VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA
DE LA COMPUTACIÓN

ANÁLISIS DE LA PARALELIZACIÓN DE RAY TRACING

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
PAULINA MENDOZA MONROY

TUTOR:
DR. JORGE LUIS ORTEGA ARJONA
FACULTAD DE CIENCIAS, UNAM.

MÉXICO, D. F. AGOSTO 2015

Agradecimientos

A mi mamá por todo el apoyo, cariño, esfuerzo diario y dedicación.
A mi hermana por creer siempre en mi y ser mi guía de vida.
A Oscar por ser mi impulso diario.
A mi gran amigo y compañero Edgar en especial por hacerme ver mis errores.
A Arturo, gracias siempre.
A mi tutor, principalmete por la paciencia.
A la UNAM, mi segunda casa.
A CONACYT.

Índice general

Índice de figuras

RESUMEN

En computación gráfica, un reto importante es lograr que la apariencia de las imágenes se vea tan real como sea posible. Una técnica para obtener una imagen que luzca real es a través de la iluminación. Una desventaja de los programas de iluminación es que, a mayor detalle en la escena, mayor es el tiempo de procesamiento. Como ejemplo, el algoritmo de iluminación Ray Tracing, el cual es capaz de entregar resultados tan buenos como sea posible dependiendo de la capacidad de procesamiento, provocando en algunas ocasiones que el tiempo de ejecución se incremente considerablemente.

Debido al tiempo que requieren los programas de iluminación para alcanzar resultados con mayor detalle, se han estudiado varias técnicas para hacerlos más eficientes. Algunas técnicas se enfocan en procesar intersecciónes de manera más rápida, como *Object bounding volumes* o *Bounding volumes hierarchies*; otras usan métodos estadísticos para reducir la cantidad de rayos; algunas otras utilizan modificaciones para trazar el rayo, como *beam*, *cone* y *pencil tracing*; también se utilizan diversos métodos de sombreado que hacen más rápido el proceso. Una técnica que se ocupa frecuentemente en programas que requieren mayor cantidad de procesamiento y en la que se basa este trabajo, es desarrollar un sistema de red que puede ser concurrente, distribuido o paralelo, debido a que se realiza una división de la carga de trabajo.

Existe variedad de trabajos sobre paralelización que ayudan a definir si llevar a cabo un trabajo en este ámbito es necesario. Tal es el caso del artículo *Is Parallel for you?* de Cherry Pancake [7], el cual plantea estrategias para determinar si el algoritmo es candidato a paralelizarse. Otros trabajos han desarrollado técnicas que facilitan la labor de paralelizar, como los Patrones de Diseño de Software Paralelo [1]. Estos patrones hacen hincapié en el diseño de la coordinación, comunicación y sincronización de los procesos, ya que estos factores son los principales responsables de que un programa paralelo se ejecute en mayor tiempo que uno secuencial.

Los Patrones de Diseño son una base importante para realizar la paralelización de Ray Tracing. Con un análisis detallado de los elementos que intervienen se puede asegurar un buen diseño y desempeño y que la ejecución del programa Ray Tracing paralelo se realize en menos tiempo que el secuencial. La selección del Patrón de Software se realiza mediante las características del programa que se va a paralelizar. Una vez que se identifica el patrón que se adecúa a las características del algoritmo, se toman en cuenta las opciones que ofrece el patrón para llevar a cabo la sincronización, comunicación y coordinación de los procesadores, dependiendo de la infraestructura con la que se cuenta. Un aspecto más a considerar al momento de realizar un trabajo de paralelización, es la distribución de la carga. El patrón se encarga de determinar como será la forma de interactuar entre los elementos participantes, pero también es importante determinar como es que se hara la distribución de trabajo entre estos componentes. La idea es que todos los componentes tengan cargas iguales con el objetivo de que terminen su ejecución en tiempos similares.

En Ray Tracing la repartición de trabajo se puede hacer mediante la división de la imagen entre los procesadores. Una forma de repartir la información de una imagen equitativamente, es definir parámetros para medir la complejidad de la porción de imagen que le corresponde a cada procesador. Una vez que se miden los parámetros para cada porción de imagen, se pueden considerar estas medidas para poder repartir la carga de trabajo.

El fin de este trabajo es hacer un análisis de la paralelización del algoritmo con base en métodos de software paralelo, que ayuden a hacer más sencilla esta tarea, así como proponer una distribución que mejore la repartición de carga entre los procesadores participantes, tomando en cuenta la complejidad de la imagen.

Capítulo 1

Introducción

1.1. Contexto

Computación gráfica es la descripción de cualquier uso de la computadora para crear y manipular imágenes. Estas pueden ser completamente sintéticas, o modificaciónes de fotografías. Entre las aplicaciones más comunes estan los videojuegos, dibujos animados, efectos visuales, dise/ no y manufactura, películas, simulaciones, imágenes médicas, vizualización de datos entre otras. Debido a la demanda de mayor calidad en la imagen y en el menor tiempo de procesamiento posible, se han diseñado una gran variedad de técnicas y algorítmos. Entre ellos, los algoritmos de iluminación son una parte escencial en la representación de las imágenes, pues se encargan de colocar luz en una escena y procesarla, para lograr efectos como tipo de material, propiedades reflexivas y refractivas entre otras características propias de los objetos. Un algoritmo frecuentemente utilizado es Ray Tracing, debido a la interacción que tiene entre objetos, fuentes de luz y a que considera efectos de refracción y reflexión. Para que la ejecución de Ray Tracing genere una imagen con mayor calidad, requiere mayor tiempo de procesamiento y para algunas aplicaciones es indispensable obtener resultados precisos en un tiempo corto. Es por ello que se ha trabajado con varias técnicas para hacer más rápido este proceso, la paralelización es una de ellas.

Los sistemas paralelos se definen como la repartición de trabajo entre múltiples procesadores, con el objetivo de aumentar la velocidad con respecto al procesamiento secuencial [1]. El procesamiento en paralelo se ha convertido en la clave para construir los sistemas de cómputo del futuro [4]. Actualemente se han hecho varios estudios para determinar el nivel de necesidad de la paralelización de los sistemas,

así como estudios para reducir la complejidad en el proceso de paralelización. Los patrones para el diseño de software paralelo son una buena práctica para la realización de esta tarea.

1.2. Problema

Existen propuestas relacionadas con la paralelización de algoritmos de distintas índoles. En lo concerniente a los algoritmos de iluminación en computación gráfica, algunos de los trabajos que se han realizado han obtenido resultados adversos e inclusive, en algunos casos la ejecución del programa secuencial resulta más rápida que el programa en paralelo. Esto se debe en parte al poco tiempo invertido en el diseño del algoritmo. La falta de un esquema que proporcione bases para diseñar la paralelización de los algoritmos hace que la tarea de paralelizar se vuelva más compleja y que no se obtengan los resultados deseados.

La paralelización de Ray Tracing que permita obtener mejores resultados en las imágenes, en un menor tiempo de procesamiento, es uno de los desafíos más importantes tanto en computación gráfica como en los sistemas paralelos en la actualidad.

1.3. Hipótesis

La hipótesis planteada en esta tesis es:

> La paralelización del algoritmo de ilumniación Ray Tracing realizada con base en patrones de diseño, particularmente utilizando el patrón Manager Workers, resulta más eficiente que la paralelización diseñada con base en otras técnicas, debido a que se toma en cuenta la coordinación, comunicación y sincronización de los procesos en el diseño del algoritmo.
>
> La distribución de carga de trabajo entre los workers, basada en parámetros que determinen la complejidad de la porción de imagen que tendrá cada uno, disminuye el tiempo de ejecución del algoritmo. Esto se debe a que el algoritmo será tan rápido como su worker más lento, por tal motivo es relevante que la carga de trabajo sea similar, con el objetivo de esperar lo menos posible por el worker más lento.

1.4. Aproximación

Se propone la paralelización del algoritmo de iluminación Ray Tracing basada en el patrón Manager Workers. Medición de la eficiencia, en términos de la velocidad de la ejecución del algoritmo. Se verifica que Ray Tracing diseñado con base en patrones tiene un mejor desempeño debido a la metodología en el diseño de la coordinación, sincronización y comunicación de los procesos. Se realiza una propuesta para mejorar la distribución de la carga entre los procesadores participantes y así hacer más rápida la ejecución del Ray Tracing.

Lo anterior se hace a través de un programa el cual toma en cuenta todas las características del patrón Manager Workers para realizar la paralelización, se implementa la clase rendezvous entre los workers y el manager para realizar paso de mensajes entre ellos utilizando memoria compartida. Se crea un canal de comunicación entre el manager y cada worker coordinados a través de semáforos. Respecto a la distribución se toma en cuenta la complejidad de la imagen en términos de número de objetos por franja, esto permite una mejor distribución y por lo tanto menor tiempo de ejecución del algoritmo.

1.5. Contribuciones

1. Un programa del algoritmo de iluminación Ray Tracing concurrente basado en el patrón Manager Workers.
2. Análisis y metodología para paralelizar el algoritmo de Ray Tracing.
3. Propuesta de una distribución de la escena basada en la complejidad de la franja que le corresponde a cada worker.

1.6. Estructura de la Tesis

- En el **capítulo 2** se describen los conceptos básicos de computación gráfica: iluminación, algoritmos de iluminación, en particular el algoritmo de iluminación Ray Tracing; procesamiento en paralelo, patrones de diseño de software paralelo y procesamiento en paralelo aplicado a computación gráfica.
- En el **capítulo 3** se describen los trabajos que se han realizado al respecto, los métodos que se han utilizado para paralelizar Ray Tracing, los resultados que se han obtenido y las principales dificultades a las que se han enfrentado.

- En el **capítulo 4** se describe la propuesta de las características que se usan en el diseño de los algoritmos, como organización de la memoria: distribuída o compartida, patrón de diseño, Manager Workers, arquitectura, lenguaje de programación y especificaciones de sincronización, comunicación y coordinación de los procesos.

- En el **capítulo 5** se analiza la implementación del algoritmo de iluminación, se describen las escenas de prueba utilizadas y pruebas en diferentes escenas en el algoritmo Ray Tracing. Se presenta los datos obtenidos de los diferentes experimentos y se proporciona el análisis de dichos datos.

- En el **capítulo 6** se presentan las implicaciones de cada uno de los experimentos, así como las conclusiones generales. Se enlistan las contribuciones obtenidas y el trabajo a futuro.

Capítulo 2

Antecedentes

En este capítulo se revisan los conceptos relacionados con procesamiento paralelo y sus principales características; una introducción a los patrones de diseño de software paralelo, así como conceptos de computación grafica, haciendo énfasis en iluminación con el algoritmo Ray Tracing.

2.1. Procesamiento en Paralelo

Desde la llegada de la computación electrónica en los años 40's, la expectativa y demanda relacionadas con el rendimiento se han incrementado. La noción de rendimiento no tiene un criterío definido, pero éste puede ser medido de acuerdo a diferentes parámetros dependiendo de los requerimientos de usuarios y objetivos del uso de los sistemas de cómputo. Por ejemplo, en sistemas de bases de datos, el rendimiento que se busca, es el proveer espacios de direcciones muy largos y acceso rápido a medios de almacenamiento secundario. Para sistemas multi-usuario puede ser el número de usuarios concurrentes que se pueden soportar y el tiempo de respuesta bajo condiciones de carga pesadas. Una medida aceptada universalmente en el rendimiento computacional, particularmente en el campo de las ciencias e ingeniería, es el número de operaciones de punto flotante que se pueden realizar por segundo [4].

Procesamiento paralelo se define como la división de trabajo entre múltiples procesadores que operan simultaneamente para lograr un objetivo común. Se espera que el resultado de operar con multiprocesadores sea más rápido que con un solo procesador. La ventaja de los sistemas paralelos es que pueden manejar tareas de gran escala que con sistemas secuenciales no puede ser posible o no tomaría un gran

tiempo llevar a cabo su ejecución [1].

La programación en paralelo es una actividad compleja. El diseño en papel para aplicar multi-procesos a procesos simples suele ser sencillo. En la práctica, el procesamiento en paralelo tiende a ser difícil y costoso. Se requiere un gran esfuerzo por parte del programador. Se tienen que considerar nuevas técnicas para conocer adecuadamente la forma de programar en un ambiente paralelo. Normalmente las técnicas que se usan en los sistemas con un solo procesador para mejorar el redimiento, no aplican en la programación en paralelo. Adicionalmente, es necesario considerar que el ambiente de ejecución en paralelo es inherentemente inestable e empredecible, es decir, no determinístico [1].

El rendimiento es considerado como la principal razón para usar la programación de sistemas paralelos [7]. Como rendimiento en sistemas paralelos, se define a la capacidad de respuesta de un sistema, es decir el tiempo que se requiere para que los procesadores proporcionen una respuesta simultaneamente.

2.1.1. Clasificación de las computadoras paralelas

El tipo de paralelización que se puede realizar, es con base en datos y/o secuencia de instrucciones que puede ejecutar un procesador. Resulta sencillo creer que para paralelizar un sistema basta con dividir las instrucciones y aplicar técnicas para la distribución de los datos, sin embargo el diseño de sistemas paralelos implica mucho más. Esta falsa idea en ocaciones resulta en programas paralelos más lentos que su versión secuencial [1].

Hay diversos métodos que se han propuesto para organizar la programación paralela. Existen cuatro clasificaciones que destacan y en las que se pueden organizar los sistemas paralelos [1]. Gracias a esta clasificación es posible hacer algunas observaciones y determinar el costo-beneficio de algunos problemas que se paralelizan. Debido a que la base para paraleizar está en la réplica de datos e instrucciones, los cuatro tipos de clasificación de acuerdo con Flynn son: instrucción simple y datos simples (SISD), instrucción simple y múltiples datos (SIMD), instrucciones múltiples y datos simples (SDMI) y finalmente instrucción y datos múltiples (MIMD). SIMD y MIMD son las únicas consideradas como paralelismo real.

- SISD

  Se trata de la arquitectura convencional, Von Neumann. Un solo procesador que ejecuta secuencialmente las operaciones sobre un simple flujo de datos.

Aunque las operaciones se ejecuten secuencialmente las instrucciones se pueden traslapar o se puede utilizar pipeline.

- SIMD

  Este tipo es una arquitectura de arreglo de procesadores, en este caso las operaciones son coordinadas por un proceso controlador. Se ejecuta la misma instrucción sobre diferentes datos.

- MISD

  Varios procesadores ejecutan distintas instrucciones sobre los mismos datos. Pero esta clasificación no es considerada como un esquema de ejecución real.

- MIMD

  Consiste en distintos procesadores autónomos que operan asíncronamente, cada uno con distintos datos.

2.1.2. Organización de la memoria

Dentro de los recursos que se comparten cuando se tienen multiprocesadores se encuentran las memorias y los periféricos. En el caso de la organizacin de la memoria, existen dos esquemas: memoria compartida y memoria distribuida. De acuerdo a la forma de organización que se utilice se define el tipo de comunicación que se utiliza, variables compartidas o paso de mensajes respectivamente [5].

Memoria Compartida

Se establece la memoria compartida, cuando todos los procesadores tienen accesos a todas las localidades de la memoria y su comunicación es por medio de una red.

Dicha red es controlada por el hardware, el programador no tiene que preocuparse por está, solo observa una memoria general y cada localidad de memoria es única. Se maneja automáticamente la selección de la localidad de memoria en la cual se va a escribir, y esto se realiza a través de variables compartidas [5].

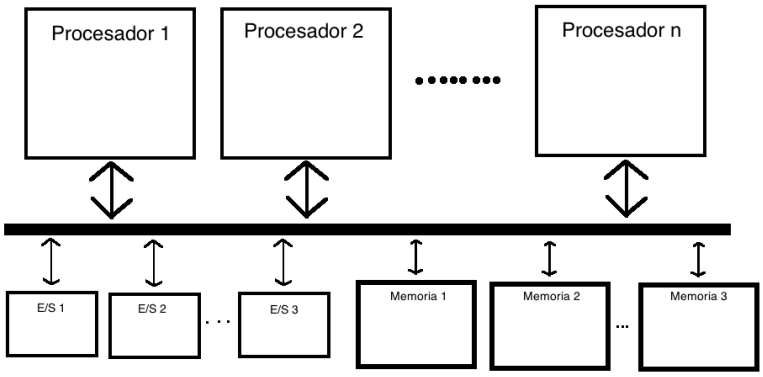

Figura 2.1: Memoria Compartida [5].

Memoria Distribuida

Cada procesador utiliza su memoria privada, comunicándose con otros procesadores mediante una red [5].

La red se forma a partir de un número de Procesadores que se conectan entre sí y se basa en una topología que puede modificarse durante la ejecución del programa [5].

