bien utilizarlas como guía para aprender un poco sobre la manera de programar una GPU, cosa que en un futuro muy próximo será ampliamente valioso.

VI. ANEXO

```
#include <windows.h>
#include <winuser.h>
#include <process.h>
#define GLEW_STATIC
#include <gl/glew.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "C:\Users\Juan Carlos\Documents\docs code blocks1\dlls\dlls_plug-in1m\proccnx\include\proccnx.h"
#include "C:\Users\Juan Carlos\Documents\docs code blocks1\dlls\dlls_plug-in1m\ImpColas1\include\ImpColas1.h"
#include "C:\Users\Juan Carlos\Documents\docs code blocks1\dlls\dlls_plug-in1m\matef1\include\matef1.h"
#include "C:\Users\Juan Carlos\Documents\docs code blocks1\dlls\dlls_plug-in1m\shad1m\include\shad1m.h"

#define pi 3.141592654
#define eps3 0.001
#define eps2 0.01
#define eps1 0.1

class cFlotante1m{
    public:
        float vf;

        void asignanumaleatorio(){
            int i1;
            for(i1=0;i1<32;i1++){
                vf=cambiabitsf1(vf,i1,int((2*rand())/RAND_MAX));
            }
        }
        float retornafloat1(){
            return vf;
        }
        void imprimebits1(){
            int i1;
            for(i1=0;i1<32;i1++){
                printf("%i",retornabitsf1(vf,i1));
            }
        }
        void cambiabit(int nb,float val){
            if(nb<0 && nb>=32){
                return ;
            }
            vf=cambiabitsf1(vf,nb,val);
        }
        float retornabit(int nb){
            if(nb<0 && nb>=32){
                return 0.0;
            }
            return retornabitsf1(vf,nb);
        }
};
class cListaConjuntoEntrenamiento1{
    public:
        float *entrada_,*salida_;
        cListaConjuntoEntrenamiento1 *sig_;
};

class cManConjuntoEntrenamiento1{
    public:
        int np_;
        int te_,ts_;
        cListaConjuntoEntrenamiento1 *elm1_;

        cManConjuntoEntrenamiento1(int te,int ts){
            np_=0;
            te_=te;
            ts_=ts;
            elm1_=NULL;
        }
        ~cManConjuntoEntrenamiento1(){
            int i1;
            cListaConjuntoEntrenamiento1 *auxlce1,*auxlce2;

            auxlce1=elm1_;
            for(i1=0;i1<np_;i1++){
```

```
77                  free(auxlce1->entrada_);
78                  free(auxlce1->salida_);
79                  auxlce2=auxlce1;
80                  auxlce1=auxlce1->sig_;
81                  delete auxlce2;
82              }
83          }
84
85          void AgregaElmAlInicio(float *entrada,float *salida){
86              int i1;
87              cListaConjuntoEntrenamiento1 *auxlce1;
88
89              auxlce1=new cListaConjuntoEntrenamiento1();
90              auxlce1->entrada_=(float *)malloc(te_*sizeof(float));
91              for(i1=0;i1<te_;i1++){
92                  *(auxlce1->entrada_+i1)=*(entrada+i1);
93              }
94              auxlce1->salida_=(float *)malloc(ts_*sizeof(float));
95              for(i1=0;i1<ts_;i1++){
96                  *(auxlce1->salida_+i1)=*(salida+i1);
97              }
98              auxlce1->sig_=elm1_;
99
100             elm1_=auxlce1;
101             np_++;
102         }
103         void AgregaElmAlFinal(float *entrada,float *salida){
104             int i1;
105             cListaConjuntoEntrenamiento1 *auxlce1,*auxlce2;
106
107             auxlce1=new cListaConjuntoEntrenamiento1();
108             auxlce1->entrada_=(float *)malloc(te_*sizeof(float));
109             for(i1=0;i1<te_;i1++){
110                 *(auxlce1->entrada_+i1)=*(entrada+i1);
111             }
112             auxlce1->salida_=(float *)malloc(ts_*sizeof(float));
113             for(i1=0;i1<ts_;i1++){
114                 *(auxlce1->salida_+i1)=*(salida+i1);
115             }
116             auxlce1->sig_=NULL;
117
118             if(np_==0){
119                 elm1_=auxlce1;
120                 np_++;
121                 return ;
122             }
123             auxlce2=elm1_;
124             for(i1=0;i1<np_-1;i1++){
125                 auxlce2=auxlce2->sig_;
126             }
127             auxlce2->sig_=auxlce1;
128             np_++;
129         }
130         void AgregaElmEnPosi(float *entrada,float *salida,int i){
131             int i1;
132             cListaConjuntoEntrenamiento1 *auxlce1,*auxlce2;
133
134             if(i<=0){
135                 AgregaElmAlFinal(entrada,salida);
136                 return ;
137             }
138             if(i>=np_){
139                 AgregaElmAlFinal(entrada,salida);
140                 return ;
141             }
142
143
144             auxlce1=new cListaConjuntoEntrenamiento1();
145             auxlce1->entrada_=(float *)malloc(te_*sizeof(float));
146             for(i1=0;i1<te_;i1++){
147                 *(auxlce1->entrada_+i1)=*(entrada+i1);
148             }
149             auxlce1->salida_=(float *)malloc(ts_*sizeof(float));
150             for(i1=0;i1<te_;i1++){
151                 *(auxlce1->salida_+i1)=*(salida+i1);
152             }
153
154             auxlce2=elm1_;
155             for(i1=0;i1<i-1;i1++){
156                 auxlce2=auxlce2->sig_;
157             }
158             auxlce1->sig_=auxlce2->sig_;
159             auxlce2->sig_=auxlce1;
160             np_++;
161         }
162
163         void RetornaElmEnPosi(int i,float *entrada,float *salida){
164             int i1;
165             cListaConjuntoEntrenamiento1 *auxlce1;
166
167             if(i<0 || i>=np_){
168                 return ;
```

```
169             }
170
171             auxlce1=elm1_;
172             for(i1=0;i1<i;i1++){
173                 auxlce1=auxlce1->sig_;
174             }
175
176             for(i1=0;i1<te_;i1++){
177                 *(entrada+i1)=*(auxlce1->entrada_+i1);
178             }
179             for(i1=0;i1<ts_;i1++){
180                 *(salida+i1)=*(auxlce1->salida_+i1);
181             }
182         }
183
184         cListaConjuntoEntrenamiento1 *RetornaEstCompletaEnPosi(int i){
185             int i1;
186             cListaConjuntoEntrenamiento1 *auxlce1;
187
188             if(i<0 || i>=np_){
189                 return NULL;
190             }
191
192             auxlce1=elm1_;
193             for(i1=0;i1<i;i1++){
194                 auxlce1=auxlce1->sig_;
195             }
196             return auxlce1;
197         }
198
199         void QuitarElmDelInicio(){
200             cListaConjuntoEntrenamiento1 *auxlce1;
201
202             if(np_<=0){
203                 return ;
204             }
205             auxlce1=elm1_;
206             elm1_=elm1_->sig_;
207             free(auxlce1->entrada_);
208             free(auxlce1->salida_);
209             delete auxlce1;
210             np_--;
211         }
212         void QuitarElmDelFinal(){
213             cListaConjuntoEntrenamiento1 *auxlce1;
214             int i1;
215
216             if(np_<=0){
217                 return ;
218             }
219
220             if(np_==1){
221                 free(elm1_->entrada_);
222                 free(elm1_->salida_);
223                 delete elm1_;
224                 elm1_=NULL;
225                 np_=0;
226                 return ;
227             }
228             auxlce1=elm1_;
229             for(i1=0;i1<np_-1;i1++){
230                 auxlce1=auxlce1->sig_;
231             }
232             free(auxlce1->sig_->entrada_);
233             free(auxlce1->sig_->salida_);
234             delete auxlce1->sig_;
235             auxlce1->sig_=NULL;
236             np_--;
237         }
238         void QuitarElmEnPosi(int pos){
239             cListaConjuntoEntrenamiento1 *auxlce1,*auxlce2;
240             int i1;
241
242             if(pos<0 || pos>=np_){
243                 return ;
244             }
245             if(pos==0){
246                 QuitarElmDelInicio();
247                 return ;
248             }
249             if(pos==np_-1){
250                 QuitarElmDelFinal();
251                 return ;
252             }
253
254             auxlce1=elm1_;
255             for(i1=0;i1<pos-1;i1++){
256                 auxlce1=auxlce1->sig_;
257             }
258             auxlce2=auxlce1->sig_;
259             auxlce1->sig_=auxlce2->sig_;
260             free(auxlce2->entrada_);
```

```
261             free(auxlce2->salida_);
262             delete auxlce2;
263             np_--;
264         }
265 };
266 class cRedNeu1m{
267     public:
268         int nc_,*t_;
269         float *valsr1_,*valsbias_,**valsal1_,***mpesc1_,***deltapesoM_;
270 
271         cRedNeu1m(int nc,int *t){
272             int i1,i2;
273 
274             nc_=nc;
275             t_=(int *)malloc(nc_*sizeof(float));
276             for(i1=0;i1<nc_;i1++){
277                 *(t_+i1)=*(t+i1);
278             }
279 
280             valsr1_=(float *)malloc(*t*sizeof(float));
281             valsbias_=(float *)malloc((nc_-1)*sizeof(float));
282             for(i1=0;i1<nc_-1;i1++){
283                 *(valsbias_+i1)=-1;
284             }
285             valsal1_=(float **)malloc((nc_-1)*sizeof(float*));
286             for(i1=0;i1<nc_-1;i1++){
287                 *(valsal1_+i1)=(float *)malloc(*(t_+i1+1)*sizeof(float));
288             }
289 
290             mpesc1_=(float ***)malloc((nc_-1)*sizeof(float**));
291             deltapesoM_=(float ***)malloc((nc_-1)*sizeof(float**));
292             for(i1=0;i1<nc_-1;i1++){
293                 *(mpesc1_+i1)=(float **)malloc(*(t_+i1+1)*sizeof(float*));
294                 *(deltapesoM_+i1)=(float **)malloc(*(t_+i1+1)*sizeof(float*));
295                 for(i2=0;i2<*(t_+i1+1);i2++){
296                     *(*(mpesc1_+i1)+i2)=(float *)malloc((*(t_+i1)+1)*sizeof(float));
297                     *(*(deltapesoM_+i1)+i2)=(float *)malloc((*(t_+i1)+1)*sizeof(float));
298                 }
299             }
300         }
301         ~cRedNeu1m(){
302             int i1,i2;
303 
304             for(i1=0;i1<nc_-1;i1++){
305                 for(i2=0;i2<*(t_+i1+1);i2++){
306                     free(*(*(mpesc1_+i1)+i2));
307                     free(*(*(deltapesoM_+i1)+i2));
308                 }
309                 free(*(mpesc1_+i1));
310                 free(*(deltapesoM_+i1));
311             }
312             free(mpesc1_);
313             free(deltapesoM_);
314 
315             for(i1=0;i1<nc_-1;i1++){
316                 free(*(valsal1_+i1));
317             }
318             free(valsal1_);
319 
320             free(t_);
321             free(valsr1_);
322             free(valsbias_);
323         }
324 };
325 
326 class cEstVentana1{
327     public:
328         HWND vent1;
329         HANDLE mutexgral;
330 };
331 class cManProc1{
332     public:
333         void *proc1;
334         HANDLE mutexleer,mutexesribir,mutexproc1;
335 };
336 class cColaInstrYDatos1{
337     public:
338         void *cola,*colapaq;
339         HANDLE mutexgral;
340 
341 };
342 class cRedyConjConMutex1{
343     public:
344         cRedNeu1m *redneu1_;
345         cManConjuntoEntrenamiento1 *conj1_;
346         HANDLE mutexgral_;
347 };
348 
349 class cEstHiloVisor3d1{
350     public:
351         HGLRC hRC1;
352         HDC hDC1;
```

```
353         int est1[3];
354         HANDLE mgral1;
355 };
356
357 class cOjetosDelComputoEnGPU1{
358     public:
359         GLuint fbprop,texmatbuffprop[2],texmatinfocompprop,texmatguiasprop[2];
360         int tamlvisionprop,tamlinfocompprop;
361 };
362
363 int creaclaseventanas1(HINSTANCE);
364 void PropagacionGPU1();
365
366 cManProc1 g1m_procp;
367 cColaInstrYDatos1 g1m_colapad1,g1m_colapadsend1,g1m_colagpuops;
368 cRedyConjConMutex1 g1m_redneu1;
369 HANDLE g1m_MutexUsoOpg1;
370 cOjetosDelComputoEnGPU1 g1m_ocgpu1;
371
372 float f(FILE *ar1,float alpha,float ***pesos){
373     int i1,i2,i3,i4;
374     cListaConjuntoEntrenamiento1 *auxlce1;
375     float vf1,errac1;
376
377     errac1=0;
378
379     for(i1=0;i1<g1m_redneu1.conj1_->np_;i1++){
380
381         auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i1);
382
383         for(i2=0;i2<*(g1m_redneu1.redneu1_->t_);i2++){
384             *(g1m_redneu1.redneu1_->valsr1_+i2)=*(auxlce1->entrada_+i2);
385         }
386
387         for(i2=1;i2<g1m_redneu1.redneu1_->nc_;i2++){
388             if(i2==1){
389                 for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2);i3++){
390                     vf1=0;
391                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i2-1);i4++){
392                         vf1+=*(g1m_redneu1.redneu1_->valsr1_+i4)*(*(*(*(g1m_redneu1.redneu1_->mpesc1_)+i3)+i4)+alpha**(*(*(pesos)+i3)+i4));
393                     }
394                     vf1+=*(g1m_redneu1.redneu1_->valsbias_+i2-1)*(*(*(*(g1m_redneu1.redneu1_->mpesc1_)+i3)+i4)+alpha**(*(*(pesos)+i3)+i4));
395                     *(*(g1m_redneu1.redneu1_->valsal1_)+i3)=1/(1+exp(-vf1));
396                 }
397             }
398             else{
399                 for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2);i3++){
400                     vf1=0;
401                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i2-1);i4++){
402                         vf1+=*(*(g1m_redneu1.redneu1_->valsal1_+i2-2)+i4)*(*(*(*(g1m_redneu1.redneu1_->mpesc1_+i2-1)+i3)+i4)+alpha**(*(*(pesos+i2-1
403                     }
404                     vf1+=*(g1m_redneu1.redneu1_->valsbias_+i2-1)*(*(*(*(g1m_redneu1.redneu1_->mpesc1_+i2-1)+i3)+i4)+alpha**(*(*(pesos+i2-1)+i3)+i4)
405                     *(*(g1m_redneu1.redneu1_->valsal1_+i2-1)+i3)=1/(1+exp(-vf1));
406                 }
407             }
408         }
409         vf1=0;
410         for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1);i2++){
411             vf1+=pow(*(*(g1m_redneu1.redneu1_->valsal1_+g1m_redneu1.redneu1_->nc_-2)+i2)-*(auxlce1->salida_+i2),2);
412         }
413         errac1+=vf1;
414     }
415     return errac1;
416 }
417 float goldenSectionSearch(float a,float b,float c,float tau,int n,float ***pesos) {
418     float x,auxa,auxb,auxc,resphi=.5;
419     int i1;
420 /*
421     for(i1=0;i1<n;i1++){
422         if(c-b>b-a){
423             x=b+resphi*(c-b);
424         }
425         else{
426             x=b-resphi*(b-a);
427         }
428         if(fabs(c-a)<tau*(abs(b)+abs(x))){
429             return(c+a)/2;
430         }
431         if(f(x,pesos)<f(b,pesos)){
432             if (c - b > b - a){
433                 auxa=b;
434                 auxb=x;
435                 auxc=c;
436                 a=auxa;
437                 b=auxb;
438                 c=auxc;
439             }
440             else{
441                 auxa=a;
442                 auxb=x;
443                 auxc=b;
444                 a=auxa;
```

```
                b=auxb;
                c=auxc;
            }
        }
        else {
            if (c - b > b - a){
                auxa=a;
                auxb=b;
                auxc=x;
                a=auxa;
                b=auxb;
                c=auxc;
            }
            else{
                auxa=x;
                auxb=b;
                auxc=c;
                a=auxa;
                b=auxb;
                c=auxc;
            }
        }
    }
  */  return (c+a)/2;
}

float otrabusqueda1(FILE *ar1,float ***pesos){
    float vf1;
    float vf2,vf3,vf4;

    vf2=f(ar1,.1,pesos);
    vf3=.1;

    for(vf1=0;vf1<=10;vf1=vf1+.1){
        vf4=f(ar1,vf1,pesos);
//        fprintf(ar1,"%f\t",vf4);
        if(vf4<=vf2){
            vf2=vf4;
            vf3=vf1;
        }
    }
//    fprintf(ar1,"\n\n",vf4);

    return vf3;
}

void Propagacion1(int npat){
    int i1,i2,i3;
    cListaConjuntoEntrenamiento1 *auxlce1;
    float vf1;

    if(g1m_redneu1.redneu1_==NULL || g1m_redneu1.conj1_==NULL){
        return ;
    }

    auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(npat);

    if(auxlce1==NULL){
        return ;
    }

    for(i1=0;i1<*(g1m_redneu1.redneu1_->t_);i1++){
        *(g1m_redneu1.redneu1_->valsr1_+i1)=*(auxlce1->entrada_+i1);
    }
    for(i1=1;i1<g1m_redneu1.redneu1_->nc_;i1++){
        if(i1==1){
            for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1);i2++){
                vf1=0;
                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i1-1);i3++){
                    vf1+=*(g1m_redneu1.redneu1_->valsr1_+i3)*
                    *(*(*(g1m_redneu1.redneu1_->mpesc1_)+i2)+i3);
                }
                vf1+=*(g1m_redneu1.redneu1_->valsbias_+i1-1)*
                *(*(*(g1m_redneu1.redneu1_->mpesc1_+i1-1)+i2)+i3);
                *(*(g1m_redneu1.redneu1_->valsal1_)+i2)=1/(1+exp(-vf1));
            }
        }
        else{
            for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1);i2++){
                vf1=0;
                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i1-1);i3++){
                    vf1+=*(*(g1m_redneu1.redneu1_->valsal1_+i1-2)+i3)*
                    *(*(*(g1m_redneu1.redneu1_->mpesc1_+i1-1)+i2)+i3);
                }
                vf1+=*(g1m_redneu1.redneu1_->valsbias_+i1-1)*
                *(*(*(g1m_redneu1.redneu1_->mpesc1_+i1-1)+i2)+i3);
                *(*(g1m_redneu1.redneu1_->valsal1_+i1-1)+i2)=1/(1+exp(-vf1));
            }
        }
    }

}
```

```
537
538 void Entrenamiento1(int nepoch,float eta){
539     int i1,i2,i3,i4,i5;
540     float ti,oi,delta,oj,vf1,**deltas;