La comunicación que se realiza entre los procesadores requiere el envío y recepción de datos. Los procesadores pueden modificar libremente su memoria local, pero cuando requieren modificar o leer la memoria de otro procesador realizan un intercambio de datos y explícitamente envían mensajes al procesador con el cual se quieren comunicar [5].

Este envío de mensajes es conocido como paso de mensajes. La comunicación es punto a punto (comunicación solo entre dos procesadores), unidireccional(un proceso origen y un proceso receptor) y no bufferizada (no hay almacenamiento momentáneo) [5].

2.1.3. Patrones de diseño de software paralelo

Quienes se han enfrentado a la tarea de realizar programación en paralelo coinciden en que hay tres aspectos a considerar: hardware, el lenguaje de programación y el problema a resolver. Sin embargo, en muy pocas ocasiones se hace enfasis en el tercer punto, y esto es de suma importancia debido a que es

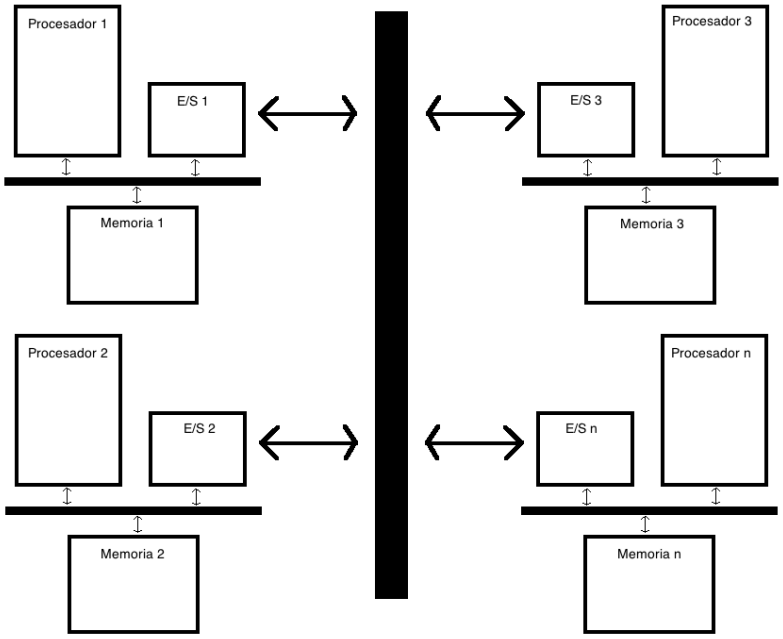

Figura 2.2: Memoria Distribuida [5].

precisamente en éste donde puede radicar el desempeño del programa [1].

El problema a resolver se describe en términos del algoritmo, la abstracción de las operaciones que se aplicaran a los datos y la paralelización del algoritmos, de los datos o de ambos. En sí, el problema mayor es mejorar la eficiencia de los algoritmos de iluminación, reduciendo el tiempo de ejecución. [1].

Paralelizar un programa es enfretarse a problemas tales como, qué técnica ocupar, que no haga más complejo el proceso de paralelizar, y al mismo tiempo se vea una mejora sustancial. La sincronización entre los procesadores afecta directamente en el desempeño del programa, y finalmente la complejidad [1]. Lo que se busca al emplear patrones de diseño es reducir estos problemas. Aún no hay una receta que seguir ya que la paralelización depende del tamaño del programa o los datos, pero sí se puede realizar una mejor organización y muchas veces evitar incidir en problemas tales como que la ejecución del programa secuencial sea más rápida que en paralelo [1].

Existen dos retos importantes cuando se paraleliza un algoritmo. En primer lugar el problema a resolver, abstraer el problema y llevarlo a un algoritmo que pueda ser ejecutado por una computadora. En segundo lugar el proble-

ma en paralelo, coordinar la forma en la cual interactuaran los procesadores para resolver el problema. Para el segundo caso se han desarrollado los Patrones de Diseño de Software Paralelo, los cuales son técnicas que segmentan en componentes independientes la forma en la que actuaran los procesadores al paralelizar el sistema. A continuación se describen las técnicas más comunes.

Pipes and Filters

En cada etapa se procesan los datos, y después de ser procesados, pasan por un pipe (un canal de comunicación) a otro filter (etapa de procesamiento) donde se ejecutan otras instrucciones. Cada componente paralelo realiza simultáneamente un paso diferente del cómputo. Cada paso representa un cambio del valor de entrada o un efecto sobre el tiempo. Se aplica en Graphics rendering [1].

Fuerzas:

- Preservar el orden de las operaciones.
- Preservar el orden de datos entre las operaciones.
- Considerar la independencia entre los pasos operacionales cuyos procesamientos se puede realizar en diferentes piezas de datos.
- Distribuir procesamiento de manera igual entre los pasos operacionales.
- Mejorar el desempeño mediante la reducción del tiempo de ejecución. [1]

Parallel Hierarchies

Se representa a través de un árbol el cual soporta datos recursivos, hay balance de datos, arreglo de llamadas, llamada múltiple, dos o más componentes de una capa pueden existir simultáneamente y realizan la misma operación. Son creados estáticamente cuando esperan llamadas de capas más altas. Son creados dinámicamente cuando una llamada dispara su creación. Ejemplos en Single Source, Shortest path alghoritm

Communicating Sequential Elements

Los elementos en este patrón, utilizan canales bidireccionales. Si se hacen síncronas, las comunicaciones mejoran. Con la Sincronización de barrera, los

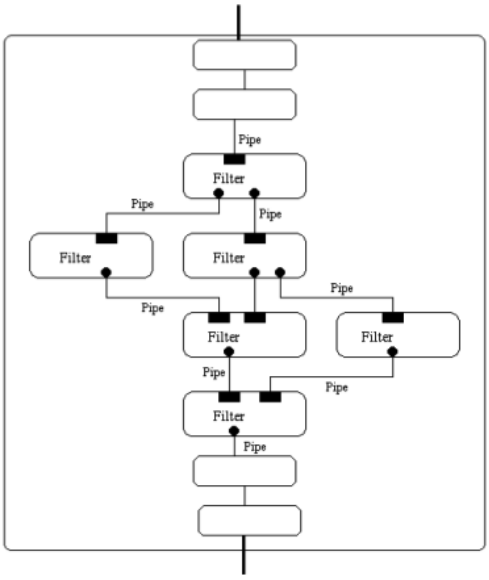

Figura 2.3: Pipes and Filters [1].

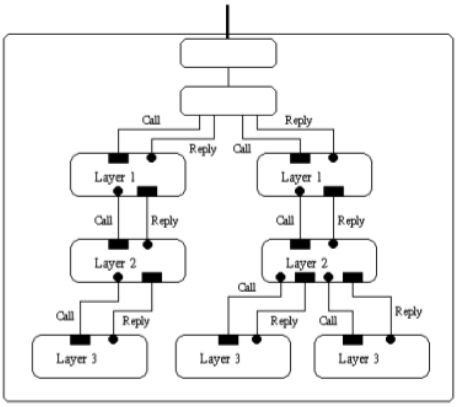

Figura 2.4: Parallel Hierarchies [1].

datos trabajan autónomamente hasta que necesitan sincronizarse. Cada componente realiza las mismas operaciones en diferentes piezas de datos. Las operaciones dependen de resultados parciales en componentes vecinos. Ejemplo la Ecuacion del calor.

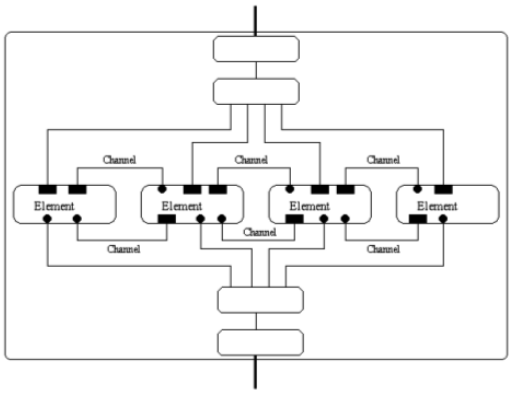

Figura 2.5: Communicating Sequential Elements [1].

Manager-Workers

El patrón Manager Workers es una variación del patrón maestro-esclavo pero para sistemas paralelos. La paralelización que se lleva a cabo, es paralelización de actividad. En este tipo de paralelización se dividen los datos y el algoritmo, se realizan las mismas operaciones sobre los datos ordenados. La principal diferencia con respecto al patrón maestro esclavo es que los componentes de este patrón son proactivos, los workers una vez que terminan de realizar su trabajo, lo regresan y hacen peticiones de más trabajo al manager; los esclavos esperan que el maestro les designe más trabajo. Cada componente del procesador realiza la misma operación simultánea e independientemente del procesamiento de otros componentes. Es importante preservar el orden de los datos, el patrón ayuda a preservar este orden por medio del manager el cual se encarga de organizar a los workers.

Las repartición de los datos a los workers puede ser de diferentes tamaños. Los workers solicitan el trabajo y el manager es quien ordena los datos recibidos. Las tareas de los workers pueden ser totalmente diferentes. Sincronización de manera sencilla. Los componentes son proactivos en lugar de reactivos. Un problema común que se resuelve con este patrón es el de traslape de polígonos.

Shared Resource

Los cómputos pueden ser realizados en datos ordenados sin importar el orden de los procesos. Los componentes realizan simultaneamente diferentes cómpu-

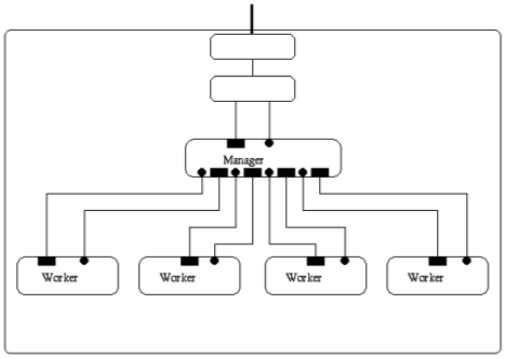

Figura 2.6: Manager-Workers [1].

tos con diferentes partes de los datos. Realiza las operaciones independientemente hasta que requiere algún dato del recurso compartido.

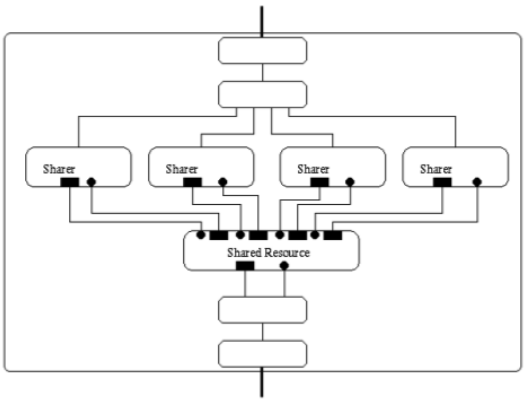

Figura 2.7: Shared Resources [1].

2.2. Computación Gráfica

Computación gráfica significa crear, almacenar y manipular modelos e imagenes. Se modelan estructuras en diversos y variados campos, incluyendo física, matemáticas, artistica, biología, incluso conceptos abstractos [10].

"Tal vez el mejor camino para definir computación gráfica es encontrar que no lo es. No es una máquina, No es una computadora, ni un grupo de programas. No es el cómo de un diseñador gráfico, programador, escritor o un especialista en animación. Computación gráfica es un manejo conciente y tec-

nología documentada dirigida hacia la comunicación de la información precisa y descriptiva"[3].

2.2.1. Iluminación

Iluminación en computación gráfica se refiere a la colocación de luz en una escena de tal forma que la escena luzca más real. La imágenes sintetizadas y los paquetes de animación contienen diferentes tipos de luz que pueden ser colocados en diferentes posiciónes y modificados cambiando sus parámetros. Frecuentemente, los desarrolladores de 3D que crean imágenes o animaciones ignoran o ponen poco interés en la iluminación. Esto es desafortunado, pues la iluminación es una parte muy importante de la sintezación de las imágenes. El uso apropiado de luz en una escena es lo que hace la diferencia entre una escena de apariencia real y una que no lo es. Esto no es un tema nuevo: hay un gran trabajo que ha sido realizado en las áreas de fotografía y video [6].

2.2.2. Ray Tracing

Ray Tracing es un algoritmo gráfico muy poderosa, es usado para desplegar efectos visuales. Ray Tracing es uno de los algoritmos de iluminación más precisos para representar los efectos de la luz en una escena. Sin embargo, su ejecución implica una demanda de procesamiento alta. Es por eso que generalmente no se usa en videojuegos u otras aplicaciones en tiempo real. Ray Tracing permite calcular correctamente los colores de los pixeles, disparando rayos a los objetos y manteniendo las trayectorias de los rayos, reflexiones, sombras y otras propiedades. Ray Tracing puede operar con un solo rayo por pixel. Sin embargo, se puede implementar super muestreo el cual requiere mas de un rayo por pixel[8].

Debido a la gran cantidad de procesamiento requerido para generar una imagen de alta resolución usando Ray Tracing, aplicar paralelización se vuelve muy importante para lograr un rendering más rápido.

Ray Tracing calcula la trayectoria de rayos de luz desde el punto de vista hasta el plano de la escena. Se verifican todos los rayos contra cada uno los objetos en la escena para determinar si los intersecta. Si el rayo no intersecta ningún objeto, el pixel es sombreado del color de fondo. Ray Tracing se basa

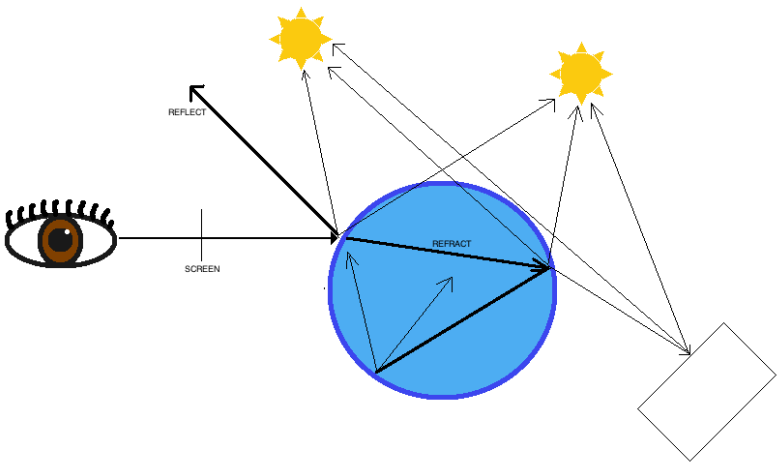

Figura 2.8: Algoritmo Ray Tracing [9].

en sombreado. Se encarga de múltiples reflexiones, refracciones y mapeo de textura [6].

Algorithm 1 Pseucode for a simple ray tracer

```
Select center of projection and window on viewplane
for each scan line in image do
  for each pixel on scan line do
    determine ray from center of projection through pixel;
    for each object in scene do
      if object is intersected and is closest considered thus far then
        record intersection and object name;
      end if
    end for
    set pixel's color to that at closest object intersection;
  end for
end for
```

Geométricamente los rayos que se toman en cuenta son los mostrados en la Figura 2.9:

- $\hat{D}$ Rayo principal.
- I Intensidad del rayo principal.
- R Rayo reflejado
- S Intensidad del rayo reflejado

- Q Rayo refractado
- T Intensidad del rayo refractado
- Lj La j-esima fuente de luz

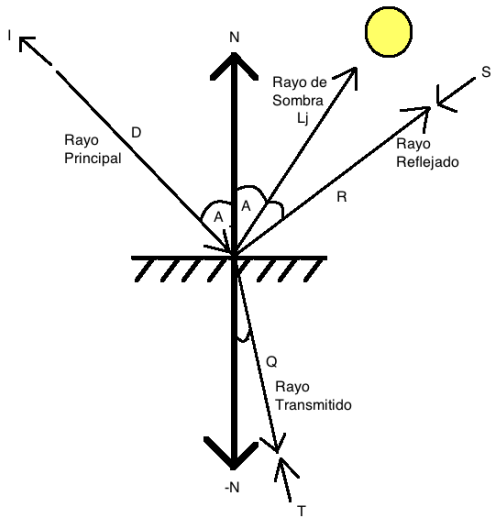

Figura 2.9: Geometría de Ray Tracing [4].

R está determinado por la dirección del reflejo de $\hat{D}$ con respecto al vector normal de la superficie $\hat{D}$. Q es dado por la ley de refracción de Snell.

La aplicación recursiva del algoritmo Ray Tracing es representada en árboles por cada pixel, donde los nodos del árbol son los rayos y los vertices son las intersecciones.