541     cListaConjuntoEntrenamiento1 *auxlce1;
542
543     deltas=(float **)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float *));
544     for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
545         *(deltas+i1)=(float *)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float));
546     }
547
548     for(i1=0;i1<nepoch;i1++){
549         for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
550             Propagacion1(i2);
551             auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
552             for(i3=g1m_redneu1.redneu1_->nc_-1;i3>=1;i3--){
553                 if(i3==g1m_redneu1.redneu1_->nc_-1){
554                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
555                         ti=*(auxlce1->salida_+i4);
556                         oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
557                         delta=(ti-oi)*oi*(1-oi);
558                         *(*(deltas+i3-1)+i4)=delta;
559                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
560                             oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
561                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
562                             vf1=vf1+eta*delta*oj;
563                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
564                         }
565                         oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
566                         vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
567                         vf1=vf1+eta*delta*oj;
568                         *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
569                     }
570                 }
571                 else if(i3==1){
572                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
573                         oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
574                         delta=0;
575                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
576                             delta+=*(*(deltas+i3)+i5)*
577                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
578                         }
579                         delta=delta*oi*(1-oi);
580                         *(*(deltas+i3-1)+i4)=delta;
581                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
582                             oj=*(g1m_redneu1.redneu1_->valsr1_+i5);
583                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
584                             vf1=vf1+eta*delta*oj;
585                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
586                         }
587                         oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
588                         vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
589                         vf1=vf1+eta*delta*oj;
590                         *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
591                     }
592                 }
593                 else{
594                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
595                         oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
596                         delta=0;
597                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
598                             delta+=*(*(deltas+i3)+i5)*
599                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
600                         }
601                         delta*=oi*(1-oi);
602                         *(*(deltas+i3-1)+i4)=delta;
603                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
604                             oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
605                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
606                             vf1=vf1+eta*delta*oj;
607                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
608                         }
609                         oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
610                         vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
611                         vf1=vf1+eta*delta*oj;
612                         *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
613                     }
614                 }
615             }
616         }
617     }
618
619     for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
620         free(*(deltas+i1));
621     }
622     free(deltas);
623 }
624 void EntrenamientoConMomentum1(int nepoch,float eta,float gamma){
625     int i1,i2,i3,i4,i5;
626     float ti,oi,delta,oj,vf1,**deltas,DeltaM;
627     cListaConjuntoEntrenamiento1 *auxlce1;
628
```

```
629     deltas=(float **)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float *));
630     for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
631         *(deltas+i1)=(float *)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float));
632     }
633     for(i1=1;i1<g1m_redneu1.redneu1_->nc_;i1++){
634         for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1);i2++){
635             for(i3=0;i3<=*(g1m_redneu1.redneu1_->t_+i1-1);i3++){
636                 *(*(*(g1m_redneu1.redneu1_->deltapesoM_+i1-1)+i2)+i3)=0;
637             }
638         }
639     }
640
641     for(i1=0;i1<nepoch;i1++){
642         for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
643             Propagacion1(i2);
644             auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
645             for(i3=g1m_redneu1.redneu1_->nc_-1;i3>=1;i3--){
646                 if(i3==g1m_redneu1.redneu1_->nc_-1){
647                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
648                         ti=*(auxlce1->salida_+i4);
649                         oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
650                         delta=(ti-oi)*oi*(1-oi);
651                         *(*(deltas+i3-1)+i4)=delta;
652                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
653                             oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
654                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
655                             DeltaM=eta*delta*oj-gamma**(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5);
656                             vf1=vf1+DeltaM;
657                             *(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5)=DeltaM;
658                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
659                         }
660                         oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
661                         vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
662                         DeltaM=eta*delta*oj-gamma**(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5);
663                         vf1=vf1+DeltaM;
664                         *(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5)=DeltaM;
665                         *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
666                     }
667                 }
668                 else if(i3==1){
669                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
670                         oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
671                         delta=0;
672                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
673                             delta+=*(*(deltas+i3)+i5)*
674                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
675                         }
676                         delta=delta*oi*(1-oi);
677                         *(*(deltas+i3-1)+i4)=delta;
678                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
679                             oj=*(g1m_redneu1.redneu1_->valsr1_+i5);
680                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
681                             DeltaM=eta*delta*oj-gamma**(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5);
682                             vf1=vf1+DeltaM;
683                             *(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5)=DeltaM;
684                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
685                         }
686                         oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
687                         vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
688                         DeltaM=eta*delta*oj-gamma**(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5);
689                         vf1=vf1+DeltaM;
690                         *(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5)=DeltaM;
691                         *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
692                     }
693                 }
694                 else{
695                     for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
696                         oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
697                         delta=0;
698                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
699                             delta+=*(*(deltas+i3)+i5)*
700                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
701                         }
702                         delta*=oi*(1-oi);
703                         *(*(deltas+i3-1)+i4)=delta;
704                         for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
705                             oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
706                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
707                             DeltaM=eta*delta*oj-gamma**(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5);
708                             vf1=vf1+DeltaM;
709                             *(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5)=DeltaM;
710                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
711                         }
712                         oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
713                         vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
714                         DeltaM=eta*delta*oj-gamma**(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5);
715                         vf1=vf1+DeltaM;
716                         *(*(*(g1m_redneu1.redneu1_->deltapesoM_+i3-1)+i4)+i5)=DeltaM;
717                         *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
718                     }
719                 }
720             }
```

```
721             }
722         }
723 
724         for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
725             free(*(deltas+i1));
726         }
727 
728         free(deltas);
729     }
730     void Entrenamiento3(int nepoch,float eta){
731         int i1,i2,i3,i4,i5;
732         float ti,oi,delta,oj,vf1,**deltas,error,salida,salidad,errorm=0;
733         cListaConjuntoEntrenamiento1 *auxlce1;
734 
735 
736         deltas=(float **)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float *));
737         for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
738             *(deltas+i1)=(float *)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float));
739         }
740 
741         for(i1=0;i1<nepoch;i1++){
742             for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
743                 Propagacion1(i2);
744 
745                 error=0;
746                 auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
747                 for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1);i3++){
748                     salida=*(*(g1m_redneu1.redneu1_->valsal1_+g1m_redneu1.redneu1_->nc_-2)+i3);
749                     salidad=*(auxlce1->salida_+i3);
750                     error=error+fabs(salida-salidad);
751                 }
752                 if(i1==0 && i2==0){
753                     errorm=error/eta;
754                 }
755                 else{
756                     eta=error/errorm;
757                 }
758                 auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
759                 for(i3=g1m_redneu1.redneu1_->nc_-1;i3>=1;i3--){
760                     if(i3==g1m_redneu1.redneu1_->nc_-1){
761                         for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
762                             ti=*(auxlce1->salida_+i4);
763                             oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
764                             delta=(ti-oi)*oi*(1-oi);
765                             *(*(deltas+i3-1)+i4)=delta;
766                             for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
767                                 oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
768                                 vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
769                                 vf1=vf1+eta*delta*oj;
770                                 *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
771                             }
772                             oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
773                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
774                             vf1=vf1+eta*delta*oj;
775                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
776                         }
777                     }
778                     else if(i3==1){
779                         for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
780                             oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
781                             delta=0;
782                             for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
783                                 delta+=*(*(deltas+i3)+i5)*
784                                 *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
785                             }
786                             delta=delta*oi*(1-oi);
787                             *(*(deltas+i3-1)+i4)=delta;
788                             for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
789                                 oj=*(g1m_redneu1.redneu1_->valsr1_+i5);
790                                 vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
791                                 vf1=vf1+eta*delta*oj;
792                                 *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
793                             }
794                             oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
795                             vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
796                             vf1=vf1+eta*delta*oj;
797                             *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
798                         }
799                     }
800                     else{
801                         for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
802                             oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
803                             delta=0;
804                             for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
805                                 delta+=*(*(deltas+i3)+i5)*
806                                 *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
807                             }
808                             delta*=oi*(1-oi);
809                             *(*(deltas+i3-1)+i4)=delta;
810                             for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
811                                 oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
812                                 vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
```

```
                            vf1=vf1+eta*delta*oj;
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5);
                        vf1=vf1+eta*delta*oj;
                        *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3-1)+i4)+i5)=vf1;
                    }
                }
            }
        }
    }

    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        free(*(deltas+i1));
    }
    free(deltas);
}
void EntrenamientoConjugado1(int nepoch,float eta){
    int i1,i2,i3,i4,i5;
    float ti,oi,delta,oj,vf1,**deltas;
    cListaConjuntoEntrenamiento1 *auxlce1;
    float ***pesos1;

    deltas=(float **)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float *));
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        *(deltas+i1)=(float *)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float));
    }
    pesos1=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        *(pesos1+i1)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float *));
        for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
            *(*(pesos1+i1)+i2)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i1)+1)*sizeof(float *));
        }
    }

    for(i1=0;i1<nepoch;i1++){
        for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                    *(*(*(pesos1+i2)+i3)+i4)=0;
                }
            }
        }
        for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
            Propagacion1(i2);
            auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
            for(i3=g1m_redneu1.redneu1_->nc_-1;i3>=1;i3--){
                if(i3==g1m_redneu1.redneu1_->nc_-1){
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        ti=*(auxlce1->salida_+i4);
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=(ti-oi)*oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
                            vf1=delta*oj;
                            *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                    }
                }
                else if(i3==1){
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=0;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
                            delta+=*(*(deltas+i3)+i5)*
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
                        }
                        delta=delta*oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(g1m_redneu1.redneu1_->valsr1_+i5);
                            vf1=delta*oj;
                            *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                    }
                }
                else{
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=0;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
                            delta+=*(*(deltas+i3)+i5)*
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
                        }
```

```
                        delta*=oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
                            vf1=delta*oj;
                            *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                    }
                }
            }
        }
        for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                    *(*(*(g1m_redneu1.redneu1_->mpesc1_+i2)+i3)+i4)+=eta**(*(*(pesos1+i2)+i3)+i4);
                }
            }
        }
    }

    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
            free(*(*(pesos1+i1)+i2));
        }
        free(*(pesos1+i1));
    }
    free(pesos1);
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        free(*(deltas+i1));
    }
    free(deltas);
}
void EntrenamientoConjugadoFR1(int nepoch,float eta){
    int i1,i2,i3,i4,i5;
    float ti,oi,delta,oj,vf1,vf2,**deltas;
    cListaConjuntoEntrenamiento1 *auxlce1;
    float ***g_k,***g_km1,***p_k,***p_km1,b_k,a_k=eta;

    deltas=(float **)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float *));
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        *(deltas+i1)=(float *)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float));
    }
    g_k=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
    g_km1=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
    p_k=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
    p_km1=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        *(g_k+i1)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float *));
        *(g_km1+i1)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float *));
        *(p_k+i1)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float *));
        *(p_km1+i1)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float *));
        for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
            *(*(g_k+i1)+i2)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i1)+1)*sizeof(float *));
            *(*(g_km1+i1)+i2)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i1)+1)*sizeof(float *));
            *(*(p_k+i1)+i2)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i1)+1)*sizeof(float *));
            *(*(p_km1+i1)+i2)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i1)+1)*sizeof(float *));
            for(i3=0;i3<=*(g1m_redneu1.redneu1_->t_+i1);i3++){
                *(*(*(g_k+i1)+i2)+i3)=0;
                *(*(*(g_km1+i1)+i2)+i3)=0;
                *(*(*(p_k+i1)+i2)+i3)=0;
                *(*(*(p_km1+i1)+i2)+i3)=0;
            }
        }
    }

    for(i1=0;i1<nepoch;i1++){
        for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                    *(*(*(g_k+i2)+i3)+i4)=0;
                }
            }
        }
        for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
            Propagacion1(i2);
            auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
            for(i3=g1m_redneu1.redneu1_->nc_-1;i3>=1;i3--){
                if(i3==g1m_redneu1.redneu1_->nc_-1){
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        ti=*(auxlce1->salida_+i4);
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=(ti-oi)*oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
                            vf1=delta*oj;
                            *(*(*(g_k+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
```

```
                        vf1=delta*oj;
                        *(*(*(g_k+i3-1)+i4)+i5)+=vf1;
                    }
                }
                else if(i3==1){
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=0;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
                            delta+=*(*(deltas+i3)+i5)*
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
                        }
                        delta=delta*oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(g1m_redneu1.redneu1_->valsr1_+i5);
                            vf1=delta*oj;
                            *(*(*(g_k+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(g_k+i3-1)+i4)+i5)+=vf1;
                    }
                }
                else{
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=0;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
                            delta+=*(*(deltas+i3)+i5)*
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
                        }
                        delta*=oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
                            vf1=delta*oj;
                            *(*(*(g_k+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(g_k+i3-1)+i4)+i5)+=vf1;
                    }
                }
            }
        }
        if(i1==0){
            for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                    for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                        *(*(*(p_k+i2)+i3)+i4)=*(*(*(g_k+i2)+i3)+i4);
                    }
                }
            }
        }
        else{
            vf1=0;vf2=0;
            for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                    for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                        vf1+=pow(*(*(*(g_k+i2)+i3)+i4),2);
                        vf2+=pow(*(*(*(g_km1+i2)+i3)+i4),2);
                    }
                }
            }
            if(vf2==0){
                b_k=vf1/.00001;
            }
            else{
                b_k=vf1/vf2;
            }
            for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                    for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                        *(*(*(p_k+i2)+i3)+i4)=*(*(*(g_k+i2)+i3)+i4)-b_k**(*(*(p_km1+i2)+i3)+i4);
                    }
                }
            }
        }
        /*line search*/{
            //a_k=otrabusqueda1(p_k);
        }
        for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                    *(*(*(g1m_redneu1.redneu1_->mpesc1_+i2)+i3)+i4)+=a_k**(*(*(p_k+i2)+i3)+i4);
                }
            }
        }
        for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
```

```
                    *(*(*(p_km1+i2)+i3)+i4)=*(*(*(p_k+i2)+i3)+i4);
                    *(*(*(g_km1+i2)+i3)+i4)=*(*(*(g_k+i2)+i3)+i4);
                }
            }
        }
    }

    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
            free(*(*(g_k+i1)+i2));
            free(*(*(g_km1+i1)+i2));
            free(*(*(p_k+i1)+i2));
            free(*(*(p_km1+i1)+i2));
        }
        free(*(g_k+i1));
        free(*(g_km1+i1));
        free(*(p_k+i1));
        free(*(p_km1+i1));
    }
    free(g_k);
    free(g_km1);
    free(p_k);
    free(p_km1);
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        free(*(deltas+i1));
    }
    free(deltas);
}

void EntrenamientoConLearnigRateVariable1(int nepoch){
    int i1,i2,i3,i4,i5;
    float ti,oi,delta,oj,vf1,vf2,**deltas;
    cListaConjuntoEntrenamiento1 *auxlce1;
    float ***pesos1,vf4;
    FILE *ar1=fopen("1.txt","w");

    deltas=(float **)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float *));
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        *(deltas+i1)=(float *)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float));
    }
    pesos1=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        *(pesos1+i1)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float *));
        for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
            *(*(pesos1+i1)+i2)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i1)+1)*sizeof(float *));
        }
    }

    for(i1=0;i1<nepoch;i1++){
        g1m_redneu1.conj1_->RetornaElmEnPosi(0,&vf1,&vf2);

        for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                    *(*(*(pesos1+i2)+i3)+i4)=0;
                }
            }
        }
        for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
            Propagacion1(i2);
            auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
            for(i3=g1m_redneu1.redneu1_->nc_-1;i3>=1;i3--){
                if(i3==g1m_redneu1.redneu1_->nc_-1){
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        ti=*(auxlce1->salida_+i4);
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=(ti-oi)*oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
                            vf1=delta*oj;
                            *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                    }
                }
                else if(i3==1){
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=0;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
                            delta+=*(*(deltas+i3)+i5)*
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
                        }
                        delta=delta*oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(g1m_redneu1.redneu1_->valsr1_+i5);
                            vf1=delta*oj;
                            *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
```

```
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                    }
                }
                else{
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=0;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
                            delta+=*(*(deltas+i3)+i5)*
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
                        }
                        delta*=oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
                            vf1=delta*oj;
                            *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesos1+i3-1)+i4)+i5)+=vf1;
                    }
                }
            }
        }
        /*Line search*/{
            vf4=otrabusqueda1(ar1,pesos1);
            //vf4=goldenSectionSearch(.1,2,3.9,.01,1000,pesos1);
        }
        for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
                    *(*(*(g1m_redneu1.redneu1_->mpesc1_+i2)+i3)+i4)+=vf4**(*(*(pesos1+i2)+i3)+i4);
                }
            }
        }
    }

    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
            free(*(*(pesos1+i1)+i2));
        }
        free(*(pesos1+i1));
    }
    free(pesos1);
    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        free(*(deltas+i1));
    }
    free(deltas);
    fclose(ar1);
}

void EntrenamientoGenetico1(int nit){
    int i1,i2,i3;
    float ****pesos;
    cListaConjuntoEntrenamiento1 *auxlce1;

    pesos=(float ****)malloc(6*sizeof(float ***));
    for(i1=0;i1<6;i1++){
        *(pesos+i1)=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
        for(i2=0;i2<(g1m_redneu1.redneu1_->nc_-1);i2++){
            *(*(pesos+i1)+i2)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i2+1)*sizeof(float *));
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
                *(*(*(pesos+i1)+i2)+i3)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i2)+1)*sizeof(float));
            }
        }
    }

    for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
        for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
            for(i3=0;i3<=*(g1m_redneu1.redneu1_->t_+i1);i3++){
                *(*(*(*(pesos+0)+i1)+i2)+i3)=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i1)+i2)+i3);
                *(*(*(*(pesos+1)+i1)+i2)+i3)=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i1)+i2)+i3);
            }
        }
    }

    for(i1=0;i1<nit;i1++){
        for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
            Propagacion1(i2);
            auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);


        }
    }

    for(i1=0;i1<6;i1++){
        for(i2=0;i2<(g1m_redneu1.redneu1_->nc_-1);i2++){
            for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
```

```
                    free(*(*(*(pesos+i1)+i2)+i3));
                }
                free(*(*(pesos+i1)+i2));
            }
            free(*(pesos+i1));
        }
        free(pesos);
    }
    DWORD WINAPI HiloLeeInformacionPadre(PVOID pParam){
        char cad1[100],cad2[200],cad3[100];
        int nleidos;
        int vi1,vi2,vi3,i1,i2,i3;
        float vf1,vf2,*apvf1,*apvf2;
        FILE *ar1;
        cListaConjuntoEntrenamiento1 *auxlce1;


        pParam=pParam;
        while(1){
            cad1[0]='\x0';
            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
            if(LeeBytesDeProcesoPadre1(g1m_procp.proc1,cad1,50*sizeof(char),&nleidos)==0){
                ReleaseMutex(g1m_procp.mutexleer);
                return 0;
            }
            else{
                ReleaseMutex(g1m_procp.mutexleer);
            }

            if(strcmp(cad1,"Salir")==0){
                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
                WaitForSingleObject(g1m_procp.mutexproc1,INFINITE);
                    CierraHandlesDeComunicacionConPadre(g1m_procp.proc1);
                ReleaseMutex(g1m_procp.mutexproc1);
                strcpy(cad1,"Salir");
                WaitForSingleObject(g1m_colapad1.mutexgral,INFINITE);
                    InsertaBytesEnCola1(g1m_colapad1.cola,cad1,20);
                ReleaseMutex(g1m_colapad1.mutexgral);
                return 0;
            }
            if(strcmp(cad1,"Cambiar Valor de Retina")==0){
                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
                ReleaseMutex(g1m_procp.mutexleer);
                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                    if(g1m_redneu1.redneu1_!=NULL){
                        if(vi1>=0 && vi1<*(g1m_redneu1.redneu1_->t_)){
                            *(g1m_redneu1.redneu1_->valsr1_+vi1)=vf1;
                        }
                    }
                ReleaseMutex(g1m_redneu1.mutexgral_);
            }
            if(strcmp(cad1,"Retornar Valor de Retina")==0){
                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
                ReleaseMutex(g1m_procp.mutexleer);
                vf1=RetornaNandf1();
                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                    if(g1m_redneu1.redneu1_!=NULL){
                        if(vi1>=0 && vi1<*(g1m_redneu1.redneu1_->t_)){
                            vf1=*(g1m_redneu1.redneu1_->valsr1_+vi1);
                        }
                    }
                ReleaseMutex(g1m_redneu1.mutexgral_);
                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
                    EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vf1,sizeof(float));
                ReleaseMutex(g1m_procp.mutexesribir);
            }
            if(strcmp(cad1,"Retornar numero de capas")==0){
                vi1=0;
                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                    if(g1m_redneu1.redneu1_!=NULL){
                        vi1=g1m_redneu1.redneu1_->nc_;
                    }
                ReleaseMutex(g1m_redneu1.mutexgral_);
                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
                    EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vi1,sizeof(int));
                ReleaseMutex(g1m_procp.mutexesribir);
            }
            if(strcmp(cad1,"Retornar tamano de capa")==0){
                vi2=0;
                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
                ReleaseMutex(g1m_procp.mutexleer);

                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                    if(g1m_redneu1.redneu1_!=NULL){
                        if(vi1>=0 && vi1<g1m_redneu1.redneu1_->nc_){
                            vi2=*(g1m_redneu1.redneu1_->t_+vi1);
                        }
```

```
1365                }
1366            ReleaseMutex(g1m_redneu1.mutexgral_);
1367            WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1368                EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vi2,sizeof(int));