Constructive Solid Geometry (CSG) es un método que convierte objetos complejos a figuras geométricas simples como esferas, cilindros, etc. con las operaciones booleanas: Union, Intersección y diferencia. Es el método que se utliliza para hacer las intersecciones en Ray Tracing.

Una de las principales rutinas de Ray Tracing es encontrar las intersecciones del rayo con los objetos. A grandes rasgos, una función para encontrar las intersecciones regresa un valor el cual indica si hay intersecciones. En caso de que las haya, también devuelve qué tan lejos está el objeto y el vector normal de la superficie en el punto donde se realiza la intersección. El vector normal es importante para poder calcular el rayo refractado y el rayo reflejado.

Un rayo puede ser considerado como una línea semi-infinita, donde:

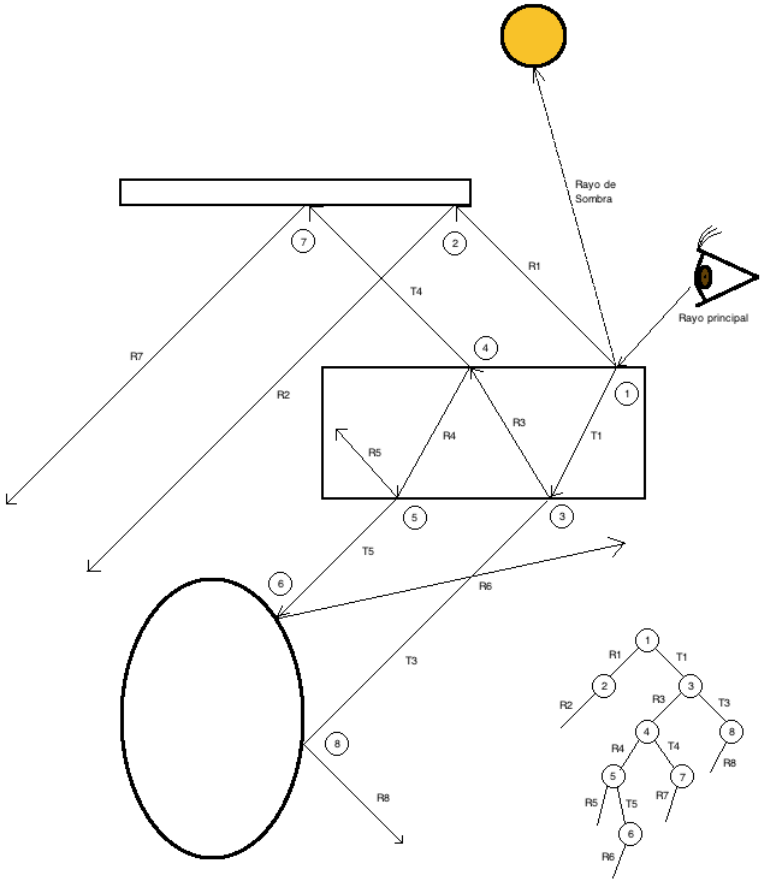

Figura 2.10: rbol de rayos en un pixel.[4].

- *P Punto de inicio del rayo principal.*
- $\hat{D}$ Dirección del rayo principal

Entonces cualquier punto X que pertenezca al rayo puede ser expresado en la siguiente equación pramétrica que representa al rayo:

$$X = P + \lambda \widehat{D}, \lambda >= 0 \tag{2.1}$$

La dirección del vector ha sido normalizada,

- λ *Denota la distancia del rayo*

Todos los puntos en el rayo pueden ser representados por algunos valores de λ.

Se necesita hacer una mejora en el desempeño del cálculo de las intersecciones, con el objetivo de optimizar el cálculo de estás en toda la escena. Para eso se toma el ejemplo de la esferea. Una esfera puede ser descrita geometricamente:

- *V Punto central de la esfera*
- *r Radio*

Entonces un punto X sobre la superficie de la esfera está dado por:

$$|X - V| = r \tag{2.2}$$

Todo punto X satisface las dos ecuaciones así que sustituyendo la ecuación 2.1 y 2.2 obtenemos:

$$\left|P + \lambda\widehat{D} - V\right| = r \tag{2.3}$$

Utilizando la propiedad de que $|X| = \sqrt{X * X}$, la ecuación 2.3 puede ser expresada como:

$$((P - V) + \lambda\widehat{D})^2 = r^2 \tag{2.4}$$

Expandiendo la ecuación obtenemos:

$$\lambda^2 + 2(P - V)\widehat{D}\lambda + (P - V)^2 - r^2 = 0 \tag{2.5}$$

Tenemos una ecuación cuadratica respecto a λ con coeficientes A, B y C dados por:

$$A = 1 \tag{2.6}$$

$$B = 2(P - V)\widehat{D} \tag{2.7}$$

$$C = (P - V)^2 - r^2 \tag{2.8}$$

La solución para la ecuación cuadratica λ es:

$$B^2 - 4AC > 0 \tag{2.9}$$

Calculando los puntos de intersección de acuerdo al vector normal:

Paso 1: Cálculo de A,B y C.

$$A = 1 \tag{2.10}$$

$$B = 2[(P_x - V_x)D_x + (P_y - V_y)D_y + (P_z - V_z)D_z] \tag{2.11}$$

$$C = (P_x - V_x)^2 + (P_y - V_y)^2 + (P_z - V_z)^2 - r^2 \tag{2.12}$$

Paso 2: Cálculo y prueba de $B^2 - 4AC$.

$$(A = 1) \tag{2.13}$$

$$d = B^2 - 4C \tag{2.14}$$

$$Test : d >= 0 \tag{2.15}$$

Paso 3: Cálculo y prueba de λ_1

$$\lambda_1 = -0{,}5(B + \sqrt{d}) \tag{2.16}$$

$$Test : \lambda_1 > 0 \tag{2.17}$$

Paso 4: Cálculo y prueba de λ_2

$$\lambda_2 = -0{,}5(B - \sqrt{d}) \tag{2.18}$$

$$Test : \lambda_2 > 0; Set\lambda = min(\lambda_1, \lambda_2) \tag{2.19}$$

Paso 5: Determinar el punto de intersección

$$X_x = P_x + \lambda D_x \tag{2.20}$$

$$X_y = P_y + \lambda D_y \tag{2.21}$$

$$X_z = P_z + \lambda D_z \tag{2.22}$$

Paso 6: Determinar el vector normal

$$N_x = \frac{1}{r}(X_x - V_x) \tag{2.23}$$

$$N_y = \frac{1}{r}(X_y - V_y) \tag{2.24}$$

$$N_z = \frac{1}{r}(X_z - V_z) \tag{2.25}$$

Ray Tracing está representado por el algoritmo .Algorithm 2".

Existen algunas formas de incrementar la velocidad computacional [6]:

1. Usar máquinas más rápidas.
2. Usar hardware especializado, especialmente procesadores paralelos.
3. Usar algoritmos más eficientes.
4. Reducir el número de rayos - objetos.

Existen diversas formas de incrementar el rendimiento del algoritmo de Ray Tracing. Sin embargo, este trabajo se enfoca en el uso de Procesamiento en Paralelo.

2.3. Resumen

En este capítulo se revisa brevemente los conceptos que se utilizan a los largo del trabajo. Paralelización es un paradigma que busca mejorar el rendimiento con respecto de los programas secuenciales. Existen muchos algoritmos secuenciales que debido a su alto procesamiento tienen la necesidad de mejorar su rendimiento, tal es el caso de muchos algoritmos que se utilizan en computación gráfica, en donde con el objetivo de representar una escena real es necesario adquirir muchos recursos de los procesadores. Dentro de los diversos algorítmos de computación gráfica, algunos de los más importantes son los de iluminación, ya que es una forma de detallar la escena y lograr hacerla lucir más real. Estos algorimos no están excentos de la necesidad de alto desempeño computacional. Tal es el caso que se pretende analizar de los más encontrados en la literatura para determinar cuál es más adecuado para paralelizarse y de que forma. El algorímo elegido es Ray Tracin. Ray Tracing está enfocado a calcular rayos desde la vista del usuario a objetos que lo puedan intersectar para determinar como se ilumina.

Algorithm 2 Código Ray Tracing

```
1: for int j = 0; j < imageHeight; ++j do
2:   for int i = 0; i < imageWidth; ++i do
3:     Ray primRay;
4:     computePrimRay(i,j, &primRay);
5:     Point pHit;
6:     Normal nHit;
7:     float minDist = INFINITY;
8:     Object object = NULL;
9:     for int k = 0; k < object.size(); ++k do
10:      if Intersect(objects[k],primRay,&pHit,&nHit) then
11:        float distance = Distance (eyePosition, pHit);
12:        if distance < minDistance then
13:          object = objects[k];
14:          minDistance = distance;
15:        end if
16:      end if
17:    end for
18:    if Object != NULL then
19:      Ray shadowRay;
20:      shadowRay.direction = lightPosition - pHit;
21:      bool isShadow = false;
22:      for int k = 0; k < objects.size(); ++k do
23:        if Intersect(objects[k], shadowRay) then
24:          isInShadow = true;
25:          break;
26:        end if
27:      end for
28:    end if
29:    if !isInShadow then
30:      pixels[i][j] = object-> color * light.brightness;
31:    else
32:      pixels[i][j] = 0;
33:    end if
34:  end for
35: end for
```

Capítulo 3

Trabajo relacionado

En este capítulo se presentan algunos de los trabajos más representativos relacionados con la paralelización del algoritmo Ray Tracing, cuáles son sus aportaciones, qué dificultades encuentran, qué procedimiento utilizan para paralelizar, ventajas y desventajas de dichos procedimientos, y el trabajo futuro a desarrollar.

3.1. Parallel Ray Tracing

"Parallel Ray Tracing"(Ray Tracing en paralelo), de Steve Glazer, Sara Jackson y Samuel Milton [8] propone una arquitectura paralela para Ray Tracing basada en estructuras de aceleración de malla regular que se encarga de subdividir la escena. Una estructura de aceleración como una malla regular, un árbol o un octeto (regular grid, KD-tree and octree) dividen la escena en conjuntos para ser procesados, decrementando el número de pruebas de intersección necesarias, y así, mejorar el tiempo de renderizado. Normalmente la estructura de árbol KD es la más utilizada cuando se hace una implementación de Ray Tracing. En este artículo se realiza una propuesta de Ray Tracing usando una red uniforme, implementándolo en un hardware básico con el objetivo de mejorar el tiempo de renderizado.

El trabajo se enfoca en mostrar el nivel de mejora en un programa de Ray Tracing secuencial con respecto a uno paralelo, implementado en ocho núcleos. Se muetra que es importante para los programas gráficos ejecutarlos en paralelo[8].

Para este proyecto se trazan tres clases de rayos. Estos rayos se enfocan en renderizar mundos 3D que contienen multiples objetos. Estos objetos consisten en esferas y triángulos. Pero también se contemplan formas más complejas que pueden surgir de la combinación de varios triángulos pequeños. Este proyecto contiene tres clases de rayos, para los objetos en 3D: luz, cámara y el mundo que los contiene. La cámara renderiza el mundo que se encuentra dentro del campo de vista a archivos de imagenes [8].

En cada ejucución, independientemente de qué algorítmo de iluminación se use, se guardan las estadísticas de la salida estándar con respecto al desempeño del programa. Las entradas son el tipo de mundo y sus parámetros así como la semilla que se genera aleatoreamente que se usa. Después se obtiene el tiempo promedio en que cada frame se renderiza, el tiempo promedio de entrada-salida y el tiempo promedio total que se tarda en producir cada frame. Finalemente la salida es el tiempo total que se tarda en ejecutar la secuencia dividido entre cada frame. sto da las métricas que se utilizan para medir el desempeño de cada algorítmo (Ray Tracing). Todas las estadísticas se producen de la misma manera con el objetivo de no afectar el desepeño del programa [8].

En la siguiente figura se puede ver el diseño básico de los tres rayos. El rayo toma los mundos del generador y los envía a la cámara para renderizarlos. La cámara los renderiza hacía la dirección que esta observando y así es como se producen los frames [8].

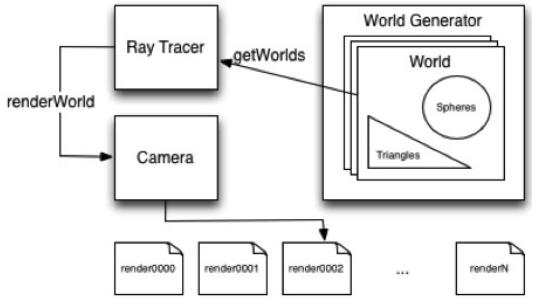

Figura 3.1: Diseño de Ray Tracing en paralelo

Ray Tracing Secuencial

La ejecución secuencial de los rayos trabaja en cada frame uno a la vez. Cuando se trabaja con solo un frame, se obtiene cada pixel empezando por el borde

superior-izquierdo y computando cada fila hasta que se termina todo el frame. Y así sucesivamente con los siguientes frames lo que resulta en la animación [8].

Versión 1

El proceso fundamental de renderizar una imagen es mantenerse sin cambio respecto a la versión secuencial esto es para no generar conflictos al momento de escribir en la imagen. Para ello se dividen los fames de la animación en los diferentes hilos. Por ejemplo si una imagen de 8 frames se renderiza en un sistema que tiene 8 núcleos, entonces una imagen se renderiza en cada uno de los núcleos [8].

Cada frame es independiente de la imagen. Si se asume que se sabe que se hace con cada frame, no existe dependencia secuencial entre los frames lo cuál permite distribuir os frmaes entre los núcleos y renderizarlos en cualquier orden [8].

Se usa una calendarización para dividir el trabajo. La cantidad de trabajo en cada imagen no es la misma. Por un instante en la animación es posible para un marble ir fuera del campo de vista de la cámara. Si esto llega a pasar hay menos rayos de reflexión para computar. Esto también pasa cuando los marbles están más lejos. Cuando se ejecuta una animación usando sólo una calendarización, es posible que algunos hilos terminen antes que otros, incluso cuando la distribución sea homogenea. Entonces se puede asignar más trabajo a aquellos que terminen antes. Es por eso que se crea una segunda versión en paralelo [8].

Versión 2

En esta versión, todos los hilos pueden trabajar juntos sobre un mismo frame al mismo tiempo. Esto tiene muchos beneficios. Uno de los principales es que ya no es necesario hacer esperar a los hilos más rápidos. Esto hace más funcional el Ray Tracing tanto para una simple escena como para las animaciones [8].

Con la computadora de 8 núcleos no se pudo optimizar el paralelo versión 2 respecto el secuencial. La velocidad no disminuye de 7.3 segundos y la eficiencia no se reduce de 90 %. Durante la fase de desarrollo alcanzan velocidades

negativas con el algorítmo en paralelo, debido a la alta cantidad de interferecia con el caché [8].

El mayor problema es que se tiene mayor velocidad con el algoritmo paralelizado que con el secuencial. Es por eso que se tuvo que hacer una segunda versión en la que se paralelizaban los pixeles en los frames. Se encuentra que toma más tiempo renderizar la imagen completa que la mitad de ésta y así sucesivamente, entre más se particiona más tiempo toma. Después de optimizar el código se llega a la conclusión de que el problema es un exceso de comunicación. El problema radica en interferencias con la memoria caché. Existen problemas de métodos recursivos con muchos objetos y llamadas cada vez que se crea un nuevo objeto. Se reescribe el código y se arreglan estos métodos para que funcionen mejor con la memoria cache.

Un sistema híbrido puede funcionar perfectamente para Ray Tracing. Se implementa un ordenamiento paralelo de los objetos en la escena con el objetivo de optimizar la eliminación de superficie oculta. Se encuentran problemas con el algoritmo para las reflexiones y refracciones pues tienen que ser recursivas y necesitan paralelizar el ordenamiento con cada iteración del método recursivo. Con un sistema híbrido se podría paralelizar la renderizacion del pixel sobre diferentes nodos y paralelizar el ordenamiento de estos objetos a traves de cada núcleo [8].

3.2. An efficient Parallel Ray Tracing Scheme for Distributed Memory Parallel Computers

"An efficient Parallel Ray Tracing Scheme for Distributed Memory Parallel Computers"(un esquema eficiente de Ray Tracing paralelo), de Wilfrid Lefer, se hace el estudio de la implementación del algoritmo de Ray Tracing en una computadora paralela de memoria distribuida. Se propone una solución basada en la asociación de paralelismo de datos. Se implementa un mecanismo de redistribución de datos de carga dinámico.

Para realizar la paralelizacion de Ray Tracing se necesita distribuir el cómputo y los datos. Es necesario que cada pixel sea procesado, los rayos se pueden propagar en todo el espacio. Es por eso que cada vez que se procesa un pixel es necesario tener la base de datos completa.