1369            ReleaseMutex(g1m_procp.mutexesribir);
1370        }
1371        if(strcmp(cad1,"Iteracion Con Patron De Entrada")==0){
1372            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1373                apvf1=(float *)malloc(*g1m_redneu1.redneu1_->t_*sizeof(float));
1374                apvf2=(float *)malloc(*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1)*sizeof(float));
1375            ReleaseMutex(g1m_redneu1.mutexgral_);
1376
1377            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1378                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)apvf1,*g1m_redneu1.redneu1_->t_*sizeof(float),&nleidos);
1379            ReleaseMutex(g1m_procp.mutexleer);
1380
1381            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1382                if(g1m_redneu1.redneu1_!=NULL){
1383                    for(i1=0;i1<*(g1m_redneu1.redneu1_->t_);i1++){
1384                        *(g1m_redneu1.redneu1_->valsr1_+i1)=*(apvf1+i1);
1385                    }
1386                    for(i1=1;i1<g1m_redneu1.redneu1_->nc_;i1++){
1387                        if(i1==1){
1388                            for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1);i2++){
1389                                vf1=0;
1390                                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i1-1);i3++){
1391                                    vf1+=*(g1m_redneu1.redneu1_->valsr1_+i3)*
1392                                    *(*(*(g1m_redneu1.redneu1_->mpesc1_)+i2)+i3);
1393                                }
1394                                vf1+=*(g1m_redneu1.redneu1_->valsbias_+i1-1)*
1395                                *(*(*(g1m_redneu1.redneu1_->mpesc1_+i1-1)+i2)+i3);
1396                                *(*(g1m_redneu1.redneu1_->valsal1_)+i2)=1/(1+exp(-vf1));
1397                            }
1398                        }
1399                        else{
1400                            for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1);i2++){
1401                                vf1=0;
1402                                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i1-1);i3++){
1403                                    vf1+=*(*(g1m_redneu1.redneu1_->valsal1_+i1-2)+i3)*
1404                                    *(*(*(g1m_redneu1.redneu1_->mpesc1_+i1-1)+i2)+i3);
1405                                }
1406                                vf1+=*(g1m_redneu1.redneu1_->valsbias_+i1-1)*
1407                                *(*(*(g1m_redneu1.redneu1_->mpesc1_+i1-1)+i2)+i3);
1408                                *(*(g1m_redneu1.redneu1_->valsal1_+i1-1)+i2)=1/(1+exp(-vf1));
1409                            }
1410                        }
1411                    }
1412                    for(i1=0;i1<*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1);i1++){
1413                        *(apvf2+i1)=*(*(g1m_redneu1.redneu1_->valsal1_+g1m_redneu1.redneu1_->nc_-2)+i1);
1414                    }
1415                }
1416            ReleaseMutex(g1m_redneu1.mutexgral_);
1417            WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1418                EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)apvf2,*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1)*sizeof(float));
1419            ReleaseMutex(g1m_procp.mutexesribir);
1420            free(apvf1);
1421            free(apvf2);
1422        }
1423        if(strcmp(cad1,"Iteracion Con Patron De Entrada En GPU")==0){
1424            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
1425                apvf1=(float *)malloc(*g1m_redneu1.redneu1_->t_*sizeof(float));
1426            }ReleaseMutex(g1m_redneu1.mutexgral_);
1427            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);{
1428                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)apvf1,*g1m_redneu1.redneu1_->t_*sizeof(float),&nleidos);
1429            }ReleaseMutex(g1m_procp.mutexleer);
1430            WaitForSingleObject(g1m_colagpuops.mutexgral,INFINITE);{
1431                strcpy(cad2,"Propagar1");
1432                InsertaBytesEnCola1(g1m_colagpuops.cola,cad2,20);
1433                InsertaBytesEnColaPaquetes1(g1m_colagpuops.colapaq,apvf1,*g1m_redneu1.redneu1_->t_*sizeof(float));
1434            }ReleaseMutex(g1m_colagpuops.mutexgral);
1435            free(apvf1);
1436        }
1437
1438        if(strcmp(cad1,"Cambiar Valor de Peso")==0){
1439            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1440                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1441                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi2,sizeof(int),&nleidos);
1442                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi3,sizeof(int),&nleidos);
1443                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
1444            ReleaseMutex(g1m_procp.mutexleer);
1445
1446            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1447                if(g1m_redneu1.redneu1_!=NULL){
1448                    if(vi1>=1 && vi1<g1m_redneu1.redneu1_->nc_){
1449                        if(vi2>=0 && vi2<*(g1m_redneu1.redneu1_->t_+vi1)){
1450                            if(vi3>=0 && vi3<=*(g1m_redneu1.redneu1_->t_+vi1-1)){
1451                                *(*(*(g1m_redneu1.redneu1_->mpesc1_+vi1-1)+vi2)+vi3)=vf1;
1452                            }
1453                        }
1454                    }
1455                }
1456            ReleaseMutex(g1m_redneu1.mutexgral_);
```

```
1457            }
1458            if(strcmp(cad1,"Retornar Valor de Peso")==0){
1459                vf1=RetornaNandf1();
1460                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1461                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1462                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi2,sizeof(int),&nleidos);
1463                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi3,sizeof(int),&nleidos);
1464                ReleaseMutex(g1m_procp.mutexleer);
1465
1466                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1467                    if(g1m_redneu1.redneu1_!=NULL){
1468                        if(vi1>=1 && vi1<g1m_redneu1.redneu1_->nc_){
1469                            if(vi2>=0 && vi2<*(g1m_redneu1.redneu1_->t_+vi1)){
1470                                if(vi3>=0 && vi3<=*(g1m_redneu1.redneu1_->t_+vi1-1)){
1471                                    vf1=*(*(*(g1m_redneu1.redneu1_->mpesc1_+vi1-1)+vi2)+vi3);
1472                                }
1473                            }
1474                        }
1475                    }
1476                ReleaseMutex(g1m_redneu1.mutexgral_);
1477                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1478                    EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vf1,sizeof(float));
1479                ReleaseMutex(g1m_procp.mutexesribir);
1480            }
1481            if(strcmp(cad1,"Retornar Salida De Neurona")==0){
1482                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1483                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1484                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi2,sizeof(int),&nleidos);
1485                ReleaseMutex(g1m_procp.mutexleer);
1486                vf1=RetornaNandf1();
1487                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1488                    if(g1m_redneu1.redneu1_!=NULL){
1489                        if(vi1>=1 && vi1<g1m_redneu1.redneu1_->nc_){
1490                            if(vi2>=0 && vi2<*(g1m_redneu1.redneu1_->t_+vi1)){
1491                                vf1=*(*(g1m_redneu1.redneu1_->valsal1_+vi1-1)+vi2);
1492                            }
1493                        }
1494                    }
1495                ReleaseMutex(g1m_redneu1.mutexgral_);
1496                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1497                    EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vf1,sizeof(float));
1498                ReleaseMutex(g1m_procp.mutexesribir);
1499            }
1500            if(strcmp(cad1,"Retornar Salida De Neurona")==0){
1501                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1502                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1503                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi2,sizeof(int),&nleidos);
1504                ReleaseMutex(g1m_procp.mutexleer);
1505                vf1=RetornaNandf1();
1506                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1507                    if(g1m_redneu1.redneu1_!=NULL){
1508                        if(vi1>=1 && vi1<g1m_redneu1.redneu1_->nc_){
1509                            if(vi2>=0 && vi2<*(g1m_redneu1.redneu1_->t_+vi1)){
1510                                vf1=*(*(g1m_redneu1.redneu1_->valsal1_+vi1-1)+vi2);
1511                            }
1512                        }
1513                    }
1514                ReleaseMutex(g1m_redneu1.mutexgral_);
1515                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1516                    EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vf1,sizeof(float));
1517                ReleaseMutex(g1m_procp.mutexesribir);
1518            }
1519            if(strcmp(cad1,"Iniciar Pesos Aleatorios")==0){
1520                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1521                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1522                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
1523                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf2,sizeof(float),&nleidos);
1524                ReleaseMutex(g1m_procp.mutexleer);
1525                srand(vi1);
1526
1527                for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
1528                    for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
1529                        for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i1);i3++){
1530                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i1)+i2)+i3)= vf1+(vf2-vf1)*float(rand())/float(RAND_MAX);
1531                        }
1532                    }
1533                }
1534
1535            }
1536            if(strcmp(cad1,"Guardar Red En Archivo")==0){
1537                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1538                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1539                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,cad2,vi1*sizeof(char),&nleidos);
1540                ReleaseMutex(g1m_procp.mutexleer);
1541                srand(vi1);
1542
1543                ar1=fopen(cad2,"wb");
1544                strcpy(cad3,"mathgl_MLP_FC1m");
1545                fwrite(cad3,15*sizeof(char),1,ar1);
1546
1547                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1548                    fwrite(&(g1m_redneu1.redneu1_->nc_),sizeof(int),1,ar1);
```

```
1549
1550                for(i1=0;i1<g1m_redneu1.redneu1_->nc_;i1++){
1551                    fwrite(g1m_redneu1.redneu1_->t_+i1,sizeof(int),1,ar1);
1552                }
1553
1554                for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
1555                    for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
1556                        fwrite(*(*(g1m_redneu1.redneu1_->mpesc1_+i1)+i2),sizeof(float),*(g1m_redneu1.redneu1_->t_+i1)+1,ar1);
1557                    }
1558                }
1559                fclose(ar1);
1560            ReleaseMutex(g1m_redneu1.mutexgral_);
1561        }
1562
1563        if(strcmp(cad1,"Guardar Conjunto De Entrenamiento En Archivo")==0){
1564            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1565                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1566                LeeBytesDeProcesoPadre1(g1m_procp.proc1,cad2,vi1*sizeof(char),&nleidos);
1567            ReleaseMutex(g1m_procp.mutexleer);
1568            srand(vi1);
1569
1570            ar1=fopen(cad2,"wb");
1571            strcpy(cad3,"mathgl_MLP_FC1mTrainingSet");
1572            fwrite(cad3,sizeof(char),26,ar1);
1573
1574            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1575                fwrite(&g1m_redneu1.conj1_->te_,sizeof(int),1,ar1);
1576                fwrite(&g1m_redneu1.conj1_->ts_,sizeof(int),1,ar1);
1577                fwrite(&g1m_redneu1.conj1_->np_,sizeof(int),1,ar1);
1578                auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(0);
1579                for(i1=0;i1<g1m_redneu1.conj1_->np_;i1++){
1580                    fwrite(auxlce1->entrada_,sizeof(float),g1m_redneu1.conj1_->te_,ar1);
1581                    fwrite(auxlce1->salida_,sizeof(float),g1m_redneu1.conj1_->ts_,ar1);
1582                    auxlce1=auxlce1->sig_;
1583                }
1584            ReleaseMutex(g1m_redneu1.mutexgral_);
1585            fclose(ar1);
1586        }
1587        if(strcmp(cad1,"Agrgar Conjunto De Entrenamiento De Archivo")==0){
1588            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1589                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1590                LeeBytesDeProcesoPadre1(g1m_procp.proc1,cad2,vi1*sizeof(char),&nleidos);
1591            ReleaseMutex(g1m_procp.mutexleer);
1592            srand(vi1);
1593
1594            ar1=fopen(cad2,"rb");
1595            fread(cad3,sizeof(char),26,ar1);
1596            cad3[26]='\x0';
1597            if(strcmp("mathgl_MLP_FC1mTrainingSet",cad3)==0){
1598                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1599                    fread(&vi1,sizeof(int),1,ar1);
1600                    fread(&vi2,sizeof(int),1,ar1);
1601                    if(vi1==g1m_redneu1.conj1_->te_ && vi2==g1m_redneu1.conj1_->ts_){
1602                        fread(&vi1,sizeof(int),1,ar1);
1603                        apvf1=(float *)malloc(g1m_redneu1.conj1_->te_*sizeof(float));
1604                        apvf2=(float *)malloc(g1m_redneu1.conj1_->ts_*sizeof(float));
1605                        for(i1=0;i1<vi1;i1++){
1606                            fread(apvf1,sizeof(float),g1m_redneu1.conj1_->te_,ar1);
1607                            fread(apvf2,sizeof(float),g1m_redneu1.conj1_->ts_,ar1);
1608                            g1m_redneu1.conj1_->AgregaElmAlFinal(apvf1,apvf2);
1609                        }
1610                        free(apvf1);
1611                        free(apvf2);
1612                    }
1613                ReleaseMutex(g1m_redneu1.mutexgral_);
1614            }
1615            fclose(ar1);
1616        }
1617
1618        if(strcmp(cad1,"Entrenar red (congugado 1)")==0){
1619            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1620                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1621                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
1622            ReleaseMutex(g1m_procp.mutexleer);
1623
1624            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1625                EntrenamientoConjugado1(vi1,vf1);
1626            ReleaseMutex(g1m_redneu1.mutexgral_);
1627
1628            strcpy(cad2,"Listo");
1629            WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1630                EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1631            ReleaseMutex(g1m_procp.mutexesribir);
1632
1633        }
1634        if(strcmp(cad1,"Entrenar red (congugadoFR 1)")==0){
1635            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1636                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1637                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
1638            ReleaseMutex(g1m_procp.mutexleer);
1639
1640            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
```

```
1641                    EntrenamientoConjugadoFR1(vi1,vf1);
1642                ReleaseMutex(g1m_redneu1.mutexgral_);
1643
1644                strcpy(cad2,"Listo");
1645                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1646                    EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1647                ReleaseMutex(g1m_procp.mutexesribir);
1648
1649            }
1650            if(strcmp(cad1,"Entrenar red (simple 1)")==0){
1651                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1652                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1653                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
1654                ReleaseMutex(g1m_procp.mutexleer);
1655
1656                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1657                    Entrenamiento1(vi1,vf1);
1658                ReleaseMutex(g1m_redneu1.mutexgral_);
1659
1660                strcpy(cad2,"Listo");
1661                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1662                    EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1663                ReleaseMutex(g1m_procp.mutexesribir);
1664
1665            }
1666            if(strcmp(cad1,"Entrenar red (Con Momentum 1)")==0){
1667                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1668                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1669                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
1670                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf2,sizeof(float),&nleidos);
1671                ReleaseMutex(g1m_procp.mutexleer);
1672
1673                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1674                    EntrenamientoConMomentum1(vi1,vf1,vf2);
1675                ReleaseMutex(g1m_redneu1.mutexgral_);
1676
1677                strcpy(cad2,"Listo");
1678                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1679                    EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1680                ReleaseMutex(g1m_procp.mutexesribir);
1681
1682            }
1683            if(strcmp(cad1,"Entrenamiento3")==0){
1684                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1685                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1686                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(float),&nleidos);
1687                ReleaseMutex(g1m_procp.mutexleer);
1688
1689                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1690                    Entrenamiento3(vi1,vf1);
1691                ReleaseMutex(g1m_redneu1.mutexgral_);
1692
1693                strcpy(cad2,"Listo");
1694                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1695                    EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1696                ReleaseMutex(g1m_procp.mutexesribir);
1697
1698            }
1699            if(strcmp(cad1,"Entrenar red (variable LR1)")==0){
1700                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
1701                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1702                ReleaseMutex(g1m_procp.mutexleer);
1703
1704                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1705                    EntrenamientoConLearnigRateVariable1(vi1);
1706                ReleaseMutex(g1m_redneu1.mutexgral_);
1707
1708                strcpy(cad2,"Listo");
1709                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
1710                    EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1711                ReleaseMutex(g1m_procp.mutexesribir);
1712            }
1713            if(strcmp(cad1,"Entrenar red en GPU (variable LR1)")==0){
1714                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);{
1715                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vf1,sizeof(int),&nleidos);
1716                }ReleaseMutex(g1m_procp.mutexleer);
1717                WaitForSingleObject(g1m_colagpuops.mutexgral,INFINITE);{
1718                    strcpy(cad2,"Entrenar VLR1");
1719                    InsertaBytesEnCola1(g1m_colagpuops.cola,cad2,20);
1720                    InsertaBytesEnColaPaquetes1(g1m_colagpuops.colapaq,&vf1,sizeof(int));
1721                }ReleaseMutex(g1m_colagpuops.mutexgral);
1722            }
1723
1724            if(strcmp(cad1,"Insertar En Cunjunto De Entrenamiento Al Final")==0){
1725                WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
1726                    apvf1=(float *)malloc(*g1m_redneu1.redneu1_->t_*sizeof(float));
1727                    apvf2=(float *)malloc(*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1)*sizeof(float));
1728                    vi1=g1m_redneu1.conj1_->te_;
1729                    vi2=g1m_redneu1.conj1_->ts_;
1730                ReleaseMutex(g1m_redneu1.mutexgral_);
1731
1732                WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
```

```
                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)apvf1,vi1*sizeof(float),&nleidos);
                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)apvf2,vi2*sizeof(float),&nleidos);
            ReleaseMutex(g1m_procp.mutexleer);

            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                g1m_redneu1.conj1_->AgregaElmAlFinal(apvf1,apvf2);
            ReleaseMutex(g1m_redneu1.mutexgral_);
            free(apvf1);
            free(apvf2);
        }
        if(strcmp(cad1,"Retornar De Cunjunto De Entrenamiento En Pos")==0){

            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
            ReleaseMutex(g1m_procp.mutexleer);

            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                apvf1=(float *)malloc(*g1m_redneu1.redneu1_->t_*sizeof(float));
                apvf2=(float *)malloc(*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1)*sizeof(float));
                vi2=*g1m_redneu1.redneu1_->t_;
                vi3=*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1);
                g1m_redneu1.conj1_->RetornaElmEnPosi(vi1,apvf1,apvf2);
            ReleaseMutex(g1m_redneu1.mutexgral_);

            WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
                EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)apvf1,vi2*sizeof(float));
                EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)apvf2,vi3*sizeof(float));
            ReleaseMutex(g1m_procp.mutexesribir);

            free(apvf1);
            free(apvf2);
        }
        if(strcmp(cad1,"Borrar De Cunjunto De Entrenamiento En Pos")==0){
            WaitForSingleObject(g1m_procp.mutexleer,INFINITE);
                LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
            ReleaseMutex(g1m_procp.mutexleer);

            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                g1m_redneu1.conj1_->QuitarElmEnPosi(vi1);
            ReleaseMutex(g1m_redneu1.mutexgral_);
        }
        if(strcmp(cad1,"Retornar Numero De Cunjuntos De Entrenamiento")==0){
            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
                vi1=g1m_redneu1.conj1_->np_;
            ReleaseMutex(g1m_redneu1.mutexgral_);

            WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
                EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vi1,sizeof(int));
            ReleaseMutex(g1m_procp.mutexesribir);
        }
    }
    return 0;
}
DWORD WINAPI HiloEscribeInformacionAPadre1(PVOID pParam){
    char cad1[100],cad2[100];
    float vf1;

    pParam=pParam;
    while(1){
        cad1[0]='\x0';
        WaitForSingleObject(g1m_colapadsend1.mutexgral,INFINITE);
        if(RetornaNumDeElmsEnCola1(g1m_colapadsend1.cola)>=20){
            ExtraeBytesDeCola1(g1m_colapadsend1.cola,(void *)cad1,20);
            ReleaseMutex(g1m_colapadsend1.mutexgral);
            if(strcmp(cad1,(char *)"Salir De Hilo")==0){
                return 0;
            }
            if(strcmp(cad1,(char *)"Salida Del GPU 1")==0){
                WaitForSingleObject(g1m_colapadsend1.mutexgral,INFINITE);
                    ExtraeBytesDeColaPaquetes1(g1m_colapadsend1.colapaq,(void *)&vf1);
                ReleaseMutex(g1m_colapadsend1.mutexgral);
                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
                    EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&vf1,sizeof(float));
                ReleaseMutex(g1m_procp.mutexesribir);
            }
            if(strcmp(cad1,(char *)"Envia Listo")==0){
                strcpy(cad2,"Listo");
                WaitForSingleObject(g1m_procp.mutexesribir,INFINITE);
                    EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&cad2,50*sizeof(char));
                ReleaseMutex(g1m_procp.mutexesribir);
            }
        }
        else{
            ReleaseMutex(g1m_colapadsend1.mutexgral);
        }
        Sleep(100);
    }
    return 0;
}
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){
    HWND vent1;
    MSG msg;
```

```
1825        BOOL bQuit = FALSE;
1826        char cad1[100],cad2[200];
1827        HANDLE hanleleer1,hanleescribir1;
1828        int i1,i2,vi1,*veap1,nleidos,*apvi1;
1829        FILE *ar1;
1830
1831        hPrevInstance=hPrevInstance;
1832        lpCmdLine=lpCmdLine;
1833        {
1834            {
1835                g1m_redneu1.mutexgral_=CreateMutex(0,FALSE,0);
1836                g1m_redneu1.redneu1_=NULL;
1837            }
1838            {
1839                g1m_procp.proc1=CreaConexionConProcesoPadre1(GetStdHandle(STD_INPUT_HANDLE),GetStdHandle(STD_OUTPUT_HANDLE));
1840                g1m_procp.mutexesribir=CreateMutex(0,FALSE,0);
1841                g1m_procp.mutexleer=CreateMutex(0,FALSE,0);
1842                g1m_procp.mutexproc1=CreateMutex(0,FALSE,0);
1843            }
1844            {
1845                g1m_colapad1.cola=CreaCola1();
1846                g1m_colapad1.mutexgral=CreateMutex(0,FALSE,0);
1847            }
1848            {
1849                g1m_colapadsend1.colapaq=CreaColaPquetes1();
1850                g1m_colapadsend1.cola=CreaCola1();
1851                g1m_colapadsend1.mutexgral=CreateMutex(0,FALSE,0);
1852            }
1853            {
1854                g1m_colagpuops.colapaq=CreaColaPquetes1();
1855                g1m_colagpuops.cola=CreaCola1();
1856                g1m_colapadsend1.mutexgral=CreateMutex(0,FALSE,0);
1857
1858            }
1859            {
1860                LeeBytesDeProcesoPadre1(g1m_procp.proc1,cad1,50*sizeof(char),&nleidos);
1861                if(strcmp("Crear Red Con Parametros Dados",cad1)==0){
1862                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1863                    veap1=(int *)malloc(vi1*sizeof(int));
1864                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)veap1,vi1*sizeof(int),&nleidos);
1865                    g1m_redneu1.redneu1_=new cRedNeu1m(vi1,veap1);
1866                    g1m_redneu1.conj1_=new cManConjuntoEntrenamiento1(*veap1,*(veap1+vi1-1));
1867                    free(veap1);
1868                }
1869                else if(strcmp(cad1,"Crear Red Desde Archivo")==0){
1870                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,(char *)&vi1,sizeof(int),&nleidos);
1871                    LeeBytesDeProcesoPadre1(g1m_procp.proc1,cad2,vi1*sizeof(char),&nleidos);
1872
1873                    ar1=fopen(cad2,"rb");
1874                    fread(cad2,sizeof(char),15,ar1);
1875                    cad2[15]='\x0';
1876                    if(strcmp("mathgl_MLP_FC1m",cad2)!=0){
1877                        bQuit=TRUE;
1878                        strcpy(cad2,"No Se Puede");
1879                        EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1880                    }
1881                    else{
1882
1883                        strcpy(cad2,"Si Se Puede");
1884                        EscribeBytesAProcesoPadre(g1m_procp.proc1,cad2,50*sizeof(char));
1885
1886                        fread(&vi1,sizeof(int),1,ar1);
1887                        apvi1=(int *)malloc(vi1*sizeof(int));
1888                        fread(apvi1,sizeof(int),vi1,ar1);
1889                        g1m_redneu1.redneu1_=new cRedNeu1m(vi1,apvi1);
1890                        g1m_redneu1.conj1_=new cManConjuntoEntrenamiento1(*apvi1,*(apvi1+vi1-1));
1891
1892
1893                        for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
1894                            for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
1895                                fread(*(*(g1m_redneu1.redneu1_->mpesc1_+i1)+i2),sizeof(float),*(g1m_redneu1.redneu1_->t_+i1)+1,ar1);
1896                            }
1897                        }
1898                        EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)&(g1m_redneu1.redneu1_->nc_),sizeof(int));
1899                        EscribeBytesAProcesoPadre(g1m_procp.proc1,(char *)g1m_redneu1.redneu1_->t_,g1m_redneu1.redneu1_->nc_*sizeof(int));
1900                    }
1901                    fclose(ar1);
1902                }
1903            }
1904            hanleleer1=CreateThread(NULL,0,HiloLeeInformacionPadre,(PVOID)NULL,0,NULL);
1905            hanleescribir1=CreateThread(NULL,0,HiloEscribeInformacionAPadre1,(PVOID)NULL,0,NULL);
1906            g1m_MutexUsoOpg1=CreateMutex(0,FALSE,0);
1907        }
1908        creaclaseventanas1(hInstance);
1909        vent1 = CreateWindowEx(0,"GLSample","MathGPU Motor MLP_FC1m",WS_OVERLAPPED|WS_CAPTION|WS_CLIPCHILDREN|WS_VISIBLE,100,10,648,530,NULL,NULL,hInst
1910        ShowWindow(vent1,nCmdShow);
1911
1912        while (!bQuit){
1913            if (PeekMessage(&msg,NULL,0,0,PM_REMOVE)){
1914                if (msg.message == WM_QUIT){
1915                    bQuit = TRUE;
1916                }
```

```
1917                else{
1918                    TranslateMessage(&msg);
1919                    DispatchMessage(&msg);
1920                }
1921            }
1922            else{
1923                WaitForSingleObject(g1m_colapad1.mutexgral,INFINITE);
1924                if(RetornaNumDeElmsEnCola1(g1m_colapad1.cola)>=20){
1925                    ExtraeBytesDeCola1(g1m_colapad1.cola,(void *)cad1,20);
1926                    ReleaseMutex(g1m_colapad1.mutexgral);
1927                    if(strcmp(cad1,(char *)"Salir")==0){
1928                        bQuit = TRUE;
1929                    }
1930                }
1931                else{
1932                    ReleaseMutex(g1m_colapad1.mutexgral);
1933                }
1934                Sleep(100);
1935            }
1936        }
1937        DestroyWindow(vent1);
1938        {
1939            strcpy(cad1,"Salir De Hilo");
1940            WaitForSingleObject(g1m_colapadsend1.mutexgral,INFINITE);
1941                InsertaBytesEnCola1(g1m_colapadsend1.cola,cad1,20);
1942            ReleaseMutex(g1m_colapadsend1.mutexgral);
1943
1944            WaitForSingleObject(hanleescribir1,INFINITE);
1945            WaitForSingleObject(hanleleer1,INFINITE);
1946
1947            {
1948                DestruyeCola1(g1m_colagpuops.cola);
1949                DestruyeColaPaquetes1(g1m_colagpuops.colapaq);
1950                CloseHandle(g1m_colapadsend1.mutexgral);
1951            }
1952            {
1953                DestruyeColaPaquetes1(g1m_colapadsend1.colapaq);
1954                DestruyeCola1(g1m_colapadsend1.cola);
1955                CloseHandle(g1m_colapadsend1.mutexgral);
1956            }
1957            {
1958                CloseHandle(g1m_procp.mutexleer);
1959                CloseHandle(g1m_procp.mutexesribir);
1960                CloseHandle(g1m_procp.mutexproc1);
1961                DestruyeEstructuraProcesoPadre1(g1m_procp.proc1);
1962            }
1963            {
1964                CloseHandle(g1m_redneu1.mutexgral_);
1965                if(g1m_redneu1.redneu1_!=NULL){
1966                    delete g1m_redneu1.redneu1_;
1967                }
1968                if(g1m_redneu1.conj1_!=NULL){
1969                    delete g1m_redneu1.conj1_;
1970                }
1971            }
1972            CloseHandle(g1m_MutexUsoOpg1);
1973        }
1974        return msg.wParam;
1975    }
1976    LRESULT CALLBACK WindowProc(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam){
1977        static HMENU hMenu,hMenuPopup1;
1978        static HWND ventopg1;
1979        enum {
1980                MENU_PROPIEDADES_PROPIEDADES1,
1981                MENU_PROPIEDADES_PROPIEDADES2,
1982        };
1983        switch (message){
1984            case WM_CREATE:{
1985                hMenu=CreateMenu();{
1986                    hMenuPopup1=CreateMenu();
1987                        AppendMenu(hMenuPopup1,MF_STRING,MENU_PROPIEDADES_PROPIEDADES1,"Propiedades 1");
1988                        AppendMenu(hMenuPopup1,MF_STRING,MENU_PROPIEDADES_PROPIEDADES2,"Propiedades 2");
1989                    AppendMenu(hMenu,MF_POPUP,(UINT_PTR)hMenuPopup1,"Propiedades");
1990                }
1991                ventopg1=CreateWindowEx(0,"visor3d1","",WS_BORDER|WS_CHILD|WS_VISIBLE,10,50,300,300,hwnd,(HMENU)0,((LPCREATESTRUCT)lParam)->hInstance,N
1992                UpdateWindow(hwnd);
1993                return 0;
1994            }
1995            case WM_DESTROY:{
1996                return 0;
1997            }
1998            case WM_COMMAND:{
1999                switch (lParam){
2000                    case 0:{
2001                        switch(HIWORD(wParam)){
2002                            case 0:{
2003                                switch(LOWORD(wParam)){
2004                                    case MENU_PROPIEDADES_PROPIEDADES1:{
2005                                        break;
2006                                    }
2007                                    case MENU_PROPIEDADES_PROPIEDADES2:{
2008                                        break;
```

```
                                }
                                default:{
                                    break;
                                }
                            }
                            break;
                        }
                        case 1:{
                            break;
                        }
                        default:{
                            break;
                        }
                    }
                    break;
                }
                default:{
                    break;
                }
            }
            return 0;
        }
        default:{
            return DefWindowProc(hwnd,message,wParam,lParam);