Existen tres maneras de realizar esta acción:

- Duplicación de la base de datos completa en cada nodo
- Distribución geométrica de la base de datos
- Distribución adaptativa de la base de datos

Sin embargo, al duplicar la base completa en todas las memorias locales se limita el tamaño de la escena que puede ser procesada. La distribución geométrica de la base de datos incrementa el problema del desbalanceo de carga. Modificando la subdivición del espacio se incrementan los problemas de la paralelización. Distribuir la base de datos provoca que la comunicación se incremente. Finalmente el submuestreo no es una solución muy adecuada, debido a que cada pixel depende de su posición en la pantalla y ésto varía durante la fase de síntesis. Los algoritmos con flujo de datos implican el uso de memoria virtual y mecanismos de caché. Su efectividad depende del tamaño de la memoria y de la estrategia de distribución de los pixeles. No resulta fácil hacer una comparación entre métodos porque generalmente la escena que se utiliza para uno y para otro varia de un artículo a otro.

Se propone un nuevo algoritmo de balance de carga con el objetivo de procesar imágenes más grandes. Para eso se distribuye la base de datos en la memoria local. El algoritmo se basa en una aproximación a la distribución geométrica, y además, se propone un mecanismo de redistribución de carga, que permite un buen balance de carga. Una característica importante de este algoritmo es que el balance de carga se hace de forma dinámica, se realiza durante la fase de síntesis.

El algoritmo se implementa en una computadora basada en 8 procesadores de 25MHz con 2Mbytes cada uno. El algoritmo se prueba con 4 bases de datos. La profundidad máxima de el árbol de Ray Tracing se fija en 5, y las imágenes tienen una resolución de 512x512. Los resultados arrojaron un buen desempeño para todas las escenas que se probaron.

Existe un nuevo esquema de Ray Tracing paralelo, el cual combina paralelización de datos con paralelización de tareas. Se asegura balance de carga con un mecanismo de redistribución de carga dinámico. Estos buenos resultados se puede escalar a más de ocho procesadores. Dentro de las mejoras que se puede implementar.

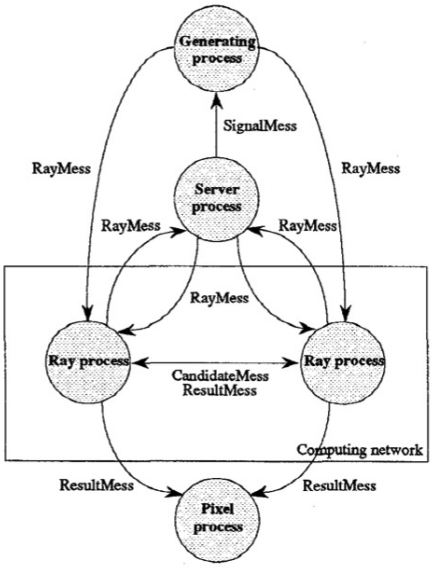

Figura 3.2: Organización de la red

3.3. Resumen

En este capítulo se analiza los trabajos que se han realizado con Ray Tracing en paralelo. Se presentan cuáles son las características que han funcionado y cuáles no para mejorar el rendimiento de los algoritmos cuando se paralelizan. Esto nos ayuda a la realización de la propuesta.

Capítulo 4

Análisis de la Paralelización de Ray Tracing

En este capítulo se presenta la metodología de la paralelización del algoritmo Ray Tracing. Se explica la necesidad de reducir el tiempo de procesamiento, y se da una descripción de las características y propiedades principales para realizar la paralelización de Ray Tracing con el patrón Manager Workers. Se propone una mejor distribución para los procesadores, para disminuir el tiempo de respuesta del algoritmo.

4.1. Procesamiento en Paralelo aplicado a Ray Tracing

Las aplicaciones de computación gráfica consumen una mayor cantidad de usu de procesador. La demanda de gráficos en tiempo real y las técnicas para la síntesis de imágenes para que parezcan reales, han permitido disponer de las facilidades que ofrece el procesamiento en paralelo para los sistemas de computación gráfica. Combinando los recursos de un número separado de procesadores, el rendimiento del sistema puede incrementarse para cubrir con la demanda del procesamiento de algunos de los problemas que requieren alto procesamiento en computación gráfica [4].

Ray Tracing implica procesamiento costoso en términos de tiempo y recursos entre mayor calidad tenga la imagen. Por tal motivo, existe la necesidad de

mejorar el rendimiento. Existen diversas técnicas para mejorar la velocidad de ejecución, como algoritmos para prepocesar la información, en este trabajo se opta por realizar la paralelización del algoritmo como método para mejorar el rendimiento.

4.1.1. Ray Tracing en Paralelo

Ray Tracing consume mucho tiempo de ejecución debido a los cálculos de intersección ya que cada rayo debe ser revisado contra todos los objetos. Para un Ray Tracing convencional (sin técnicas de aceleración), el tiempo es proporcional al número de rayos multiplicado por el número de objetos en la escena. Cada intersección requiere, en el mejor de los casos entre 5 y 7 operaciones de punto flotante (fp); en el peor de 15 a 20. Aproximadamente en una escena con resolución de 512x512, asumiendo 10 operaciones por objeto probado, hay 250 000 x 100 x 10 = 250 000 000 operaciones. Esto es tan solo para los primeros rayos (desde el punto de vista hacia el plano de la imagen), sin anti-aliasing y sin tomar en cuenta los rayos de reflexión y de refracción [6].

4.2. Paralelización del algoritmo Ray Tracing

4.2.1. Precondiciones

La principal razón por la cuál se realiza el proceso de paralelización de algoritmos, es debido a que de otra forma toma algunas horas, días incluso meses, todo depende de qué tan complejo sea el algoritmo y los datos que se manipulan. En el caso de Ray Tracing, si se requiere una buena calidad de la imagen, se puede realizar una ejecución considerando más rayos. A mayor número de rayos, mayor tiempo de procesamiento paro una mejor resolución de la imagen y de apariencia más real debido a la cantidad de rayos que hay que procesar.

En algunas ocaciones existen algorimos que por la naturaleza del problema son suceptibles a cambios constantes, esta razón hace que el trabajo de paralelizarlo tenga que hacerse cada vez que el algoritmo sufre cambios. Puesto que el algoritmo de Ray Tracing es muy común y los cambios que se hacen son pocos puesto que está basado en un algoritmo matemático que lo respalda; los

cambios que se realizan no son suficientemente significativos como para afectar el proceso de paralelización.

Independientemente del tiempo que se tarde en ejecutar el programa, es importante recalcar que siempre se puede mejorar la calidad y realidad de la imagen. El hacer esto requiere un mayor procesamiento computacional. En el caso de los algoritmos de iluminación , realizar una mejora es muy significativo puesto que es un objetivo fundamental de la computación gráfica.

El objetivo de Ray Tracing es realizar lo más real posible la iluminación de una escena que principalmente contenga objetos reflejantes. Entre mayor calidad se requiera para la imagen se requiere mayor procesamiento.

Para simplificar el problema, en este trabajo, se utilizan escenas de un solo tamaño 600x800 pixeles. Normalmente la solución secuencial hace un recorrido pixel por pixel sobre toda la pantalla y por cada pixel se manda un rayo principal y recursivamente se repite hasta una profundidad máxima. Para una profundidad muy grande o para una gran cantidad de datos Ray Tracing resulta un algoritmo que requiere mucho procesamiento. El análisis de cada pixel se hace de forma independiente y es por ese motivo que se puede paralelizar por datos y de la misma forma por operaciones. El resultado de un pixel no afecta a los otros pixeles, es decir el color final de ese pixel no depende del color final de sus vecinos o de los pixeles de los objetos con los que intersecten los rayos, los valores que utiliza para calcular el color de un pixel siempre serán los valores iniciales. Las operaciones de igual forma no dependen unas de las otras.

4.2.2. Patrones de diseño de software paralelo

Los patrones de arquitectura para la programación paralela representan una aproximación de software para el diseño de la coordinación de los procesos en programas paralelos. Estos patrones son un apoyo para hacer el salto entre los algoritmos y los programas, en este proceso se especifican propiedades y responsabiidades de los sub-sistemas y la forma particular en la que se ensamblan para trabajar coordinadamente.

Los patrones de arquitectura permiten a los diseñadores y desarrolladores entender el software complejo en bloques conceptuales y sus relaciones con el

objetivo de reducir la carga. Se proponen diferentes formas en las cuales los componentes de los sistemas paralelos pueden ser estructurados u organizados.

El paso más importante en el diseño de programas paralelos es la coordinación. Los patrones proveen descripciones acerca de cómo coordinar un programa paralelo.

Al utilizar los patrones se pueden obtener las siguientes ventajas:

- Se provee una descripción entre el enunciado del problema y la solución. Un enlace entre el algorimto o datos y la organización, coordinación y comunicación.
- La partición de un programa es la clave para determinar si un programa paralelo puede funcionar. Los patrones de arquitectura para programación paralela han sido desarrolldos y clasificados basados en la forma de partición aplicada al algorimto y/o los datos presentados en el problema.
- Dependiendo del algoritmo los datos y la forma de particionarlos se podrá elegir un patrón el cuál manejara la selección de una potencial coordinación paralela observando y estudiando las características el orden y la dependencia entre las instrucciones y los datos.

La complejidad del algoritmo Ray Tracing en una computadora secuencial, es n^2: supongamos un ejemplo en el cual se tiene una imagen de 500x500 pixeles,con una profundidad de 3. Por cada pixel se crean 7 nodos de un árbol, por cada nodo se realizan 4 operaciones: calcula posibles intersecciones, calcular el color que aporta la fuente de luz, calcular el rayo que refleja y calcular el rayo que refracta. Se realiza un total de 28 operaciones por cada pixel existente en la pantalla, lo cuál daría un total 7 000 000 operaciones. Evidentemente los cuatro cálculos a grandes rasgos contienen dentro una serie de operaciones geométricas con las que trabaja Ray Tracing. Ahora, si aumentamos en solo uno el nivel de profundidad del árbol las operaciones que se realizan se incrementan a más del doble, se realizan 15 000 000 de operaciones. Con este pequeño cambio de parámentros en el aumento de la profundidad del árbol afecta radicalmente en el número de operaciones lo que a la vez tienen un impacto directo en el tiempo que tarda en devolver una solución.

- Problema.

Ray Tracing puededesarrollarse de una manera más eficiente de la siguiente manera:

- Se puede aprovechar la independencia de las operaciones para poder realizarlas simultaneamente.
- Permitir que se trabaje simultaneamente con un grupo de pixeles asignados a los nodos.
- Asegurar que la distribución del trabajo a los nodos sea la adecuada para hacer que el algoritmo responda en menor tiempo.

- Descripción del problema.

  El programa en paralelo, para hacer su trabajo más eficiente toma un arreglo de pixeles y debe considerar algunas operaciones geométricas variables y constantes tales como la profundidad del árbol, fuentes de luz y los objetos con los cuales puede intersectar. Cada pixel es responsable de encontrar si intersecta o no con los objetos así como realizar los cálculos del color de las fuentes de luz y los rayos que reflejan y refractan de el objeto. Una vez que todos los nodos del árbol hayan realizado sus correspondientes opereaciones, podrán regresar el valor del color que ha afectado.

Selección del Patrón de Software adecuado

Basado en las especificaciones descritas anteriormente se puede realizar la selección del Patrón adecuado.

1. Análisis del diseño del problema y sus especificaciones. De acuerdo al análisis realizado de los datos y las operaciones se tiene que se pueden realizar simultaneamente debido a la independencia de los datos y de las operaciones.
2. Selección de la categoría de paralelismo El paralelismo que se realiza es un paralelismo de Actividad esto es porque se puede paralelizar por datos y por operaciones.
3. Selección de la categoría de los componentes de acuerdo a su naturaleza. Las operaciones que se realizan por cada pixel son las mismas lo cual indica que las operaciones son homogeneas.

4. Por la distribución de los datos y la naturaleza homogenea de las opereaciones que realiza cada componente, el patrón que más se adecua al problema es Manager-Workers.

Debido a la independencia de las operaciones se puede ver que se pueden hacer una serie de operaciones que trabajen simultaneamente en cada pixel.

Se hace una descripción de las funciones de cada compomponente del patrón a utilizar.

1. Manager. La función del Manager es crear Workers y distribuir entre ellos el trabajo. Maneja y recolecta los trabajos que individualmente hace cada worker.
2. Worker. El trabajo de los workers es buscar en el arreglo de objetos y realizar las operaciones de trazar un rayo, calcular el color que aportan las fuentes de luz y determinar si el rayo al intersecar con un objeto refleja o refracta.

La información que proporciona Manager-Workers se puede utilizar para realizar la descripción de la estructura de Ray Tracing.

Relación Manager Workers en el algoritmo Ray Tracing

Con la arquitectura Manager Workers todos los pixeles en la escena son distribuidos y operados independientemente por los workers. Cada uno realiza la misma operacion sobre los diferentes pixeles simultaneamente.

Todos los procesadores son activados al mismo tiempo y distribuyen y procesan la información de distintos pixeles.

Se crean todos los componentes y son activados pero estan a la espera de que el manager les asigne el arreglo de pixeles. Una vez que los datos esten disponibles para el Manager, esté distribuye los datos a los Workers conforme lo soliciten.

Cada worker recibe una copia del arreglo y procesa las operaciones sobre los pixeles, cada operacion es independiente de las otras. Una vez que el Worker terminó su actividad solicita más trabajo al Manager. Para cada paso el manager contesta con otro arreglo de pixeles o recibe resultados parciales que mandan los workers.

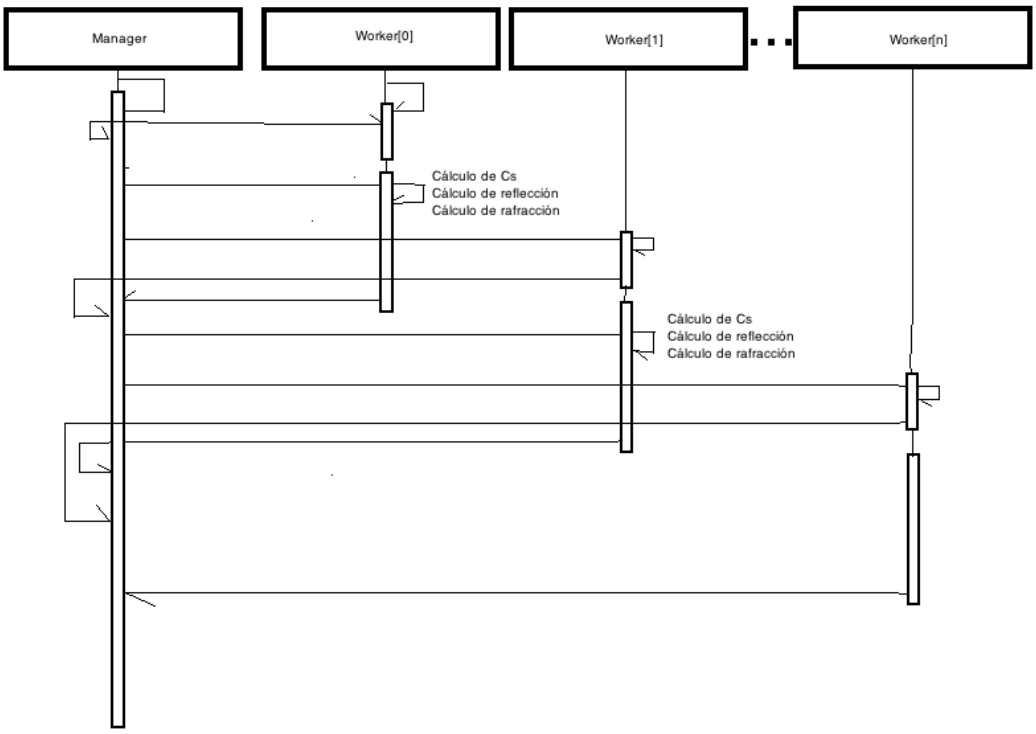

Figura 4.1: Estructura de Manager Worker aplicado a Ray Tracing.

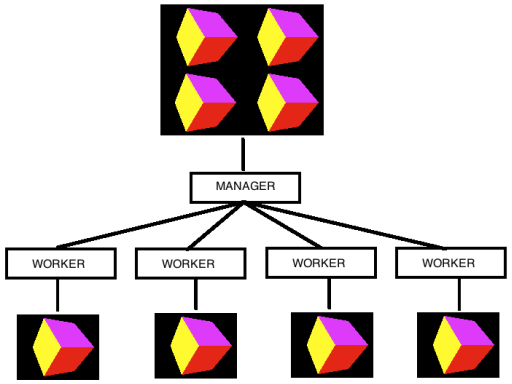

Figura 4.2: Diagrama de objetos de la división del problema

Una aplicación principal de Ray Tracing es recursiva, la cuál es representada en árboles por cada pixel, donde los nodos del árbol son los rayos y los vertices son las intersecciones. Para determinar la intensidad del pixel.