        }
    }
    return 0;
}

void EnableOpenGL(HWND hwnd, HDC* hDC, HGLRC* hRC){
    PIXELFORMATDESCRIPTOR pfd;
    int iFormat;
    *hDC = GetDC(hwnd);
    ZeroMemory(&pfd, sizeof(pfd));
    pfd.nSize = sizeof(pfd);
    pfd.nVersion = 1;
    pfd.dwFlags = PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
    pfd.iPixelType = PFD_TYPE_RGBA;
    pfd.cColorBits = 24;
    pfd.cDepthBits = 16;
    pfd.iLayerType = PFD_MAIN_PLANE;
    iFormat = ChoosePixelFormat(*hDC, &pfd);
    SetPixelFormat(*hDC, iFormat, &pfd);
    *hRC = wglCreateContext(*hDC);
    wglMakeCurrent(*hDC, *hRC);
}
void DisableOpenGL(HWND hwnd, HDC hDC, HGLRC hRC){
    wglMakeCurrent(NULL, NULL);
    wglDeleteContext(hRC);
    ReleaseDC(hwnd, hDC);
}

void PropagacionGPU1(float *apvf1){
    char shaderfuente1[5000];
/*aqui esta el shader incompleto*/{
    strcpy(shaderfuente1,"\n\
uniform sampler2DRect pesos;\n\
uniform int tamlpes;\n\
uniform sampler2DRect entradas;\n\
uniform int tamlentrada;\n\
uniform int ncapas;\n\
uniform int tamcapas[];\n\
\n\
float g1m_vi1,g1m_vi2,g1m_vi3,g1m_vi4;\n\
float g1m_salidas1[],g1m_salidas2[];\n\
\n\
float retpeso1(int ncapa,int nneu,int npes){\n\
    float vf1=texture(pesos,vec2(g1m_vi1,g1m_vi2)).x;\n\
    g1m_vi1=g1m_vi1+1;\n\
    g1m_vi2=g1m_vi2+1;\n\
    if(g1m_vi1>=tamlpes){\n\
        g1m_vi2=g1m_vi2+1;\n\
        g1m_vi1=0;\n\
    }\n\
    return vf1;\n\
}\n\
\n\
float retent1(int nent){\n\
    float vf1=texture(entradas,vec2(g1m_vi3,g1m_vi4)).x;\n\
    g1m_vi3=g1m_vi3+1;\n\
    g1m_vi4=g1m_vi4+1;\n\
    if(g1m_vi3>=tamlentrada){\n\
        g1m_vi4=g1m_vi4+1;\n\
        g1m_vi3=0;\n\
    }\n\
    if(nent>=tamcapas[0]-1){\n\
        g1m_vi4=0;\n\
        g1m_vi3=0;\n\
    }\n\
    return vf1;\n\
}\n\
\n\
```

```
2101  void main(){\n\
2102      int i1,i2,i3;\n\
2103      for(i1=0;i1<ncapas-1;i1++){\n\
2104          if(i1==0){\n\
2105              for(i2=0;i2<tamcapas[i1+1];i2++){\n\
2106                  g1m_salidas2[i2]=0;\n\
2107                  for(i3=0;i3<tamcapas[i1];i3++){\n\
2108                      g1m_salidas2[i2]=g1m_salidas2[i2]+retpeso1(i1+1,i2,i3)*retent1(i3);\n\
2109                  }\n\
2110                  g1m_salidas2[i2]=1/(1+exp(-g1m_salidas2[i2]));\n\
2111              }\n\
2112          }\n\
2113          else{\n\
2114              for(i2=0;i2<tamcapas[i1+1];i2++){\n\
2115                  g1m_salidas2[i2]=0;\n\
2116                  for(i3=0;i3<tamcapas[i1];i3++){\n\
2117                      g1m_salidas2[i2]=g1m_salidas2[i2]+retpeso1(i1+1,i2,i3)*g1m_salidas1[i3];\n\
2118                  }\n\
2119                  g1m_salidas2[i2]=1/(1+exp(-g1m_salidas2[i2]));\n\
2120              }\n\
2121          }\n\
2122          for(i2=0;i2<tamcapas[i1];i2++){\n\
2123              g1m_salidas1[i2]=g1m_salidas2[i2];\n\
2124          }\n\
2125      }\n\
2126      gl_FragColor=vec4(g1m_salidas2[0]);\n\
2127  }\n\
2128      ");
2129  }
2130      char *shaderfuente2,cad1[100],cad2[100],cad3[100];
2131      int i1,i2,i3,i4,vi1,vi2=0,vi3=0;
2132      float vf1,tamltexpesos1,tamltexentrada;
2133      float *matpesos1,*matentrada;
2134      GLuint fb,texmatbuff[2],texpesos1,texentradas1;
2135      GLint tmax1[2];
2136      void *shader1;
2137      GLint locvars;
2138
2139      /*completa el shader*/{
2140          WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2141              vf1=(float)g1m_redneu1.redneu1_->nc_;
2142          }ReleaseMutex(g1m_redneu1.mutexgral_);
2143          fcvt1m(vf1,cad1,0);
2144
2145          vi1=*(g1m_redneu1.redneu1_->t_+1);
2146          for(i1=2;i1<g1m_redneu1.redneu1_->nc_;i1++){
2147              vi2=*(g1m_redneu1.redneu1_->t_+i1);
2148              if(vi2>vi1){
2149                  vi1=vi2;
2150              }
2151          }
2152          vf1=(float)vi1;
2153          fcvt1m(vf1,cad2,0);
2154          vi1=strlen(shaderfuente1)+strlen(cad1)+2*strlen(cad2)+1;
2155          shaderfuente2=(char *)malloc(vi1*sizeof(char));
2156
2157          i1=0;i2=1;
2158          while(*(shaderfuente1+i1)!='\x0' && i2==1){
2159              for(i3=0;i3<8;i3++){
2160                  cad3[i3]=*(shaderfuente1+i1+i3);
2161              }
2162              cad3[i3]='\x0';
2163              if(strcmp("tamcapas",cad3)==0){
2164                  vi1=i1+i3+1;
2165                  i2=0;
2166              }
2167              else{
2168                  i1++;
2169              }
2170          }
2171          shaderfuente1[vi1]='\x0';
2172          strcpy(shaderfuente2,shaderfuente1);
2173          strcat(shaderfuente2,cad1);
2174          shaderfuente1[vi1]=']';
2175
2176          i1=0;i2=1;
2177          while(*(shaderfuente1+i1)!='\x0' && i2==1){
2178              for(i3=0;i3<12;i3++){
2179                  cad3[i3]=*(shaderfuente1+i1+i3);
2180              }
2181              cad3[i3]='\x0';
2182              if(strcmp("g1m_salidas1",cad3)==0){
2183                  vi2=i1+i3+1;
2184                  i2=0;
2185              }
2186              else{
2187                  i1++;
2188              }
2189          }
2190          shaderfuente1[vi2]='\x0';
2191          strcat(shaderfuente2,shaderfuente1+vi1);
2192          strcat(shaderfuente2,cad2);
```

```
2193            shaderfuente1[vi2]='];
2194
2195            i1=0;i2=1;
2196            while(*(shaderfuente1+i1)!='\x0' && i2==1){
2197                for(i3=0;i3<12;i3++){
2198                    cad3[i3]=*(shaderfuente1+i1+i3);
2199                }
2200                cad3[i3]='\x0';
2201                if(strcmp("g1m_salidas2",cad3)==0){
2202                    vi3=i1+i3+1;
2203                    i2=0;
2204                }
2205                else{
2206                    i1++;
2207                }
2208            }
2209            shaderfuente1[vi3]='\x0';
2210            strcat(shaderfuente2,shaderfuente1+vi2);
2211            strcat(shaderfuente2,cad2);
2212            shaderfuente1[vi3]=']';
2213
2214            strcat(shaderfuente2,shaderfuente1+vi3);
2215        }
2216
2217        vi1=0;
2218        WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2219            for(i1=1;i1<g1m_redneu1.redneu1_->nc_;i1++){
2220                vi1=vi1+*(g1m_redneu1.redneu1_->t_+i1)*(*(g1m_redneu1.redneu1_->t_+i1-1)+1);
2221            }
2222        }ReleaseMutex(g1m_redneu1.mutexgral_);
2223
2224        tamltexpesos1=(int)ceilf(sqrt((float)vi1));
2225        WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2226            vi1=*(g1m_redneu1.redneu1_->t_);
2227        }ReleaseMutex(g1m_redneu1.mutexgral_);
2228        tamltexentrada=(int)ceilf(sqrt((float)vi1));
2229
2230        matpesos1=(float *)malloc(tamltexpesos1*tamltexpesos1*sizeof(float));
2231        matentrada=(float *)malloc(tamltexentrada*tamltexentrada*sizeof(float));
2232        i4=0;
2233        WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2234            for(i1=1;i1<g1m_redneu1.redneu1_->nc_;i1++){
2235                for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1);i2++){
2236                    for(i3=0;i3<=*(g1m_redneu1.redneu1_->t_+i1-1);i3++){
2237                        *(matpesos1+i4)=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i1-1)+i2)+i3);
2238                        i4++;
2239                    }
2240                }
2241            }
2242        }ReleaseMutex(g1m_redneu1.mutexgral_);
2243
2244        i4=0;
2245        WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2246            for(i1=0;i1<*(g1m_redneu1.redneu1_->t_);i1++){
2247                *(matentrada+i4)=*(apvf1+i4);
2248            }
2249        }ReleaseMutex(g1m_redneu1.mutexgral_);
2250
2251        /*Prepara la vision*/{
2252            glMatrixMode(GL_PROJECTION);
2253            glLoadIdentity();
2254            gluOrtho2D(0,1,0,1);
2255            glMatrixMode(GL_MODELVIEW);
2256            glLoadIdentity();
2257            glViewport(0,0,1,1);
2258        }
2259        /*Aqui se preparan las texturas del render*/{
2260            glGenFramebuffersEXT(1,&fb);
2261            glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,fb);
2262            glFinish();
2263            glGenTextures(2,texmatbuff);
2264            glFinish();
2265            {
2266                glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,1,1,0,GL_RGBA,GL_FLOAT,0);
2267                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2268                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2269                glFinish();
2270                if(tmax1[0]!=1 || tmax1[1]!=1){
2271                    glDeleteTextures(2,texmatbuff);
2272                    return ;
2273                }
2274                glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[0]);
2275                glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,1,1,0,GL_RGBA,GL_FLOAT,0);
2276                glFinish();
2277                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2278                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2279                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2280                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2281                glFinish();
2282            }
2283            {
2284                glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,1,1,0,GL_RGBA,GL_FLOAT,0);
```

```
2285                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2286                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2287                glFinish();
2288                if(tmax1[0]!=1 || tmax1[1]!=1){
2289                    glDeleteTextures(2,texmatbuff);
2290                    return ;
2291                }
2292                glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[1]);
2293                glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,1,1,0,GL_RGBA,GL_FLOAT,0);
2294                glFinish();
2295                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2296                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2297                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2298                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2299                glFinish();
2300            }
2301            glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[0]);
2302            glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,GL_COLOR_ATTACHMENT0_EXT,GL_TEXTURE_RECTANGLE,texmatbuff[0],0);
2303            glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[1]);
2304            glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,GL_DEPTH_ATTACHMENT_EXT,GL_TEXTURE_RECTANGLE,texmatbuff[1],0);
2305            glFinish();
2306        }
2307        /*envia el vector de pesos*/{
2308            glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_LUMINANCE_FLOAT32_ATI,tamltexpesos1,tamltexpesos1,0,GL_RED,GL_FLOAT,0);
2309            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2310            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2311            glFinish();
2312            if(tmax1[0]!=tamltexpesos1 || tmax1[1]!=tamltexpesos1){
2313                glDeleteTextures(2,texmatbuff);
2314                return ;
2315            }
2316            glGenTextures(1,&texpesos1);
2317            glFinish();
2318            glBindTexture(GL_TEXTURE_RECTANGLE,texpesos1);
2319            glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_LUMINANCE_FLOAT32_ATI,tamltexpesos1,tamltexpesos1,0,GL_RED,GL_FLOAT,matpesos1);
2320            glFinish();
2321            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2322            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2323            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2324            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2325            glFinish();
2326        }
2327        /*envia el vector de entradas*/{
2328            glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_LUMINANCE_FLOAT32_ATI,tamltexentrada,tamltexentrada,0,GL_RED,GL_FLOAT,0);
2329            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2330            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2331            glFinish();
2332            if(tmax1[0]!=tamltexentrada || tmax1[1]!=tamltexentrada){
2333                glDeleteTextures(2,texmatbuff);
2334                glDeleteTextures(1,&texpesos1);
2335                return ;
2336            }
2337            glGenTextures(1,&texentradas1);
2338            glFinish();
2339            glBindTexture(GL_TEXTURE_RECTANGLE,texentradas1);
2340            glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_LUMINANCE_FLOAT32_ATI,tamltexentrada,tamltexentrada,0,GL_RED,GL_FLOAT,matpesos1);
2341            glFinish();
2342            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2343            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2344            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2345            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2346            glFinish();
2347        }
2348
2349        shader1=CreaCargaPraparaShaderDeCadena1(NULL,shaderfuente2);
2350        CorrerShader1(shader1);
2351        glFinish();
2352
2353        /*Envia las variables internas*/{
2354            glActiveTexture(GL_TEXTURE1);
2355            glBindTexture(GL_TEXTURE_RECTANGLE,texpesos1);
2356            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"pesos");
2357            glUniform1i(locvars,1);
2358            glFinish();
2359
2360            glActiveTexture(GL_TEXTURE2);
2361            glBindTexture(GL_TEXTURE_RECTANGLE,texentradas1);
2362            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"entradas");
2363            glUniform1i(locvars,2);
2364            glFinish();
2365
2366            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamlpes");
2367            glUniform1i(locvars,tamltexpesos1);
2368            glFinish();
2369
2370            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamlentrada");
2371            glUniform1i(locvars,tamltexentrada);
2372            glFinish();
2373
2374            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"ncapas");
2375            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2376                glUniform1i(locvars,g1m_redneu1.redneu1_->nc_);
```

```
2377            }ReleaseMutex(g1m_redneu1.mutexgral_);
2378            glFinish();
2379
2380            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamcapas");
2381            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2382                glUniform1iv(locvars,g1m_redneu1.redneu1_->nc_,g1m_redneu1.redneu1_->t_);
2383            }ReleaseMutex(g1m_redneu1.mutexgral_);
2384            glFinish();
2385        }
2386
2387        /*Hace computo*/{
2388            glBegin(GL_QUADS);
2389                glTexCoord2f(0,0);    glVertex2f(0,0);
2390                glTexCoord2f(1,0);    glVertex2f(1,0);
2391                glTexCoord2f(1,1);    glVertex2f(1,1);
2392                glTexCoord2f(0,1);    glVertex2f(0,1);
2393            glEnd();
2394            glFinish();
2395        }
2396
2397        /*recibe datos*/{
2398            glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[0]);
2399            glGetTexImage(GL_TEXTURE_RECTANGLE,0,GL_RED,GL_FLOAT,&vf1);
2400            glFinish();
2401        }
2402
2403        strcpy(cad1,"Salida Del GPU 1");
2404        WaitForSingleObject(g1m_colapadsend1.mutexgral,INFINITE);{
2405            InsertaBytesEnCola1(g1m_colapadsend1.cola,(void *)cad1,20);
2406            InsertaBytesEnColaPaquetes1(g1m_colapadsend1.colapaq,(void *)&vf1,sizeof(float));
2407        }ReleaseMutex(g1m_colapadsend1.mutexgral);
2408
2409        /*Terminar cosas*/{
2410            NoCorrerShader1(shader1);
2411            DestruyeShader1(shader1);
2412            glDeleteFramebuffersEXT(1,&fb);
2413            glFinish();
2414            /*Elimina las texturas*/{
2415                glDeleteTextures(1,&texentradas1);
2416                glDeleteTextures(1,&texpesos1);
2417                glDeleteTextures(2,texmatbuff);
2418                glFinish();
2419            }
2420        }
2421
2422        free(matpesos1);
2423        free(matentrada);
2424    }
2425    void EntrenamientoConLearnigRateVariableEnGPU1(int nepoch){
2426        char shaderfuente1[10000],*shaderfuente2;
2427        float vf1;
2428        char cad1[100],cad2[100],cad3[100],cad4[100];
2429        int i1,i2,i3,i4,i5,vi1=0,vi2=0,vi3=0,vi4=0,vi5=0;
2430        int tamltexpesos1,tamltexentrada,tamldeltas,tamlsaldes;
2431        float *matpesos,*matpesoserr,*matentradas,*matdeltas,*matsaldes,*materrores1;
2432        float ***pesoserr,**deltas,delta,oj,ti,oi,vf4=0,vf5,*auxfvp1;
2433        cListaConjuntoEntrenamiento1 *auxlce1;
2434        GLuint fb,texmatbuff[2],texpesos1,texpesoserror1,texentradas1,texsaldes,texdeltas;
2435        GLint tmax1[2],locvars;
2436        void *shader1;
2437        FILE *ar1=fopen("10.txt","w");
2438    /*aqui esta el shader incompleto*/{
2439        strcpy(shaderfuente1,"\n\
2440    uniform sampler2DRect pesos,errpesos;\n\
2441    uniform float tamlpes;\n\