La partición del algoritmo se basa en el rayo principal. De este rayo y dependiendo del material con el que este hecho el objeto con el que intersecta puede o no haber dos rayos nuevos, un rayo de reflección y uno de refracción. De estos rayos pueden surgir otros dos rayos. Esto genera un árbol no balanceado binario. La particiones que son generadas son de tamaños desconocidos pues esto depende de los objteos que se encuentren a su paso (Se puede delimitar este tamaño indicando una profundidad máxima al árbol). En el mejor caso el árbol estará balanceado pero puede darse el caso de que el reflejo del rayo principal no intersecte con ningún objeto y el árbol quede de un lado con una profuncidad de 1 y del otro lado con la profundidad máxima que hemos determinado. Esto hace compleja la paralelización debido a que hay un desbalanceo de carga.

La etapa de la coordinación se puede separar en tres etapas:

1. Partición. En el primer paso, el nodo 0 recibe el rayo principal busca una intersección, si la encuentra un rayo reflejado genera el nodo 1 y el rayo refractado genera el nodo 2. Posteriormente y dependiendo de la profundidad del árbol el nodo 1 y el nodo 2 pueden generar los nodos 3, 4, 5 y 6 respectivamente.
2. Cálculo. Simultaneamente se calcula el color que se tiene en ese punto de intersección con respecto a las fuentes de luz.
3. Combinación. Posteriormente basados en el ejemplo los nodos 3 y 4 pasan información al nodo 1.

Coordinación

Las ventajas y desventajas de utilizar el patrón de Manager-Workers para el diseño de la coordinación de la paralelización de Ray Tracing son la siguientes:

Ventajas

- El orden y la integridad de los datos se mantienen, en todo momento, puesto que el Manager sabe qué información envía a cada worker y qué información está recibiendo de ellos.
- La independencia de operaciones permite que se pueda lograr un mejor balance de carga al momento de repartir el trabajo, esto también lo hace escalable con imagenen más grandes.

- La sincronización se simplifica porque se centraliza en el manager. La comunicación sólo se realiza entre el manager y los workers, no existen comunicaciones entre los workers, lo cual reduce el número de comunicaciones y por lo tanto se simplifica la sincronización.
- Si se realiza un análisis del diseño es relativamente sencillo mejorar el desempeño en las operaciones, pues se puede repartir de mejor forma el trabajo entre los componentes y así evitar cuellos de botella. En el caso de los workers se puede lograr que todos tengan un trabajo equitativo.

Desventajas

- Un número grande de workers puede hacer que el proceso se haga lento debido a que estos necesitan comunicarse con el manager para poder devolver información. Es posible hacer que varios pixeles sean ejecutados al mismo tiempo en un mismo nodo con el objetivo de utilizar menos workers, la desventaja en este caso sería que el manager constantemente tiene que realizar un balanceo de carga para hacer una repartición de trabajo adecuada.
- Es muy importante tomar en consideración el desempeño que el manager realiza, esto es debido a que los workers pueden terminar muy rápido de realizar sus actividades y estar pidiendo por trabajo o entregando resultados. Si el manager no puede atenderesas peticiones el proceso se puede volver muy lento.
- Debe de haber una coordinación entre el manager y los workers. Hacer una determinación de que opción es mejor para no excedernos en workers y mantener muy ocupados a los manager pasivos o activos. Debe de haber una coordinación respecto al tamaño de los datos con el objetivo de mejorar el rendimiento del programa. A pesar de estas fallas identificadas, se tiene que hacer un manejo de los errores, esto es debido a las fallas de ejecución de los workers, fallas de comunicación entre el manager y el worker o fallas de inicio paralelo de los workers.

4.2.3. Comunicación

Se define la estructura de comunicación que coordina la ejecución de los manager y workers. Como workers solamente esta permitido comunicarse con el

manager para obtener más trabajo, definir una estructura de comunicación apropiada entre el manager y los workers es una tarea clave. La estructura de comunicación debe permitir interacciones entre el manager y cada worker para pedir datos y una vez procesados entregarresultados al manager. Algunos parámetros importantes para considerar es el tamaño y el formato de los datos, la interfaz para responder al servicio de os datos y el criterio de sincronización. Comúnmente se usa coordinación síncrona en los patrones Manayer-Workers, aunque la implementación de la estructura de comunicación depende de el lenguaje usado.

En general si el lenguaje contiene comunicaciones básicas e instrucciones de sincronización, la estructura de comunicación se puede implementar relativamente fácil siguiendo la aproximación de solo un elemento en la cuál un solo archivo provee completamente la implementación para el componente de comunicación. Sin embargo es posible reutilizar el diseño en mas de una aplicación, es conveniente considerar un aploximación más flexible usando subsistemas de comunicación configurables para el intercambio de diferentes tipos y tamaño de los datos.

Al iniciar cada worker establece conexión con el manager. Inmediatamente después, cada worker pide datos de el manager. Y es aquí cundo el paralelismo del algoritmo ocurre, pues esta función es ejecutada en cada worker en paralelo. Durante la operación, cuando se proveen los datos se operan con estos, cada worker regresa un resultado parcial a la estructura de datos de los resultados, se repite esto hasta que la profundidad del árbol máxima ha sido alcanzada.

Para Manager-Workers existe la opción de utilizar el patrón Local Rendezvous para describir una comunicación uno a uno, bidireccional, y síncrona. Permite el intercambio de informacion entre el manager y los workers. Este patrón describe un componente de comunicación capaz de emitir llamadas individuales pero síncronas al componente central, el cual lee y escribe obre la estructura de datos. El manager encapsula una estructura de datos que se puede leer por partes de manera síncrona o escrita por un solo worker. Se permite un flujo de datos del manager al worker y viceversa. Se permite que todos estos componentes se ejecuten simultaneamente, requieren comunicación síncrona entre ellos durante cada llamada. Este patrón se considera local, debido a que sus componentes son diseñado para existir y ejecutarse en una memoia compartida.

Otra opción es utilizar el patrón Rendezvous el cual describe el diseño de una comunicación remota, punto a punto, bidireccional y síncrona que permite el intercambio de infomacion entre el manager y el worker. El manager encapsula los datos en una estructura, se puede leer o escribir síncronamente desde un worker remoto y viceversa. Rendezvous es considerado remoto debido a que sus componentes son diseñados para existir y ejecutrase en una memoria distribuida.

4.2.4. Sincronización

La división de datos y las estructuras de comunicación definidas previamente son evaluadas en términos de mejorar los requerimientos. Si es necesario el tamaño de las piezas de datos es cambiado, modificando la granularidad del sistema. Las piezas de datos son combinadas para hacerlas más grandes o divididas en piezas más pequeñas para mejorar el rendimiento o para reducir los costos de comunicación. Debido a las características inherentes de este patrón, procesar es automaticamente balanceado entre los workers, pero la granularidad es modificada para balancear el procesamiento entre los manager y workers. Si las operaciones realizadas por los workers son lo suficientemente simples y reciben relativamente pequeñas cantidades de datos, ellos pueden permanecer inactivos mientras el manager esta ocupado tratando de servir a sus peticioens. De la misma manera, si las operaciones del worker son muy complejas, el manager tendrá que usar un buffer para almacenar los dato pendientes a ser procesados. El balance de carga entre el manager y el worker puede ser alcanzado modificando la granularidad de la división de datos.

En el mejor caso el hardware permite a cada componente ser asignado a un procesador con suficientes enlaces de comunicación para operaciones eficientes. Sin embargo, generalmente el número de componentes que se define suele ser más grande que el número de procesadores disponibles. Es usual poner un número similar de workers sobre cada procesador. Para mantener la estructura tan balanceada como sea posible, el manager puede ser ejecutado sobre un procesador dedicado, o al menos sobre un procesador con un número reducido de workers. La fuerza más competitiva de este patrón es maximizar el uso de los procesadores y minimizar el costo de la comunicación. El mapeo se puede especificar estáticamente o dinámicamente, permitiendo un mejor balance

de carga. Como regla general los sistemas paralelos basados en el patrón de Manager-Workers mejora razonablemente sobre un procesador MIMD (por sus siglas en inglés multiple-instruction, multiple-data) aunque podría ser difícil de implementar sobre un SIMD (por sus siglas en inglés single-instruction, multiple-data).

Un monitor es un mecanismo de sincronización basado en el concepto de un objeto que encapsula variables compartidas. Dentro del monitor, las variables compartidas son etiquetadas como recursos privados, la única manera de manipularlos es llamando a los métodos de la interface que opera sobre las variables compartidas. Por lo tanto es la única manera de intercambiar datos entre dos o más componentes de software concurrente, paralelo o distribuido ejecutandose sobre una plataforma de memoria comartida.La exclusión mutua es implícita y garantizada por el compilador, permitiendo que solo un proceso sea activado a la vez dentro del monitor, esto es ejecutando uno de los métodos. No se puede asumir ninguna inicialización.

El lenguaje de programación de Java puede crear y ejecutar hilos sobre el mismo procesador o sobre diferentes procesadores. Permite la comunicación entre hilos, Java especifica el modificador synchronized". Para implementar un monito como un objeto en Java, el modificador synchronized es usado por todos los metodos de la clase en la cual solo un hilo debe ser ejecutado a la vez. Estos metodos son normalmente declarados como publicos y modifican a las variables compartidas declaradas como privadas dentro del monitor. Los métodos también pueden declararse privado, siempre y cuando el acceso público al monitor consista en llamadas a varios de estos métodos sincronizados.

En Java cada objeto es asociado a un candado. Un hilo que invoca a un método con el modificador synchronized en un objeto primero debe obtener el candado del objeto antes de ejecutar el código del método así tiene exclusión mutua con invocaciones de otros hilos. Solo un hilo se puede ejecutar en un método de synchronized dentro del objeto, mientras tanto los otros hilos están bloqueados.

En contexto del idioma de un monitor, un programa concurrente, paralelo o distribuido en el cual dos o más componentes se ejecutan simultaneamente sobre una plataforma de memoria compartida comunicandose por variables compartidas. Cada componente de software accede al menos una vez a la sección crítica, esto es, una secuencia de instrucciones que accede a las variables compartidas, esto es una secuencia de instrucciones que accede a las variables

compartidas. Al menos un componente escribe en las variables compartidas.

Para preservar la integridad de los datos es necesario dar un conjunto de componentes sincronizados y de acceso exclusivo a las variables compartidas para un número arbitrareo de operaciones de lectura y escritura.

Para aplicar monitores se deben tomar en consideración las siguientes fuerzas:

- Un conjunto de componentes paralelos se ejecutan no deterministicos a velocidades relativamente diferentes. Todos deben actuar sincrona e independientemente de los otros tanto como sea posible.
- La sincronización se lleva a cabo por operaciones de inspección y asignación, las cuales deben ser atómicas o inivisibles.
- Cada componente debe ser capaz de executar el código asociado con una sección crítica, accesando por medio de variables compartidas si y solo si el acceso es seguro. Los demás componentes deben bloquearse y esperar a que el componente termine su acceso.
- Los valores de las variables compartidas deben preservar su integridad durante toda la comunicación.
- El uso correcto de las variables compartidas se debe reforzar.

Un monitor es un objeto que incorpora exclusión mutual y capacidades de sincronización de hilos. Están integrados al lenguaje de programación, el compilador genera el código correcto para realizar la implementación del monitor. Solo se puede activar un hilo a la vez, con activo se hace referencia a ejecutar un método del monitor. Los monitores tienen variables condicionales, en las cuales un hilo puede esperar si las condiciones no son las corretas para continuar ejecutandose en el monito, otros pueden entrar al monitor y quiza cambiar el estado del monitor. Si las condiciones son las correctas, el hilo puede mandar una señal de "hilo en espera", se mueve a la cola de listos para regresar al monitor cuando este libre.

Java usa la palabra synchronized"para indicar que solo un hilo a la vez se puede ejecutar en cierto orden en un método del objeto que representa al monitor. Un hilo puede llamar a la fución wait() para bloquear y salir del monitor hasta que reciba una llamada de notify() o notifyAll() que pone al hilo en la cola de listos para después accesar al monitor cuando sea calendarizado. Un hilo que ha enviado un señal no tiene garantía de ser el siguiente hilo a ser ejecutrado

dentro del monitor comparado a uno que es bloqueado en una llamada del método de sincronización del monitor. Tampoco se garantiza que el hilo que ha esperado por más tiempo sea el que despierta con una llamada notify(); se escoje un hilo arbitrario por la JVM. Finalmente cuando se llama la función notifyAll() la cuál mueve a todos los hilos que estaban esperando en la lista de listo, el primer hilo en regresar al monitor no es necesariamente el que ha esperado más. Cada monitor en llava tienen las siguientes instrucciones: wait(), notify() y notifyAll(), estas llamadas son parte de los métodos de sincronización. La función wait() hace esperar al hilo que envía la señal, notify() envía un señal a algún hilo para entrar a la cola de listos y finalmente notifyAll() envía una señal a todos los nodos para regresar a la cola de listos. Los métodos que son estáticos también se pueden sincronizar. Existe un candado asosiado a la clase el cuál se debe obtener cuando un método de sincronización estática es llamado.

Usualmente todos los métodos de acceso públicos, los métodos y los servicios de accesos son sincronizados. Pero un monitor de Java puede diseñarse con algunos metodos sincronizados y otros no. Los métodos no sincronizados pueden se de acceso público y llamar a los métodos sincronizados, los cuales son privados.

Se realizo un experimento para determinar si los monitores de Java son "Signal and exit.º "Signal and continue". Usan "Signal and continuel". Cuando un hilo executa la llamada notify(), Java no necesariamente mueve el hilo que ha esperado mayor tiempo a la cola de listos, Java permite irrumpir.

4.2.5. Hardware

Basados en la información proporcionada por Pancake y Bergmark [7] y en los artículos mencionados en el capítulo 3, la opción para resolver la paralelización de estos algoritmos es memoria compartida. Las principales razones es porqeu si se ocupa memoria compartida no se tiene una restricción en cuanto a tamaño de la imagen y puede ser más sencillo el acceso a los datos, ya que no se hará una copia de todos ellos en cada memoria del procesador y cada uno de los procesadores podrá acceder a las localidades que necesite cuando requiera información.

4.2.6. Lenguaje

El lenguaje que se utiliza para hacer las pruebas es Java esto es debido a que con este lenguaje se pueden manipular gráficos de una forma no tan compleja y se puede hacer una mejor comunicación en paralelo.

4.3. Resumen

En este capítulo se presenta la solución al problema, en general como es que se realiza el diseño del algoritmo en paralelo por medio del patrón manager workers. Se describe la forma en la que se comunicarán, coordinarán y sincronizarán estos componentes. También se hace una descripción de la repartición de la carga que se hace entre los workers.

Capítulo 5

Evaluación de la paralelización de Ray Tracing

En este capítulo se presentan los experimentos respecto al algoritmo paralelo de Ray Tracing diseñado con base en el patrón manager workers. Así mismo se presentan experimentos para determinar la complejidad de una escena, el número de workers con el cuál se obtienen mejores resultados, el orden en el cual entregan trabajo los workers y finalmente se hace un experimento para mejorar los resultados con una nueva distribución del trabajo (diferentes tamaños de datos).

5.1. Complejidad de la imagen

La complejidad de la imagen se mide de acuerdo a los objetos que se encuentren en ella. Por objetos todas las figuras como esferas, triángulos, y cubos pero también se puede tener en la escena otros elementos que se pueden considerar como objetos y que se deben tomar en cuenta. Ejemplos de ello son pisos, paredes, la cámara (o el punto de vista) e inclusive la luces.

A continuación se muestra una serie de imágenes y la complejidad de cada una, se considera que en todos los casos hay una cámara y una fuente de luz.

5.2. Experimentos

5.2.1. Workers vs comunicaciones

Un objetivo de paralelizar un algoritmo es lograr una mayor velocidad repartiendo el trabajo en más de un procesador. Sin embargo, y debido a que las comunicaciones entre los participantes toman tiempo en la paralelización, en este caso entre el manager y los workers, hay un momento en que el costo de estas comunicaciones provoque que el algoritmo paralelo sea igual o más lento que el algoritmo secuencial. Por tal motivo, los primeros experimentos realizados, están dirigidos a encontrar el número de workers con los que el algoritmo reparte de la mejor manera el trabajo, sin que el tiempo de las comunicaciones afecte el tiempo total de la ejecución de Ray Tracing.

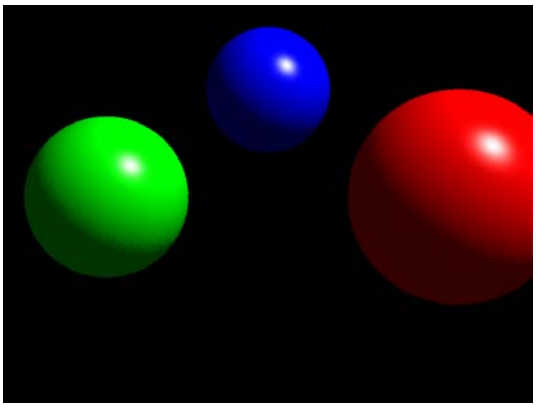

Figura 5.1: Complejidad 5.

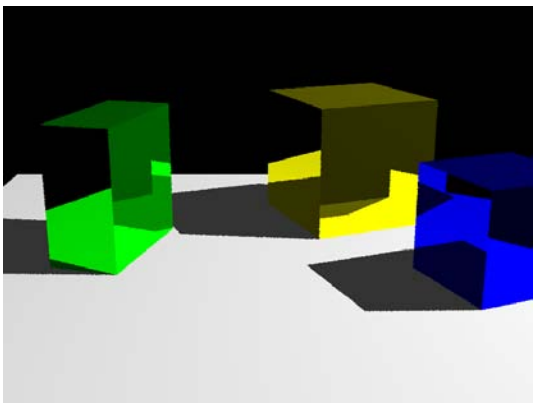

Figura 5.2: Complejidad 6.

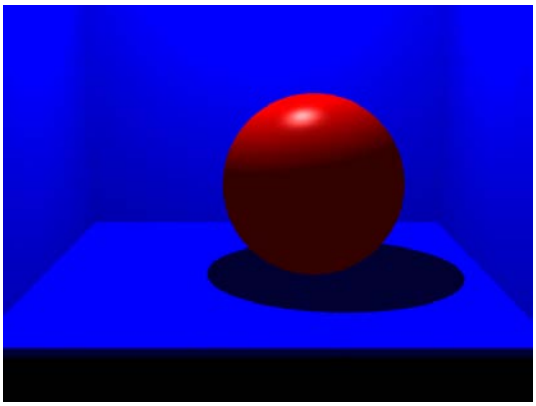

Figura 5.3: Complejidad 7.

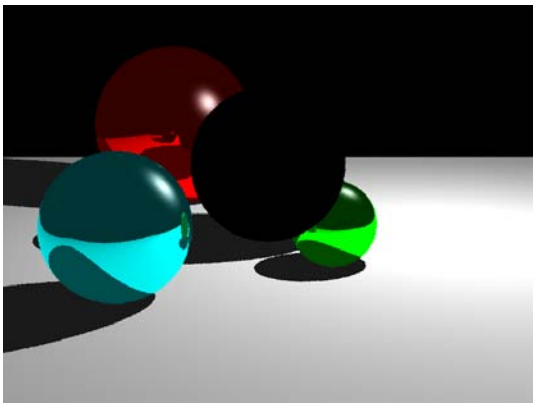

Figura 5.4: Complejidad 7.

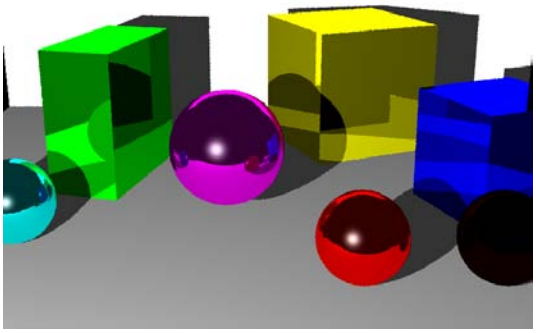

Figura 5.5: Complejidad 11.

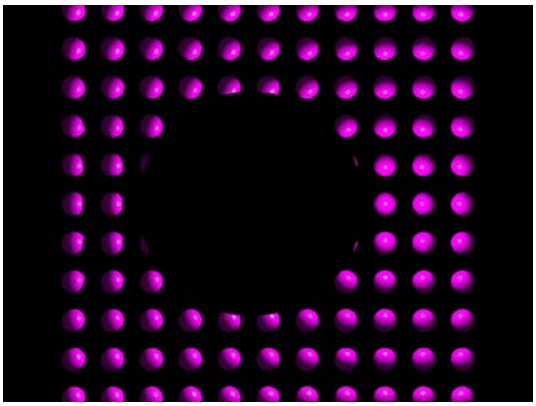

Figura 5.6: Complejidad 124.

Esfera

La imagen con la cuál se trabaja es de 800 por 600 pixeles y la repartición de trabajo se realiza por filas, es por eso que solo se puede trabajar con dividendos de 600, los dividendos de 600 son 24 que son los que se muestran en la siguiente gráfica, se realizan 10 ejecuciónes por cada uno de los dividendos. En total se analizan para este primer experimento 37464 datos. Una vez que se ejecutan 10 veces con cada dividendo de 600 se obtiene el valor mínmo, máximo y el promedio. Lo que se muestra en la gráfica es el promedio de tiempo de 10 ejecuciones que tardan los workers completar la tarea asignada por el manager.

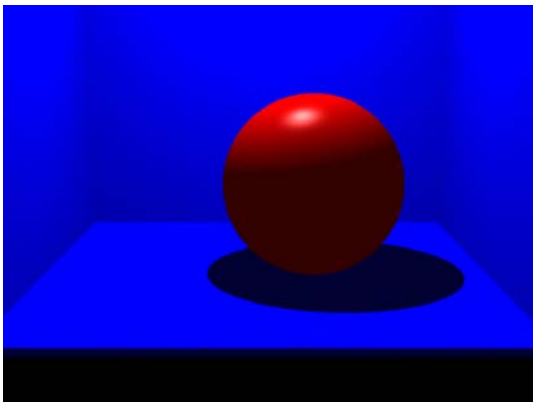

Figura 5.7: Esfera.

Al obtener datos sobre el tiempo mínimo en que un worker desempeña su trabajo y el tiempo máximo, es visible que la diferencia más notable es cuando

son más de 50 workers, en este caso siendo el que mayor diferencia tiene es con 600 workers. La tabla presenta los datos de mínimo, máximo, promedio y balance para las 24 pruebas que se realizan, los tiempos son de las 10 ejecuciones. Coloreado en verde se encuentran los resultados mínimos y en rojo los tiempos máximos.

Workers	MINcorridas	MAXcorridas	AVEcorridas
1	3623	3916	3772.6
2	2084	2296	2205.3
3	2038	2342	2123.3
4	1840	1959	1902.5
5	1590	1924	1727.1
6	1706	2112	1826.5
8	1615	2274	1845.6
10	1669	2338	2068.9
12	1810	2822	2098.2
15	2253	3190	2687.1
20	2020	2856	2494.3
24	1980	3101	2546.5
25	2179	4221	2996
30	2311	4399	3376
40	2759	4469	3413.2
50	3161	4794	3933.2
60	3096	4894	4089.4
75	2894	5393	4307.5
100	3380	5882	4581
120	4031	5685	4777
150	3642	6769	5679.1
200	5357	8461	6552.1
300	4325	7645	6554.5
600	3930	10935	9199.1

Figura 5.8: Tabla eficiencia Workers.

En general las tres gráficas muestran resultados similares, tienen la misma tendencia. Particularmente, en cuanto al tiempo promedio en la gráfica se puede apreciar que cuando se ejecuta el programa con 5 workers resulta tener el mejor tiempo promedio y hasta con 40 workers los tiempos en paralelo son menores al tiempo secuencial, adicionalmente, con 12 workers el problema de comunicaciones se incrementa, a partir de 50 workers el resultado de la ejecución en paralelo comienza a ser más lento que el secuencial.

En la gráfica de tiempos mínimos se puede observar que tiene más cambios, algunos workers obtienen buenos resultados con estos tiempos, aún así sigue siendo mejor con 5 workers y muy cerca con 8 workers e incluso con 600 workers

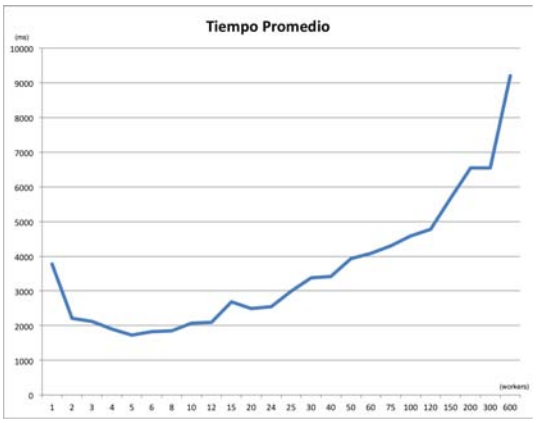

Figura 5.9: Eficiencia Workers.

el valor mínimo es cercano al algoritmo secuencial.

Figura 5.10: Tiempos mínimos por Workers.

Una de las gráficas más representativas puede ser la gráfica de tiempos máximos, esto es debido a que el algoritmo será tan rápido como el worker más lento. En la gráfica se observa que el mejor tiempo sigue siendo con 5 workers con valores cercanos para 4 y 6 workers, para estos casos mejora el doble de tiempo del algoritmo secuencial.

Finalmente se presenta una gráfica que sirve como apoyo para la evaluación del balance de carga, se hace una diferencia entre el tiempo máximo y el tiempo mínimo en que cada worker termina una tarea asignada, este balance hace una diferencia notable cuando son 600 workers, se tiene un mejor balance con menos de 60 workers.

Figura 5.11: Tiempos máximo por Workers.

Figura 5.12: Balance de Carga.

Cubo

Para tener un panorama más amplio de cómo es que está trabajando el algorítmo se hace la misma prueba con una imagen del mismo tamaño pero con más objetos dentro de la imagen. Se ocupa la misma repartición de la imagen y los resultados obtenidos son los siguientes.

En este caso y tal como lo muestran los datos se obtuvieron mejores resultados trabajando con 4 workers, un dato interesante que se puede extraer de la tabla es que se obtuvo un resultado mínimo con 20 workers. Los resultados que toman mayor tiempo son para 200, 300 y 600 workers.

La tabla de tiempo promedio muestra que a partir de 4 workes hasta 12 el

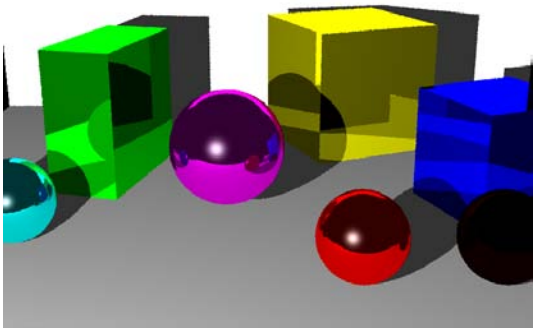

Figura 5.13: Cubos.

Workers	MIN	MAX	PROMEDIO	BALANCE
1	6642	9916	7999.9	3274
2	4704	5799	5146.6	1095
3	3962	4571	4206.5	609
4	3884	4193	4062.3	309
5	3950	4386	4073.8	436
6	3865	4473	4137.8	608
8	3777	4576	4108.1	799
10	3736	4536	4111.5	800
12	3755	4917	4199.1	1162
15	3989	5204	4585.7	1215
20	3596	5567	4650.1	1971
24	4191	5554	4824.3	1363
25	4252	6111	5140.7	1859
30	4285	5970	5125.7	1685
40	5218	6823	5811.1	1605
50	4924	7016	5999.1	2092
60	4780	8201	6761	3421
75	5831	9533	7437.4	3702
100	5777	8083	6860.7	2306
120	5831	8840	7122.4	3009
150	7055	11687	8955.6	4632
200	5389	12062	8800.7	6673
300	6785	13415	9743.8	6630
600	7470	13202	10008.2	5732

Figura 5.14: Tabla de tiempos en imagen de cubos.

tiempo promedio disminuye el doble que el algoritmo secuencial. Esto es debido a la complejidad de las imágenes, los workers ocupan más tiempo en regresar la información provocando que el manager reciba la respuesta periodicamente y teniendo más tiempo para el procesamiento del algoritmo, haciendo menos significativo el tiempo de comunicaciones.

En la tabla de mínimos se puede ver que solo con 150 y 600 workers se obtienen resultados mayores al secuencial esto indica que solo en casos dónde hay un

Figura 5.15: Tiempo promedio en imagen de cubos.

gran número de workers los resultados incluso para la ejecución mínima de una tarea no son los más adecuados.

Figura 5.16: Tiempo mínimo en imagen de cubos.

En la tabla de tiempo máximo es muy representativo cómo es que influye la complejidad de la escena en el funcionamiento del algoritmo ya que se puede observar que hasta con 120 workers sigue siendo más efectivo el algoritmo en paralelo que el algoritmo secuencial, en este caso el mejor trabajo es realizado con 4 workers.

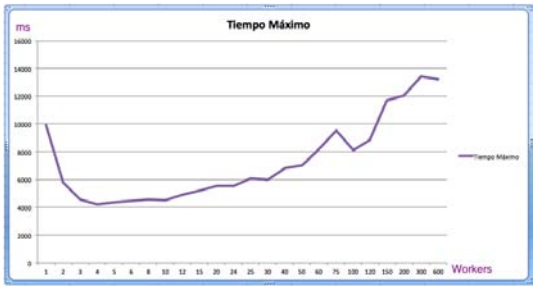

Figura 5.17: Tiempo máximo en imagen de cubos.

En cuanto al balance de carga con el que se está trabajando los resultados no

son variables y esto se debe a que si se observa la imagen de los cubos se puede apreciar que la complejidad de la escena esta concentrada en solo una porción de la imagen.

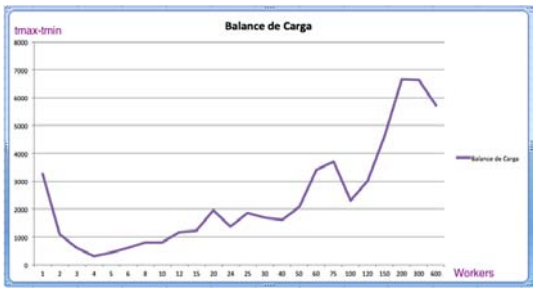

Figura 5.18: Balance de carga en imagen de cubos.

Esferas

Tomando en cuenta que las imágenes de la esfera y los cubos no tienen una diferencia muy grande en cuanto a complejidad de los objetos, se toma una imagen que contiene una mayor cantidad de objetos para trabajar con ella. La imagen tiene 124 objetos (tomando en cuenta cubos, esfera, luces y cámara) y se aplica la misma prueba con 24 opciones y 10 ejecuciones por cada una, estos son los resultados obtenidos.

En la tabla se puede observar en fondo verde que se tienen tiempos mínimos cuándo los workers son igual a 8 y los tiempos máximos se marcan con 1 o con 600 workers. Estos son mejores resultados con una imagen compleja puesto que se hace un balance entre las comunicaciones y el tiempo en el que trabajan los workers.

En la gráfica de tiempo promedio se puede apreciar cómo es que se comporta la tendencia, con 8 workers es el resultado más eficiente y aún así todos los resultados del algoritmo paralelo son mejores que los resultados en el algoritmo secuencial.

En la gráfica de máximos se puede ver que solo dos casos son peores que el paralelo, estos casos son para 300 y 600 workers. Siendo el mejor caso con 8 workers.

En la gráfica de mínimos, todos los casos en paralelo son mejores que en el secuencial. El más rápido sigue siendo con 8 workers

Workers	Mínimo	Máximo	Promedio	Balance
1	6102	6310	6179.9	208
2	3583	3842	3709.7	259
3	2685	3466	2947.7	781
4	2427	2703	2584.7	276
5	1961	2441	2111.1	480
6	1301	1849	1534.2	548
8	888	1125	996	237
10	918	1291	1103.7	373
12	1040	1389	1204.3	349
15	1241	1928	1566.8	687
20	1426	2022	1746.2	596
24	1661	2423	2058.1	762
25	1770	2096	1916.3	326
30	1615	2576	2207	961
40	2145	3732	2817.8	1587
50	1823	4252	3334.4	2429
60	1852	4922	3235.9	3070
75	2504	5060	3804.3	2556
100	1571	5991	4352.5	4420
120	4327	5859	4856.3	1532
150	4179	5176	4620.5	997
200	4253	6400	5069.4	2147
300	4204	7815	5570.5	3611
600	3977	8128	6059.7	4151

Figura 5.19: Tabla de tiempos en imagen de esferas.