2442    uniform sampler2DRect entradas;\n\
2443    uniform float tamlentradas,nentradas;\n\
2444    uniform sampler2DRect salidasdes;\n\
2445    uniform float tamlsalidas;\n\
2446    uniform sampler2DRect deltas;\n\
2447    uniform float tamldeltas;\n\
2448    uniform float ncapas;\n\
2449    uniform float tamcapas[];\n\
2450    \n\
2451    void main(){\n\
2452        float g1m_salidas1[],g1m_salidas2[],g1m_retina[];\n\
2453        int i1,i2,i3,i4;\n\
2454        float delta,error,g1m_appes,g1m_apent,g1m_apsaldes,vf1;\n\
2455        float auxvf1,auxvf2;\n\
2456    \n\
2457        g1m_appes=0;\n\
2458        g1m_apent=0;\n\
2459        g1m_apsaldes=0;\n\
2460        delta=texture(deltas,gl_TexCoord[0].st).x;\n\
2461        error=0;\n\
2462    \n\
2463        for(i4=0;i4<int(nentradas);i4++){\n\
2464            g1m_appes=0;\n\
2465            for(i1=0;i1<tamcapas[0];i1++){\n\
2466                g1m_salidas1[i1]=texture(entradas,vec2(mod(g1m_apent+float(i1),tamlentradas),floor((g1m_apent+.1+float(i1))/tamlentradas))).x;\n\
2467            }\n\
2468            g1m_apent+=tamcapas[0];\n\
```

```
2469            for(i1=0;i1<ncapas-1;i1++){\n\
2470                for(i2=0;i2<tamcapas[i1+1];i2++){\n\
2471                    vf1=0;\n\
2472                    for(i3=0;i3<tamcapas[i1];i3++){\n\
2473                        auxvf1=g1m_appes+float(i3);\n\
2474                        auxvf2=auxvf1+.1;\n\
2475                        vf1+=g1m_salidas1[i3]*(texture(pesos,vec2(mod(auxvf1,tamlpes),floor(auxvf2/tamlpes))).x+delta*texture(errpesos,vec2(mod(auxvf1,
2476                    }\n\
2477                    auxvf1=g1m_appes+float(i3);\n\
2478                    auxvf2=auxvf1+.1;\n\
2479                    vf1+=(-1)*(texture(pesos,vec2(mod(auxvf1,tamlpes),floor(auxvf2/tamlpes))).x+delta*texture(errpesos,vec2(mod(auxvf1,tamlpes),floor(a
2480                    g1m_appes+=float(tamcapas[i1])+1;\n\
2481                    g1m_salidas2[i2]=1/(1+exp(-vf1));\n\
2482                }\n\
2483                for(i2=0;i2<tamcapas[i1+1];i2++){\n\
2484                    g1m_salidas1[i2]=g1m_salidas2[i2];\n\
2485                }\n\
2486            }\n\
2487            error+=pow(g1m_salidas1[0]-texture(salidasdes,vec2(mod(g1m_apsaldes,tamlsalidas),floor((g1m_apsaldes+.1)/tamlsalidas))).x,2);\n\
2488            g1m_apsaldes+=1;\n\
2489        }\n\
2490        gl_FragColor=vec4(error,0,0,0);\n\
2491    }\n\
2492        ");
2493    }
2494
2495        /*completa el shader*/{
2496            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
2497                vf1=(float)g1m_redneu1.redneu1_->nc_;
2498            }ReleaseMutex(g1m_redneu1.mutexgral_);
2499            fcvt1m(vf1,cad1,0);
2500
2501            vi1=*(g1m_redneu1.redneu1_->t_+1);
2502            for(i1=2;i1<g1m_redneu1.redneu1_->nc_;i1++){
2503                vi2=*(g1m_redneu1.redneu1_->t_+i1);
2504                if(vi2>vi1){
2505                    vi1=vi2;
2506                }
2507            }
2508            vf1=(float)vi1;
2509            fcvt1m(vf1,cad2,0);
2510
2511            vi1=*(g1m_redneu1.redneu1_->t_);
2512            vf1=(float)vi1;
2513            fcvt1m(vf1,cad4,0);
2514
2515            vi1=strlen(shaderfuente1)+strlen(cad1)+2*strlen(cad2)+strlen(cad4)+1000;
2516            shaderfuente2=(char *)malloc(vi1*sizeof(char));
2517
2518            i1=0;i2=1;
2519            while(*(shaderfuente1+i1)!='\x0' && i2==1){
2520                for(i3=0;i3<8;i3++){
2521                    cad3[i3]=*(shaderfuente1+i1+i3);
2522                }
2523                cad3[i3]='\x0';
2524                if(strcmp("tamcapas",cad3)==0){
2525                    vi1=i1+i3+1;
2526                    i2=0;
2527                }
2528                else{
2529                    i1++;
2530                }
2531            }
2532            shaderfuente1[vi1]='\x0';
2533            strcpy(shaderfuente2,shaderfuente1);
2534            strcat(shaderfuente2,cad1);
2535            shaderfuente1[vi1]=']';
2536
2537            i1=0;i2=1;
2538            while(*(shaderfuente1+i1)!='\x0' && i2==1){
2539                for(i3=0;i3<12;i3++){
2540                    cad3[i3]=*(shaderfuente1+i1+i3);
2541                }
2542                cad3[i3]='\x0';
2543                if(strcmp("g1m_salidas1",cad3)==0){
2544                    vi2=i1+i3+1;
2545                    i2=0;
2546                }
2547                else{
2548                    i1++;
2549                }
2550            }
2551            shaderfuente1[vi2]='\x0';
2552            strcat(shaderfuente2,shaderfuente1+vi1);
2553            strcat(shaderfuente2,cad2);
2554            shaderfuente1[vi2]=']';
2555
2556            i1=0;i2=1;
2557            while(*(shaderfuente1+i1)!='\x0' && i2==1){
2558                for(i3=0;i3<12;i3++){
2559                    cad3[i3]=*(shaderfuente1+i1+i3);
2560                }
```

```
            cad3[i3]='\x0';
            if(strcmp("g1m_salidas2",cad3)==0){
                vi3=i1+i3+1;
                i2=0;
            }
            else{
                i1++;
            }
        }
        shaderfuente1[vi3]='\x0';
        strcat(shaderfuente2,shaderfuente1+vi2);
        strcat(shaderfuente2,cad2);
        shaderfuente1[vi3]=']';

        i1=0;i2=1;
        while(*(shaderfuente1+i1)!='\x0' && i2==1){
            for(i3=0;i3<10;i3++){
                cad3[i3]=*(shaderfuente1+i1+i3);
            }
            cad3[i3]='\x0';
            if(strcmp("g1m_retina",cad3)==0){
                vi4=i1+i3+1;
                i2=0;
            }
            else{
                i1++;
            }
        }
        shaderfuente1[vi4]='\x0';
        strcat(shaderfuente2,shaderfuente1+vi3);
        strcat(shaderfuente2,cad4);
        shaderfuente1[vi4]=']';

        strcat(shaderfuente2,shaderfuente1+vi4);
    }

    FILE *ar2=fopen("100.txt","w");
    fprintf(ar2,"%s",shaderfuente2);
    fclose(ar2);

    WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
        vi1=0;
        for(i1=1;i1<g1m_redneu1.redneu1_->nc_;i1++){
            vi1=vi1+*(g1m_redneu1.redneu1_->t_+i1)*(*(g1m_redneu1.redneu1_->t_+i1-1)+1);
        }
    }ReleaseMutex(g1m_redneu1.mutexgral_);
    tamltexpesos1=(int)ceilf(sqrtf((float)vi1));
    WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);{
        vi1=*(g1m_redneu1.redneu1_->t_)*g1m_redneu1.conj1_->np_;
    }ReleaseMutex(g1m_redneu1.mutexgral_);
    tamltexentrada=(int)ceilf(sqrt((float)vi1));
    tamldeltas=10;
    vi1=*(g1m_redneu1.redneu1_->t_+g1m_redneu1.redneu1_->nc_-1)*g1m_redneu1.conj1_->np_;
    tamlsaldes=(int)ceilf(sqrtf((float)vi1));

    matpesos=(float *)malloc(tamltexpesos1*tamltexpesos1*sizeof(float));
    matpesoserr=(float *)malloc(tamltexpesos1*tamltexpesos1*sizeof(float));
    matdeltas=(float *)malloc(tamldeltas*tamldeltas*sizeof(float));
    materrores1=(float *)malloc(tamldeltas*tamldeltas*sizeof(float));
    matentradas=(float *)malloc(tamltexentrada*tamltexentrada*sizeof(float));
    matsaldes=(float *)malloc(tamlsaldes*tamlsaldes*sizeof(float));

    i3=0;
    for(i1=0;i1<g1m_redneu1.conj1_->np_;i1++){
        auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i1);
        for(i2=0;i2<g1m_redneu1.conj1_->te_;i2++){
            *(matentradas+i3)=*(auxlce1->entrada_+i2);
            i3++;
        }
    }
    i3=0;
    for(i1=0;i1<g1m_redneu1.conj1_->np_;i1++){
        auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i1);
        for(i2=0;i2<g1m_redneu1.conj1_->ts_;i2++){
            *(matsaldes+i3)=*(auxlce1->salida_+i2);
            i3++;
        }
    }
    for(i1=0;i1<tamldeltas;i1++){
        for(i2=0;i2<tamldeltas;i2++){
            *(matdeltas+i1*tamldeltas+i2)=(float)i1+((float)i2)/10.;
        }
    }

    /*Prepara la vision*/{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,tamldeltas,0,tamldeltas);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glViewport(0,0,tamldeltas,tamldeltas);
    }
```

```
2653        /*Aqui se preparan las texturas del render*/{
2654            glGenFramebuffersEXT(1,&fb);
2655            glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,fb);
2656            glFinish();
2657            glGenTextures(2,texmatbuff);
2658            glFinish();
2659            {
2660                glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamldeltas,tamldeltas,0,GL_RGBA,GL_FLOAT,0);
2661                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2662                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2663                glFinish();
2664                if(tmax1[0]!=tamldeltas || tmax1[1]!=tamldeltas){
2665                    glDeleteTextures(2,texmatbuff);
2666                    return ;
2667                }
2668                glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[0]);
2669                glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamldeltas,tamldeltas,0,GL_RGBA,GL_FLOAT,0);
2670                glFinish();
2671                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2672                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2673                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2674                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2675                glFinish();
2676            }
2677            {
2678                glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamldeltas,tamldeltas,0,GL_RGBA,GL_FLOAT,0);
2679                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2680                glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2681                glFinish();
2682                if(tmax1[0]!=tamldeltas || tmax1[1]!=tamldeltas){
2683                    glDeleteTextures(2,texmatbuff);
2684                    return ;
2685                }
2686                glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[1]);
2687                glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamldeltas,tamldeltas,0,GL_RGBA,GL_FLOAT,0);
2688                glFinish();
2689                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2690                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2691                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2692                glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2693                glFinish();
2694            }
2695            glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[0]);
2696            glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,GL_COLOR_ATTACHMENT0_EXT,GL_TEXTURE_RECTANGLE,texmatbuff[0],0);
2697            glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[1]);
2698            glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,GL_DEPTH_ATTACHMENT_EXT,GL_TEXTURE_RECTANGLE,texmatbuff[1],0);
2699            glFinish();
2700        }
2701        /*envia el vector de pesos*/{
2702            glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamltexpesos1,tamltexpesos1,0,GL_RED,GL_FLOAT,0);
2703            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2704            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2705            glFinish();
2706            if(tmax1[0]!=tamltexpesos1 || tmax1[1]!=tamltexpesos1){
2707                glDeleteTextures(2,texmatbuff);
2708                return ;
2709            }
2710            glGenTextures(1,&texpesos1);
2711            glFinish();
2712            glBindTexture(GL_TEXTURE_RECTANGLE,texpesos1);
2713            glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamltexpesos1,tamltexpesos1,0,GL_RED,GL_FLOAT,0);
2714            glFinish();
2715            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2716            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2717            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2718            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2719            glFinish();
2720        }
2721        /*envia el vector de pesos de error*/{
2722            glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamltexpesos1,tamltexpesos1,0,GL_RED,GL_FLOAT,0);
2723            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2724            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2725            glFinish();
2726            if(tmax1[0]!=tamltexpesos1 || tmax1[1]!=tamltexpesos1){
2727                glDeleteTextures(2,texmatbuff);
2728                return ;
2729            }
2730            glGenTextures(1,&texpesoserror1);
2731            glFinish();
2732            glBindTexture(GL_TEXTURE_RECTANGLE,texpesoserror1);
2733            glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamltexpesos1,tamltexpesos1,0,GL_RED,GL_FLOAT,0);
2734            glFinish();
2735            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2736            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2737            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2738            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2739            glFinish();
2740        }
2741        /*envia el vector de entradas*/{
2742            glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamltexentrada,tamltexentrada,0,GL_RED,GL_FLOAT,0);
2743            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2744            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
```

```
2745            glFinish();
2746            if(tmax1[0]!=tamltexentrada || tmax1[1]!=tamltexentrada){
2747                glDeleteTextures(2,texmatbuff);
2748                glDeleteTextures(1,&texpesos1);
2749                return ;
2750            }
2751            glGenTextures(1,&texentradas1);
2752            glFinish();
2753            glBindTexture(GL_TEXTURE_RECTANGLE,texentradas1);
2754            glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamltexentrada,tamltexentrada,0,GL_RED,GL_FLOAT,matentradas);
2755            glFinish();
2756            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2757            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2758            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2759            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2760            glFinish();
2761        }
2762        /*envia el vector de salidas*/{
2763            glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamlsaldes,tamlsaldes,0,GL_RED,GL_FLOAT,0);
2764            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2765            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2766            glFinish();
2767            if(tmax1[0]!=tamlsaldes || tmax1[1]!=tamlsaldes){
2768                glDeleteTextures(2,texmatbuff);
2769                glDeleteTextures(1,&texpesos1);
2770                return ;
2771            }
2772            glGenTextures(1,&texsaldes);
2773            glFinish();
2774            glBindTexture(GL_TEXTURE_RECTANGLE,texsaldes);
2775            glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamlsaldes,tamlsaldes,0,GL_RED,GL_FLOAT,matsaldes);
2776            glFinish();
2777            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2778            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2779            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2780            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2781            glFinish();
2782        }
2783        /*envia el vector de los deltas*/{
2784            glTexImage2D(GL_PROXY_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamldeltas,tamldeltas,0,GL_RED,GL_FLOAT,0);
2785            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_WIDTH,&tmax1[0]);
2786            glGetTexLevelParameteriv(GL_PROXY_TEXTURE_RECTANGLE,0,GL_TEXTURE_HEIGHT,&tmax1[1]);
2787            glFinish();
2788            if(tmax1[0]!=tamldeltas || tmax1[1]!=tamldeltas){
2789                glDeleteTextures(2,texmatbuff);
2790                glDeleteTextures(1,&texpesos1);
2791                return ;
2792            }
2793            glGenTextures(1,&texdeltas);
2794            glFinish();
2795            glBindTexture(GL_TEXTURE_RECTANGLE,texdeltas);
2796            glTexImage2D(GL_TEXTURE_RECTANGLE,0,GL_RGBA_FLOAT32_ATI,tamldeltas,tamldeltas,0,GL_RED,GL_FLOAT,matdeltas);
2797            glFinish();
2798            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
2799            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
2800            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_S,GL_CLAMP);
2801            glTexParameteri(GL_TEXTURE_RECTANGLE,GL_TEXTURE_WRAP_T,GL_CLAMP);
2802            glFinish();
2803        }
2804
2805        shader1=CreaCargaPraparaShaderDeCadena1(NULL,shaderfuente2);
2806        CorrerShader1(shader1);
2807        glFinish();
2808
2809        /*Envia las variables internas*/{
2810            /*Envia texturas*/{
2811                glActiveTexture(GL_TEXTURE1);
2812                glBindTexture(GL_TEXTURE_RECTANGLE,texpesos1);
2813                locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"pesos");
2814                glUniform1i(locvars,1);
2815                glFinish();
2816
2817                glActiveTexture(GL_TEXTURE2);