Finalmente en la gráfica de balance de carga, el mejor caso es con solo un worker y muy de cerca con 8 workers.

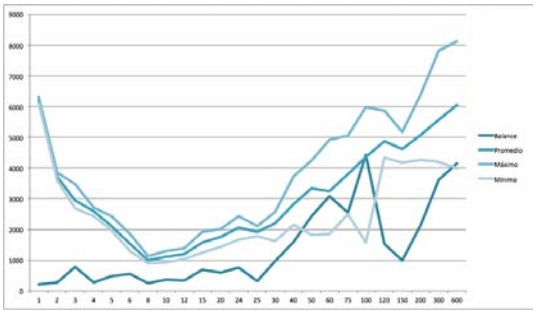

Figura 5.20: Gráficas de Tiempos promedio, máximos, mínimos y balance.

Esferas con la misma complejidad

Este experimento consiste en crear una imagen que tenga la misma complejidad en todos sus segmentos con el objetivo de que se pueda relacionar la complejidad con el número de workers más eficiente. Se eligen cinco franjas para dividir la escena y se analiza una imagen con cinco esferas del mismo tamaño y con una luz enfrente de ellas, estas esferas se reparten en la escena

a la misma distancia, tomando en cuenta que la imagen central tiene mayor complejidad debido a las esferas que se encuentran arriba y abajo de ella, se agregan dos esferas más, una en la parte superior y su correspondiente en la parte inferior de tal forma que todas las esferas son influenciadas por una esfera en la parte superior y una más en la parte inferior. La siguiente figura muestra la imagen completa una vez que se aleja la cámara para poder ver la imagen completa.

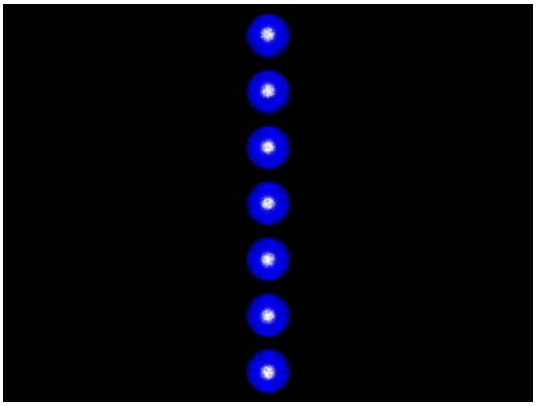

Figura 5.21: Escena completa con cámara alejada.

El experimento que se realiza nuevamente es con 24 opciones de workers y se realizan 10 ejecuciones para determinar cual es la cantidad de workers más eficiente. En la siguiente figura se muestra la imagen original ya con la cámara en la posición definida previamente y posteriormente la imagen una vez que fue procesada por 600 workers como referencia.

Figura 5.22: A la izquierda se encuentra la imagen que se procesa. A la derecha la imagen procesada por 600 workers.

Los resultados obtenidos se muestran en la siguiente tabla. En verde están marcados los mejores tiempos y en rojo los peores. Se puede observar que los mejores resultados se obtiene cuando el programa se ejecuta con 4 workers,

el balance es mejor con 8 workers seguido muy de cerca con 4 workers. Lo peores tiempos tanto para el balance de carga como para el tiempo máximo se encuentran con la ejecución en secuencial. Finalmente los peores valores para mínimo y promedio se encuentran con los workers 150 y 600 respectivamente.

WORKERS	MÍNIMO	MÁXIMO	PROMEDIO	BALANCE
1	1980	6596	2546.4	4616
2	1338	1570	1399.3	232
3	1397	1824	1519.8	427
4	1276	1485	1394.9	209
5	1510	1998	1770.3	488
6	1494	2140	1731.3	646
8	1413	1614	1495.8	201
10	1598	2283	1963.7	685
12	1488	2509	1804.6	1021
15	1638	2093	1867.6	455
20	1659	2091	1872	432
24	1375	2607	2175.5	1232
25	1762	2576	2148.3	814
30	1527	3364	2300.5	1837
40	1699	3403	2538.8	1704
50	1701	4618	2483.6	2917
60	1772	3010	2248.7	1238
75	1867	3260	2607.9	1393
100	1778	3748	2698.9	1970
120	1827	3555	2834.4	1728
150	2333	5479	2956.4	3146
200	2099	5320	3281.5	3221
300	2242	5160	3266.5	2918
600	1733	4856	3494.5	3123

Figura 5.23: Tabla de tiempos en imagen con igual complejidad.

A continuación se muestra la graficación de los resultados.

En el caso de la gráfica de mínimos casi todos los resultados son mejores que en el secuencia exepto con 150 y 200 workers. Los mejores casos son con 2, 4 y 24 workers.

Para la gráfica de máximos todos los casos son mejores que el secuencial, los mejores casos son para 4, 2, 15 y 20 workers respectivamente.

En la gráfica de promedios con menos de 20 workers el algoritmo paralelo es mejor que el secuencial, para el caso de workeres mayores a 20 el secuencial es mejor opción. Siendo las mejores opciones 4, 2, 24 y 8 respectivamente.

Finalmente en la gráfica de balance de carga, todos los casos en paralelo vuelven a ser mejores que el secuencial, teniendo los mejores y casi el mismo valor

para los workeres de 2, 4 y 8. Los workers 20 y 15 también presentan buenos resultados.

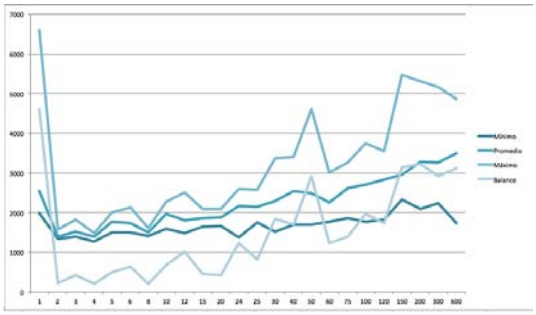

Figura 5.24: Tabla de tiempos en imagen con igual complejidad.

Este experimento es muy representativo ya que se el hecho de poner la misma complejidad para los objetos ayuda a determinar que la forma de distribución de la escena impacta directamente en los workers más eficientes, en este caso la imagen es simétrica para el caso de 2, 4 y 8 workers, al ser distribuida casi de la misma forma asegura que con estos workers el resultado sea más eficiente.

5.2.2. Orden de los datos

En este experimiento lo que se busca es saber es en qué orden los workers están regresando el trabajo. Se tomo como imagen la esfera roja y se realizarón 50 ejecuciones, las 50 ejecuciones se hicieron con 5 workers que según los resultados del experimento anterior, es la mejor opción para este algoritmo con la imagen de la esfera. La siguiente figura muestra los resultados.

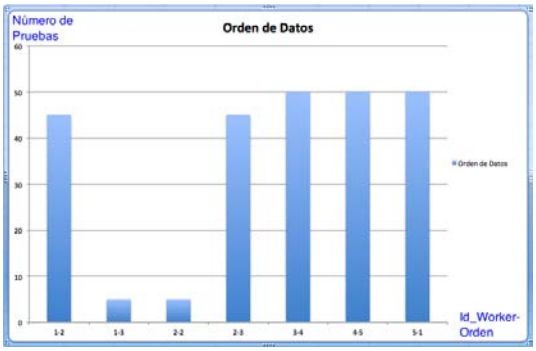

Figura 5.25: Orden de los datos.

En la gráfica se muestra que para los workers 3, 4 y 5 los resultados son siempre constantes, siempre regresan en el mismo orden, para los workers 1 y 2, 45 de 50 ejecuciones son iguales dándole al worker 1 la posición 2 y solo 5 veces le da la posición 1, de manera complementaria al worker 2 le da la posición 3 para 45 ejecuciones y la 2 en 5 de los casos.

Este experimento da como resultado solo dos imagenes diferentes manteniendo en 45 ejecuciones constantes resultados. Las imagenes que se obtuvieron se muestran a continuación.

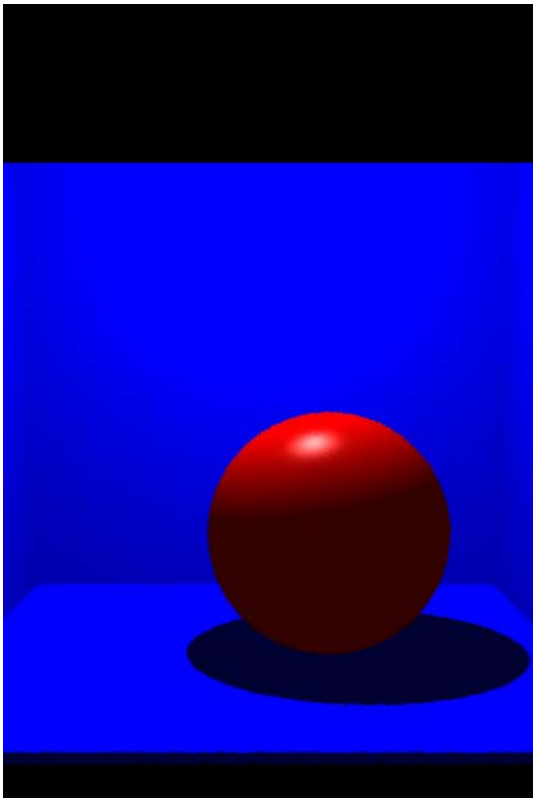

Figura 5.26: Imagen de 45 ejecuciones.

De igual forma que en el experimento uno se hace la prueba de el orden de los datos con la imagen de cubos para comprobar si la complejidad de la escena altera el orden en el que aparecen los datos.

Según se muestra en la tabla se puede ver que el worker 1, 3 y 5 siempre quedaron en la misma posición durante las 50 ejecuciones, ocuparon la posición 1, 5 y 2 respectivamente. No sucedio lo mismo con los workers 2 y 4 que 46

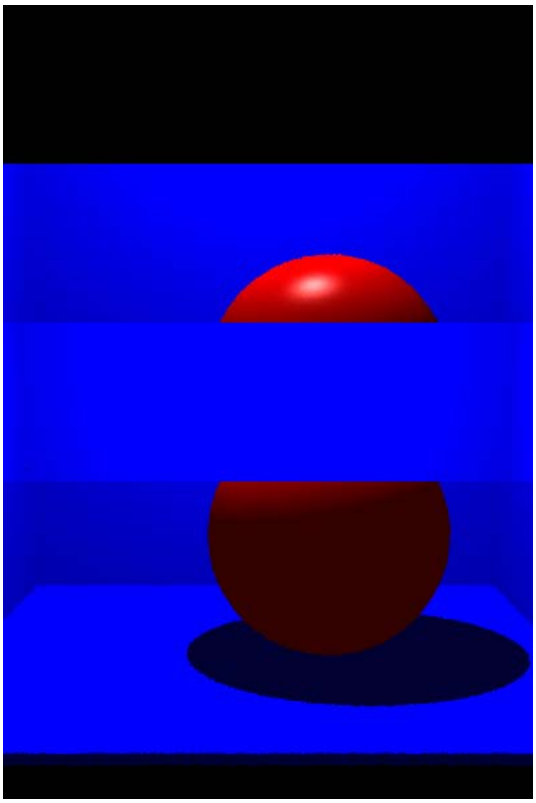

Figura 5.27: Imagen de 5 ejecuciones.

veces quedaron en las posiciones 3 y 4. Y 4 veces quedaron en las posiciones 4 y 3. Se puede concluir de este análisis que aún aumentando la complejidad de la escena el cambio no fue muy grande en el orden de hecho fue un poco más constante que con la imagen de la esfera que es más simple.

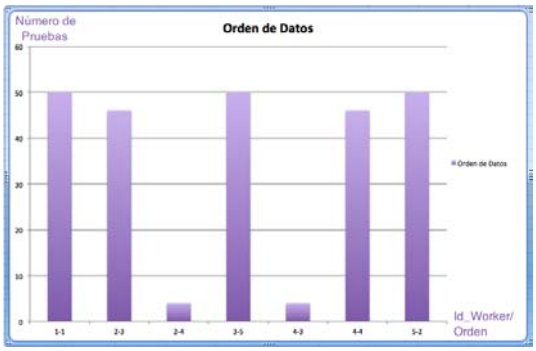

Figura 5.28: Orden de los datos para el cubo.

Las dos variantes de las imagenes que resultaron de las 50 ejecuciones con los cubos, son las siguientes.

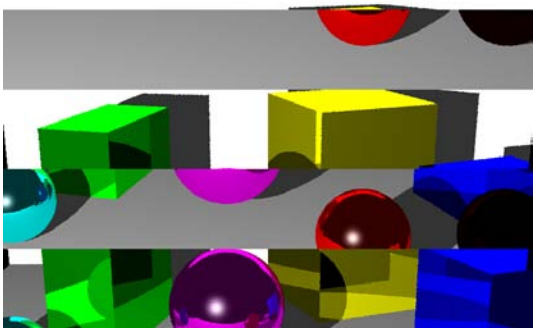

Figura 5.29: Imagen de 46 ejecuciones.

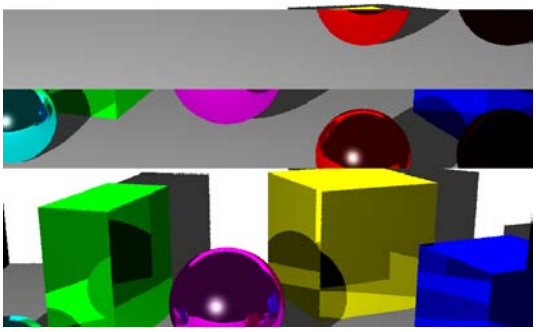

Figura 5.30: Imagen de 4 ejecuciones.

De acuerdo a la complejidad de la imagen ahora se aumenta drasticamente con una complejidad de 124 objetos y se realiza nuevamente el experimento 2 en el cuál se puede observar que si hay una gran variedad de en el orden de los datos. En general, el worker 1 y 5 comparten la posición 1 y 2, el worker 2, 3 y 4 tienen la posición 3, 4 y 5.

Tomado en cuenta el experimento con la misma franja, se realizo el mismo experimento y se obtuvieron resultados muy variables. Para empezar todos los workers estuvieron en todas las posiciones lo cuál se puede explicar debido a la no determinación de la paralelización y al acceso aleatorio a la memoria. Dónde el worker con id 1 tuvo más repeticiones en la posición 3 con 18 mientras

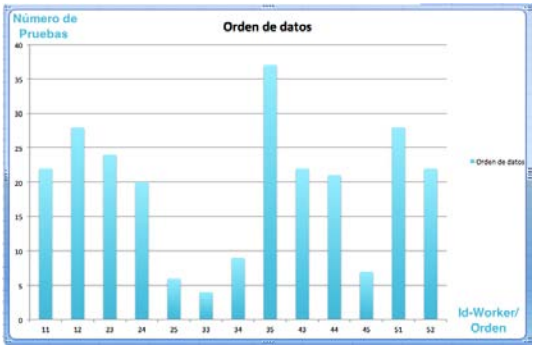

Figura 5.31: Orden de datos para esferas.

que el worker 1 y el 5 en la posición 5 y 3 respectivamente tuvieron el menor número de repeticiones 4 cada uno. Lo cual indica que la complejidad de la franja determina la constancia de la posición.

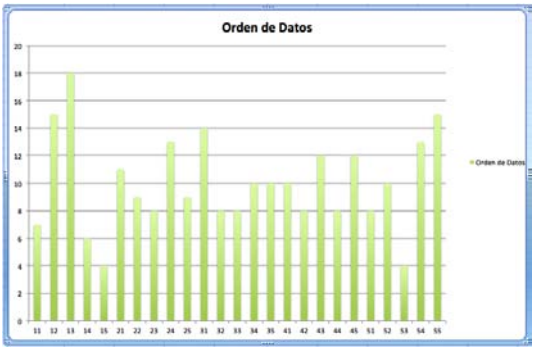

Figura 5.32: Orden de datos para franjas repetidas.

Utilizando la imagen de las eferas iguales por franja para 5 workers obtenemos un resultado muy similar al de mandar el mismo segmento de la imagen. Esto asegura que la complejidad de la imagen es la misma dando como resultado que el orden de los datos es aleatorio y todos los workers están al menos 5 veces en una posición.

5.2.3. Distribución del trabajo

Hasta el momento se han hecho experimentos respecto a la complejidad de la imagen tanto para la escena completa como por franja. El análisis de esta complejidad se hace partiendo de la escena en 3D y de su descripción a partir

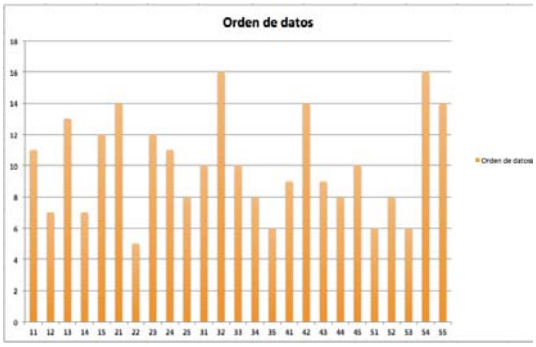

Figura 5.33: Orden de datos para escena con misma complejidad.

de un archivo con formato x3d. También se han hecho experimentos respecto al número de workers con el cuál se ejecuta en menor tiempo el algoritmo con la misma imagen. En este experimento se puede observar que los workers llegan en tiempos distintos y eso se atribuye a la diferencia de complejidad entre cada franja, hay workers que tienen más trabajo que otros. Una vez que se ha observado la relación de la complejidad en la distribución respecto al tiempo en el que terminan los workers, se puede proponer una distribución en la cuál el trabajo que realice cada worker tenga un balance de carga que permita una ejecución en menor tiempo.

Para este experimento se ocupa de igual forma una imagen de 600x800 pixeles, la imagen tiene una complejidad de 122 objetos calculado apartir de la descripción de la imagen. Debido a que de esa imagen se calcula con cuántos workers es más eficiente la ejecución del algoritmo, se decide compararlo contra el mejor caso. En este caso el que presenta un mejor resultado es ejecutando el algorítmo con 8 workers.

La siguiente tabla muestra la distribución actual con los ocho workers y su respectiva complejidad por franja, en la segunda parte de la tabla se muestran los mismos ocho workers pero con la nueva distribución y el nuevo cálculo de complejidades.

La complejidad que se mide a través de la descripción de la imagen es la principal característica que se considera cuando se hace una distribucjión en los workers, sin embargo cuando la complejidad entre franjas no es muy variable o tiene de diferencia el mínimo posible por franja, otra característica que se puede tomar en cuenta es la probabilidad de intersecciones. Particularmente en

	Igual distribución		Distinta distribución	
Workers	índices	Complejidad	Índices	Complejidad
1	(0,0)(800,75)	22	(0,0)(800,85)	22
2	(0,75)(800,150)	22	(0,85)(800,170)	22
3	(0,150)(800,225)	11	(0,170)(800,255)	22
4	(0,225)(800,300)	22	(0,255)(800,300)	11
5	(0,300)(800,375)	22	(0,300)(800,345)	11
6	(0,375)(800,450)	11	(0,345)(800,515)	22
7	(0,450)(800,525)	22	(0,430)(800,515)	22
8	(0,525)(800,600)	22	(0,515)(800,600)	22

Figura 5.34: Propuesta de distribución de carga no constante para los workers.

este caso existe mayor probabilidad der intersecciones en el centro de la imagen, esto es debido a que en general la imagen tiene una distribución homogenea y los objetos que se encuentran en las orillas tienen menos probabilides de tener intersecciones.

El primer análisis que se hace con las imágenes es determinar su complejidad, la complejidad para el número de workers más eficiente (Experimento 2) es conocida así como la complejidad por franja de acuerdo a la descripción de la escena. El siguiente experimento muestra el orden de los datos y debido a que en imagenes de igual complejidad se ha comprobado que el tiempo en el que los workers terminan es casi constante se espera obtener imagenes parecidas.

Una vez que se ejecuta el experimento 3 el cual realiza 50 ejecuciones con 8 workers y con la distribución presentada anteriormente se obtiene la siguiente tabla en el conteo de posiciones de tiempos de los workers.

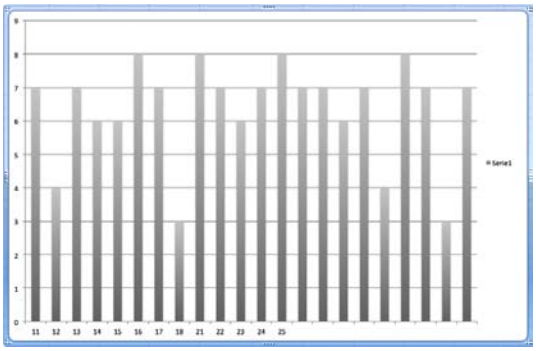

Figura 5.35: Orden de datos para propuesta de distribución.

Como se puede ver en la tabla de resultados, sucede que al igual que en la imagen que metódicamente se buscó tuviera la misma complejidad, la dsitribución de sus resultados es uniforme y respecto a los resultados que arrojó el orden de los datos con la previa solución resulta menos distribuida. Esto nos

indica que los workers terminan casi al mismo tiempo llevando una variedad probablemente solo por el acceso a memoria. El objetivo es lograr que los workers trabajen con la misma carga de trabajo pues el algoritmo será tan rápido cómo su worker más lento.

Capítulo 6

Conclusiones

En este capítulo se presentan las conclusiones respecto a los experimentos realizados en el capítulo anterior y se compara con los objetivos e hipótesis presentados en el capítulo uno. Se explica ampliamente la interpretación de los datos, tablas y gráficas presentadas. Se presentan las conclusiones individuales de cada experimento. También se presentan las conlusiones generales del motivo por el cuál se realizaron cada uno de los experimentos de esa forma para conducirnos a probar la hipótesis inicial. Una sección presentac las contribuciones obtenidas en el trabajo y finalmente el trabajo a futuro, posibles líneas de investigación con las cuales se puede mejorar la propuesta realizada hasta el momento e incrementar su campo.

6.1. Análisis de los datos

Una vez realizados cuatro tipos diferentes de experimentos, en un principio generales y por tal motivo una gran cantidad de ellos (aproximadamente 240 ejecuciones con más de 32000 datos) y conforme se avanza en la investigación más específicos y orientados a la hipotesis a comprobar, se llevan a acabo conclusiones respecto a ellos de forma particular, para con ello tener una recepción de la hipótesis planteada en un inicio al dar las conclusiones generales.

6.1.1. Complejidad de la imagen

En la realización de este experimento se busca una forma en la cuál se puede calcular la complejidad de la imagen. Esta es una práctica fundamental debido a que se debe contemplar una métrica para considerar la distribución del trabajo entre los workers.

Existen varias formas de realizar este cálculo, la mayor parte de ellas consiste en algoritmos complejos que obtienen los objetos de una imagen y los agrupan de acuerdo a distinta distribuciones. En este experimento se hace un algorimo sencillo pero representantivo para la medición de esta complejidad.

Un punto importante de este algoritmo es que se basa en la descripción de la imagen. Es decir a partir de un archivo .x3d; en el cuál se tiene una descripción de cada uno de los objetos que conforman la escena. El hecho de tomar la descripción del objeto de un archivo de texto hace que la medición de la complejidad no sea muy costosa en tiempo de ejecución pues no hace ningún tipo de intersección ni de cálculo de intersecciones como otros algorimos.

El programa para calcular la complejidad se basa en este algoritmo e incluso existe una versión la cuál hace este cálculo de la complejidad pero esta vez por franja. Los resultados obtenidos sirven tanto para la elección de la escena cómo para trabajar con imagenes controladas con la misma complejidad. El procedimiento es rápido y se puede expandir al cálculo de otras propiedades de los objetos que se pueden explotar en trabajo futuro.

6.1.2. Workers eficientes (Workers vs comunicaciones)

El objetivo principal de este experimento es el de encontrar el beneficio que ofrece el número de workers contra el número de comunicaciones que estos generan al Manager. En este experimento se encuentra cuatro casos.

El primer experimento, consiste en una esfera de complejidad 7. Una vez que se realizan 10 ejecuciones para las 24 opciones posibles, se obtiene que con 5 workers es el mejor caso, esto hablando en términos del mínimo, máximo y tiempo promedio en el cuál los workers terminan su ejecución.

En este tipo de pruebas de los datos más relevantes, es el tiempo máximo en el que un worker termina su ejecución, esto es debido a que el algoritmo será tan rápido como su worker más lento. Entonces es importante tener el conocimiento

de que tiempo es el máximo sin embargo el mínimo y promedio conservan la misma tendencia que el tiempo máximo. En la gráfica respectiva se puede observar que para más de 10 workers el tiempo en el cuál terminan aumenta significativamente lo cuál deja de hacer más eficiente la versión paralela que la secuencial.

En el segundo experimento se trabaja con una imagen de complejidad 11. En este caso se obtiene que con 4 workers se tiene el mejor resultado principalmente para el tiempo máximo y balance de carga. Una caracterísa importante que se observa desde este experimento es que el número de workers eficiente no es el mismo que en el primer experimento, y otra característica es que los workers terminan con mucho tiempo de diferencia aún para el worker más eficiente. Para workers mayores a 30 las comunicaciones toman tiempo en realizarse, lo que va haciendo más lento el algoritmo paralelo que el secuencial. Los peores tiempos registrados son para 200, 300 y 600 workers, lo cuál indica que en estos puntos las comunicaciones causan los peores resultados.

En el tercer experimento se toma una imagen más compleja de 124. Es importante el hecho de probar con imágenes que sean mucho más complejas puesto que el tiempo de procesamiento aumenta y las comunicaciones deben ser un poco menos representativas comparadas con el procesamiento de la imagen. Debido a las características de complejidad de la imagen y la distribución de los objetos en la escena, el número ideal de workers para realizar la ejecución del algoritmo es 8. De este experimento se puede extraer una característica más, en la representatividad de los workers contra comunicaciones influye la complejidad de la imagen. Entre más compleja sea la imagen, mayor será el número de workers que se utilicen para la ejecución más eficiente del algoritmo.

Finalmente se realiza un experimento que esta más delimitado, el experimento consiste en una imagen de la misma complejidad, hablando de luces distancia y objetos cercanos. En este experimento se confirma que la distribución de la carga esta ligado al número de workers requeridos para hacer que el algoritmo se ejecute eficientemente. La imagen consiste en un arreglo de esferas 7 esferas que conservan la misma distancia entre ellas y que tienen una fuente de luz frente a ellas. En el cam de vista solo se observan 5 esferas y las franjas tienen la misma complejidad. En un principio la hipótesis afirmaba que 5 workers sería el resultado ideal. Al ejecutar el algorimo el resultado fue 4 workers. Esto es debido a que la imagen al momento de dividirse en cuatro partes tenía la

misma complejidad por franja. Así mismo ocurren picos con menor tiempo en los múltiplos de 4 y 5. Este experimento lleva a responder porque se encuentran varios picos en la imagen, porque es que el tiempo no es completamente lineal y tiene variaciones drásticas. Así mismo la forma en la que la distribución afecta la ejecución eficiente del algoritmo.

6.1.3. Orden de datos

Una vez que se encuentra que los workers no terminan al mismo tiempo la ejecución de su porción de trabajo, una pregunta que surge es si existe constancia entre el tiempo que van regresando la información al manager. Para probar la constancia de término de los workers, se hace un análisis del orden en el que termina. El experimento consta de 50 ejecuciones con 5 workers en los cuales se presenta la frecuencia de repeticiones de cada posición que toma cada worker. Se repite el experimento con las mismas imágenes que se hizo en la sección pasada.

Para el experimento uno, el orden de los datos es casi constante, de la 50 ejecuciones que se realizaron 45 de ellas tuvieron el mismo orden, solo en 5 ocaciones, dos workers intercambiaron posiciones. Esto refleja nuevamente que el trabajo (complejidad) influye en el tiempo de término del programa.

El experimento dos fue muy similar al primero, la diferencia de complejidad entre la primer imagen y la segunda tiene un valor de 5, con lo cuál no se podrían esperar grandes variaciones al respecto. En este caso la constancia fue en una unidad mayor que en el experimento anterior.

El tercer experimento en dónde se tiene una imagen de mucho mayor complejidad 124, se puede ver un gran cambio. Todos los workers se encuentran en todas las posiciones posibles. No es constante el orden de los datos hay algunos workers que predominan en ciertas posiciones pero al menos se encuentra un poco más distribuido por número de objetos que le corresponde a cada franja. Buscando un balance de carga entre el worker con más posiciones y con menos, la diferencia que se obtiene es de 33.

En el cuarto experimento para mostrar que la complejidad de la imagen impacta directamente sobre el orden de los datos, se manda la misma franja de una imagen sobre los diferentes workers, de esta manera se asegura que la complejidad es la misma. Los resultados que se obtienen son más constantes. De

igual forma cada worker toma al menos 4 veces una posición. Y el balance de carga que se identifica entre el worker con más número de posiciones respecto con el menor número de posiciones es de 14 unidades. Estas diferencias que se observan puede deberse al acceso a memoria el cuál no es constante.

En el último experimento se genera una imagen buscando que para cada franja tenga la misma complejidad, tomando en cuenta distancias, luces y objetos cercanos, se vuelven a realizar 50 ejecuciones para comprobar el orden de los datos y se encuentra una mayor constancia en la gráfica. Todos los workers se encuentran en la misma posición al menos en 6 ocaciones y esta vez el balance de carga entre el worker con mayor y menor número de posiciones disminuye, en esta ocación el balance es de 11.

Con este experimento se puede ver que la complejidad de la imagen impacta directamente sobre las posiciones en las cuales van a terminar los workers, es decir si a un worker le corresponde una franja que tenga mayor complejidad, quiere decir que este worker ocupara un tiempo constante lo cuál lo pondrá la mayor parte del tiempo en una posición puesto que el término de este worker estará directamente afectado por el tiempo que le tome hacer el procesamiento de la imagen.

6.1.4. Distribución entre workers

Una vez que se observa que en las imágenes que se probaron la complejidad afecta de manera proporcional, lo que se busca es mejorar la forma en la cuál se va a distribuir la escena entre los workers. Para tal tarea se realiza un programa el cuál mida la complejidad de la imagen por franja. Una vez que se mide la complejidad por franja para la escena en la cuál hay 124 esferas, se propone una mejor distribución de la escena para poder repartir de manera más equitativa el trabajo a los workeres y de esta manera, al tener un worker menos lento se tiene un Ray Tracing más rápido. Con esta mejor distribución se realizan 10 ejecuciones al programa de Ray Tracing y los tiempos obtenidos respecto a la ejecución con menor tiempo son mejores en un 10

6.2. Conclusiones generales

Se realiza un anális respecto a la paralelización de Ray Tracing. La paralelización se realiza con el patrón manager workers. Una vez que se realiza la paralelización, se espera que todos los workers terminen en tiempos similares sin embargo el tiempo en el que terminan es muy diferente y más en imágenes que tienen una complejidad pequeña. Por tal motivo se hacen pruebas para relacionar que la distribución de la imagen basado en la complejidad de la misma es proporcional al tiempo que toman los workers en terminar. Una vez que se hace este análisis se encuentra una forma de medir la complejida por franjas a través de la descripción de la escena, y entonces se propone una mejor distribución del trabajo en este caso una distribución no uniforme.

Finalmente con esta distribución de la imagen lo que se consigue es obtener un mejor resultado de esta haciendo un análisis previo a la descripción de la escena.

6.3. Contribuciones

- Un programa de Ray Tracing paralelo basado en el patrón manager workers
- Análisis y metodología para paralelizar el algoritmo de Ray Tracing
- Propuesta de una distribución de la escena basada en la complejidad de la franja que le corresponde a cada worker

6.4. Trabajo Futuro

- Mejorar el sistema para medir la complejidad de la imagen, principalmente tomando en cuenta tamańo de los objetos, objetos vecinos, propiedades reflexivas y refractivas.
- Hacer Ray Tracing en un sistema distribuido
- Realizar la granularidad de la imagen por pixel

Bibliografía

[1] Jorge Luis Ortega Arjona. *Architectural Patterns for Parallel Programming, Models for Performance Estimation*, tomo Department of Computer Science University College London. 2006.

[2] Kadi Bouatouch y Thierry Priol Daniel Menard. Parallel radiosity using a shared virtual memory. *IRISA. Campus de Beaulieu. 35042 Rennes Cedex*, 1993.

[3] William A. Fetter. *Computer Graphics*. 1966.

[4] Stuart Green. *Parallel Processing for Computer Graphics*. The MIT Press, 1991.

[5] Jos Galavz Casas Jorge Luis Ortega Arjona. *Programacin Concurrente*. MCC, 1996.

[6] G. Scott Owen. *Pixel Cinematography: A Lighting Approach for Computer Graphics*, 1998.

[7] Pancake y Bergmark. 1990.

[8] Samuel Milton Steve Glazer, Sara Jackson. *Parallel Ray Tracing*, 2001.

[9] Naty Hoffman Tomas Akenine-Mller, Eric Haines. *Real-Time Rendering*. A K Peters, Ltd, 2008.

[10] Andries van Dam. *Introduction to Computer Graphics*, 2001.