2818                glBindTexture(GL_TEXTURE_RECTANGLE,texpesoserror1);
2819                locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"errpesos");
2820                glUniform1i(locvars,2);
2821                glFinish();
2822
2823                glActiveTexture(GL_TEXTURE3);
2824                glBindTexture(GL_TEXTURE_RECTANGLE,texentradas1);
2825                locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"entradas");
2826                glUniform1i(locvars,3);
2827                glFinish();
2828
2829                glActiveTexture(GL_TEXTURE4);
2830                glBindTexture(GL_TEXTURE_RECTANGLE,texsaldes);
2831                locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"salidasdes");
2832                glUniform1i(locvars,4);
2833                glFinish();
2834
2835                glActiveTexture(GL_TEXTURE5);
2836                glBindTexture(GL_TEXTURE_RECTANGLE,texdeltas);
```

```
2837                locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"deltas");
2838                glUniform1i(locvars,5);
2839                glFinish();
2840            }
2841
2842            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamlpes");
2843            glUniform1f(locvars,(float)tamltexpesos1);
2844            glFinish();
2845
2846            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamlentradas");
2847            glUniform1f(locvars,(float)tamltexentrada);
2848            glFinish();
2849
2850            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"nentradas");
2851            glUniform1f(locvars,(float)g1m_redneu1.conj1_->np_);
2852            glFinish();
2853
2854            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamlsalidas");
2855            glUniform1f(locvars,(float)tamlsaldes);
2856            glFinish();
2857
2858            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamldeltas");
2859            glUniform1f(locvars,(float)tamldeltas);
2860            glFinish();
2861
2862            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"ncapas");
2863            glUniform1f(locvars,(float)g1m_redneu1.redneu1_->nc_);
2864            glFinish();
2865
2866            auxfvp1=(float *)malloc(g1m_redneu1.redneu1_->nc_*sizeof(float));
2867            for(i1=0;i1<g1m_redneu1.redneu1_->nc_;i1++){
2868                *(auxfvp1+i1)=(float)*(g1m_redneu1.redneu1_->t_+i1);
2869            }
2870            locvars=glGetUniformLocation(RetornaIdentificadorPrograma1(shader1),"tamcapas");
2871            glUniform1fv(locvars,g1m_redneu1.redneu1_->nc_,auxfvp1);
2872            glFinish();
2873            free(auxfvp1);
2874        }
2875
2876        deltas=(float **)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float *));
2877        for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
2878            *(deltas+i1)=(float *)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float));
2879        }
2880        pesoserr=(float ***)malloc((g1m_redneu1.redneu1_->nc_-1)*sizeof(float **));
2881        for(i1=0;i1<g1m_redneu1.redneu1_->nc_-1;i1++){
2882            *(pesoserr+i1)=(float **)malloc(*(g1m_redneu1.redneu1_->t_+i1+1)*sizeof(float *));
2883            for(i2=0;i2<*(g1m_redneu1.redneu1_->t_+i1+1);i2++){
2884                *(*(pesoserr+i1)+i2)=(float *)malloc((*(g1m_redneu1.redneu1_->t_+i1)+1)*sizeof(float *));
2885            }
2886        }
2887        for(i1=0;i1<nepoch;i1++){
2888            for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
2889                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
2890                    for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
2891                        *(*(*(pesoserr+i2)+i3)+i4)=0;
2892                    }
2893                }
2894            }
2895            for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
2896                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
2897                    *(*(deltas+i2)+*(g1m_redneu1.redneu1_->t_+i2+1))=0;
2898                }
2899            }
2900            for(i2=0;i2<g1m_redneu1.conj1_->np_;i2++){
2901                Propagacion1(i2);
2902                auxlce1=g1m_redneu1.conj1_->RetornaEstCompletaEnPosi(i2);
2903                for(i3=g1m_redneu1.redneu1_->nc_-1;i3>=1;i3--){
2904                    if(i3==g1m_redneu1.redneu1_->nc_-1){
2905                        for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
2906                            ti=*(auxlce1->salida_+i4);
2907                            oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
2908                            delta=(ti-oi)*oi*(1-oi);
2909                            *(*(deltas+i3-1)+i4)=delta;
2910                            for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
2911                                oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
2912                                vf1=delta*oj;
2913                                *(*(*(pesoserr+i3-1)+i4)+i5)+=vf1;
2914                            }
2915                            oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
2916                            vf1=delta*oj;
2917                            *(*(*(pesoserr+i3-1)+i4)+i5)+=vf1;
2918                        }
2919                    }
2920                    else if(i3==1){
2921                        for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
2922                            oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
2923                            delta=0;
2924                            for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
2925                                delta+=*(*(deltas+i3)+i5)*
2926                                *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
2927                            }
2928                            delta=delta*oi*(1-oi);
```

```
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(g1m_redneu1.redneu1_->valsr1_+i5);
                            vf1=delta*oj;
                            *(*(*(pesoserr+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesoserr+i3-1)+i4)+i5)+=vf1;
                    }
                }
                else{
                    for(i4=0;i4<*(g1m_redneu1.redneu1_->t_+i3);i4++){
                        oi=*(*(g1m_redneu1.redneu1_->valsal1_+i3-1)+i4);
                        delta=0;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3+1);i5++){
                            delta+=*(*(deltas+i3)+i5)*
                            *(*(*(g1m_redneu1.redneu1_->mpesc1_+i3)+i5)+i4);
                        }
                        delta*=oi*(1-oi);
                        *(*(deltas+i3-1)+i4)=delta;
                        for(i5=0;i5<*(g1m_redneu1.redneu1_->t_+i3-1);i5++){
                            oj=*(*(g1m_redneu1.redneu1_->valsal1_+i3-2)+i5);
                            vf1=delta*oj;
                            *(*(*(pesoserr+i3-1)+i4)+i5)+=vf1;
                        }
                        oj=*(g1m_redneu1.redneu1_->valsbias_+i3-1);
                        vf1=delta*oj;
                        *(*(*(pesoserr+i3-1)+i4)+i5)+=vf1;
                    }
                }
            }
        }
        /*search*/{
            i5=0;
            for(i2=1;i2<g1m_redneu1.redneu1_->nc_;i2++){
                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2);i3++){
                    for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2-1);i4++){
                        *(matpesos+i5)=*(*(*(g1m_redneu1.redneu1_->mpesc1_+i2-1)+i3)+i4);
                        i5++;
                    }
                }
            }
            i5=0;
            for(i2=1;i2<g1m_redneu1.redneu1_->nc_;i2++){
                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2);i3++){
                    for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2-1);i4++){
                        *(matpesoserr+i5)=*(*(*(pesoserr+i2-1)+i3)+i4);
                        i5++;
                    }
                }
            }

            glActiveTexture(GL_TEXTURE1);
            glBindTexture(GL_TEXTURE_RECTANGLE,texpesos1);
            glFinish();
            glTexSubImage2D(GL_TEXTURE_RECTANGLE,0,0,0,tamltexpesos1,tamltexpesos1,GL_RED,GL_FLOAT,matpesos);
            glFinish();

            glActiveTexture(GL_TEXTURE2);
            glBindTexture(GL_TEXTURE_RECTANGLE,texpesoserror1);
            glFinish();
            glTexSubImage2D(GL_TEXTURE_RECTANGLE,0,0,0,tamltexpesos1,tamltexpesos1,GL_RED,GL_FLOAT,matpesoserr);
            glFinish();
            glClearColor(0,0,0,0);
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            glBegin(GL_QUADS);{
                glTexCoord2f(0,0);                      glVertex2f(0,0);
                glTexCoord2f(tamldeltas,0);             glVertex2f(tamldeltas,0);
                glTexCoord2f(tamldeltas,tamldeltas);    glVertex2f(tamldeltas,tamldeltas);
                glTexCoord2f(0,tamldeltas);             glVertex2f(0,tamldeltas);
            }glEnd();
            glFinish();

            glActiveTexture(GL_TEXTURE0);
            glBindTexture(GL_TEXTURE_RECTANGLE,texmatbuff[0]);
            glGetTexImage(GL_TEXTURE_RECTANGLE,0,GL_RED,GL_FLOAT,materrores1);
            glFinish();
            vf4=*(materrores1+1);
            vf5=.1;
            for(i2=0;i2<tamldeltas;i2++){
                for(i3=0;i3<tamldeltas;i3++){
                    if(*(materrores1+i2*tamldeltas+i3)<vf4){
                        vf4=*(materrores1+i2*tamldeltas+i3);
                        vf5=float(i2)+float(i3)/10.;
                    }
                }
            }
            if(abs(vf5)<.1){
                vf5=.1;
            }
        }
```

```
3021            for(i2=0;i2<g1m_redneu1.redneu1_->nc_-1;i2++){
3022                for(i3=0;i3<*(g1m_redneu1.redneu1_->t_+i2+1);i3++){
3023                    for(i4=0;i4<=*(g1m_redneu1.redneu1_->t_+i2);i4++){
3024                        *(*(*(g1m_redneu1.redneu1_->mpesc1_+i2)+i3)+i4)+=vf5**(*(*(pesoserr+i2)+i3)+i4);
3025                    }
3026                }
3027            }
3028        }
3029
3030        /*Terminar cosas*/{
3031            NoCorrerShader1(shader1);
3032            DestruyeShader1(shader1);
3033            glDeleteFramebuffersEXT(1,&fb);
3034            glFinish();
3035            /*Elimina las texturas*/{
3036                glDeleteTextures(1,&texentradas1);
3037                glDeleteTextures(1,&texpesos1);
3038                glDeleteTextures(1,&texpesoserror1);
3039                glDeleteTextures(1,&texsaldes);
3040                glDeleteTextures(1,&texdeltas);
3041                glDeleteTextures(2,texmatbuff);
3042                glFinish();
3043            }
3044        }
3045
3046        free(matpesos);
3047        free(matpesoserr);
3048        free(matdeltas);
3049        free(matentradas);
3050        free(matsaldes);
3051        fclose(ar1);
3052    }
3053
3054    void __cdecl ThreadProc1(void *pParam){
3055        cEstHiloVisor3d1 *h1;
3056        int i1=1,vi1;
3057        char cad1[100],cad2[100];
3058        float *apvf1;
3059
3060        h1=(cEstHiloVisor3d1 *)pParam;
3061        i1=1;
3062        while(i1==1){
3063            WaitForSingleObject(h1->mgral1, INFINITE);
3064                if(h1->est1[0]==0){
3065                    i1=0;
3066                }
3067            ReleaseMutex(h1->mgral1);
3068            if(i1!=0){
3069                WaitForSingleObject(g1m_MutexUsoOpg1,INFINITE);
3070                    WaitForSingleObject(h1->mgral1,INFINITE);{
3071                        wglMakeCurrent(h1->hDC1,h1->hRC1);
3072                    }ReleaseMutex(h1->mgral1);
3073                    WaitForSingleObject(g1m_colagpuops.mutexgral,INFINITE);
3074                    if(RetornaNumDeElmsEnCola1(g1m_colagpuops.cola)>=20){
3075                        ExtraeBytesDeCola1(g1m_colagpuops.cola,(void *)cad1,20);
3076                        ReleaseMutex(g1m_colagpuops.mutexgral);
3077                        if(strcmp(cad1,(char *)"Propagar1")==0){
3078                            WaitForSingleObject(g1m_redneu1.mutexgral_,INFINITE);
3079                                apvf1=(float *)malloc(*(g1m_redneu1.redneu1_->t_)*sizeof(float));
3080                            ReleaseMutex(g1m_redneu1.mutexgral_);
3081                            WaitForSingleObject(g1m_colagpuops.mutexgral,INFINITE);
3082                                ExtraeBytesDeColaPaquetes1(g1m_colagpuops.colapaq,apvf1);
3083                            ReleaseMutex(g1m_colagpuops.mutexgral);
3084                            PropagacionGPU1(apvf1);
3085                            free(apvf1);
3086                        }
3087                        if(strcmp(cad1,(char *)"Entrenar VLR1")==0){
3088                            WaitForSingleObject(g1m_colagpuops.mutexgral,INFINITE);
3089                                ExtraeBytesDeColaPaquetes1(g1m_colagpuops.colapaq,&vi1);
3090                            ReleaseMutex(g1m_colagpuops.mutexgral);
3091                            EntrenamientoConLearnigRateVariableEnGPU1(vi1);
3092                            strcpy(cad2,"Envia Listo");
3093                            WaitForSingleObject(g1m_colapadsend1.mutexgral,INFINITE);{
3094                                InsertaBytesEnCola1(g1m_colapadsend1.cola,(void *)cad2,20);
3095                            }ReleaseMutex(g1m_colapadsend1.mutexgral);
3096                        }
3097                    }
3098                    else{
3099                        ReleaseMutex(g1m_colagpuops.mutexgral);
3100                    }
3101                    wglMakeCurrent(NULL,NULL);
3102                ReleaseMutex(g1m_MutexUsoOpg1);
3103            }
3104            Sleep(100);
3105        }
3106
3107
3108    }
3109    LRESULT CALLBACK WindowProc2(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam){
3110        static cEstHiloVisor3d1 esth1;
3111        static HDC hdc;
3112        static HGLRC hrc;
```

```
3113        static HANDLE h1;
3114
3115        switch (message){
3116            case WM_CREATE:{
3117                EnableOpenGL (hwnd,&hdc,&hrc);
3118                wglMakeCurrent(hdc,hrc);
3119                    glewInit();
3120                    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
3121                    glClearColor(1,1,1,0);
3122                    glShadeModel(GL_SMOOTH);
3123                    glPointSize(1);
3124                    glEnable(GL_DEPTH_TEST);
3125                    glFinish();
3126                wglMakeCurrent(NULL,NULL);
3127                esth1.hDC1=hdc;
3128                esth1.hRC1=hrc;
3129                esth1.mgral1=CreateMutex(0,FALSE,0);
3130                esth1.est1[0]=1;
3131                h1=(void *)_beginthread (ThreadProc1,0,(void *)&esth1);
3132                SetThreadPriority(h1,THREAD_PRIORITY_LOWEST);
3133                return 0;
3134            }
3135            case WM_DESTROY:{
3136                WaitForSingleObject(esth1.mgral1,INFINITE);
3137                    esth1.est1[0]=0;
3138                ReleaseMutex(esth1.mgral1);
3139                WaitForSingleObject(h1,INFINITE);
3140                CloseHandle(esth1.mgral1);
3141                CloseHandle(h1);
3142                DisableOpenGL(hwnd,hdc,hrc);
3143                return 0;
3144            }
3145            default:{
3146                return DefWindowProc(hwnd, message, wParam, lParam);
3147            }
3148        }
3149        return 0;
3150    }
3151
3152    int creaclaseventanas1(HINSTANCE hInstance){
3153        WNDCLASSEX wcex;
3154        {
3155            wcex.cbSize = sizeof(WNDCLASSEX);
3156            wcex.style = CS_OWNDC;
3157            wcex.lpfnWndProc = WindowProc;
3158            wcex.cbClsExtra = 0;
3159            wcex.cbWndExtra = 0;
3160            wcex.hInstance = hInstance;
3161            wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
3162            wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
3163            wcex.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
3164            wcex.lpszMenuName = NULL;
3165            wcex.lpszClassName = "GLSample";
3166            wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
3167            if (!RegisterClassEx(&wcex)){
3168                return 0;
3169            }
3170        }
3171        {
3172            wcex.cbSize = sizeof(WNDCLASSEX);
3173            wcex.style = CS_OWNDC;
3174            wcex.lpfnWndProc = WindowProc2;
3175            wcex.cbClsExtra = 0;
3176            wcex.cbWndExtra = 0;
3177            wcex.hInstance = hInstance;
3178            wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
3179            wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
3180            wcex.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
3181            wcex.lpszMenuName = NULL;
3182            wcex.lpszClassName = "visor3d1";
3183            wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
3184            if (!RegisterClassEx(&wcex)){
3185                return 0;
3186            }
3187        }
3188        return 1;
3189    }
```

BIBLIOGRAFÍA