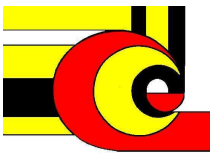
INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación



Algoritmos óptimos para la generación de imágenes fotorrealistas.

T E S I S

Que para obtener el grado de:
Doctor en Ciencias de la Computación presenta el

M. en C. Mauricio Olguín Carbajal

Director: Dr. Ricardo Barrón Fernández
Director: Dr. José Luis Oropeza Rodríguez

México D.F., julio de 2011

Agradecimientos técnicos

Al Instituto Politécnico Nacional, por toda la educación y formación que he recibido tanto en sus instalaciones como fuera de ellas, en general es la mejor educación que se puede recibir.

Al Centro de Investigación en Computación (CIC), por darme la oportunidad de estudiar en un Programa Doctoral de reconocido prestigio. Por sus instalaciones, sus profesores, equipo y todo en general.

Al CONACyT por su apoyo a lo largo de mis estudios doctorales.

Al Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC), por su apoyo, comprensión y facilidades otorgadas para realizar mis estudios doctorales.

A mi director de tesis el Dr. Ricardo Barrón Fernández, por su experta guía para el desarrollo de este trabajo, así como por su apoyo y sobre todo por su amistad.

A mi director de tesis el Dr. José Luis Oropeza Rodríguez, por su experiencia y sabia conducción. Asimismo por su gran apoyo y amistad.

A mis sinodales, el Dr. Isaac Juan Rudomin, el Dr. Sergio Suarez y el Dr. Serguei Pavlovich Levashkine, por sus muy acertados consejos y por su apoyo para la adecuada culminación de este trabajo.

A la Dra. Nareli Cruz por su clases, sus sabios consejos y su experta guía.

A todo el personal del CIC, comenzando por mis profesores, y a todo el personal en general por su apoyo.

Agradecimientos personales

Ante todo deseo agradecer a mis padres Lucia Carbajal Vilchis y Juan Olguín Trejo por todo, en particular por siempre enseñarme que la educación y el estudio son la base del desarrollo personal.

A mi familia, mi esposa Claudia por su incondicional apoyo, amor, compañía, paciencia, y su constante enseñanza.

A mis hijas por su sola presencia.

A mis hermanos, abuelos, tíos y primos que siempre me enseñaron y ayudaron a aprender.

A todos mis amigos, pasados y presentes por su amistad.

A todos mis profesores, de todos he aprendido mucho.

Resumen.

La presente tesis aborda la propuesta y desarrollo de una metodología de extracción de parámetros obtenidos a partir de imágenes de objetos reales con una cámara digital para, posteriormente, usar dichos parámetros como elementos de un modelo de iluminación en la renderización de imágenes generadas por computadora, que se aproximen lo más posible a imágenes de objetos reales.

A diferencia de las metodologías anteriores que usan un ciclo abierto para la estimación de los parámetros, la metodología propuesta usa un ciclo cerrado para la estimación de dichos parámetros de renderización por medio del uso de una heurística bioinspirada no supervisada.

El presente estudio también aporta una función de la medida de similitud entre dos imágenes, así como un conjunto de puntos de referencia para aproximar a una imagen con otra, estos puntos son parte de los parámetros ópticos obtenidos por la metodología.

Índice general

Índice de figuras

Índice de tablas

Resumen

La presente tesis aborda la propuesta y desarrollo de una metodología de extracción de parámetros obtenidos a partir de imágenes de objetos reales con una cámara digital para, posteriormente, usar dichos parámetros como elementos de un modelo de iluminación en la renderización de imágenes generadas por computadora, que se aproximen lo más posible a imágenes de objetos reales.

Se propone una metodología para que, a partir de una imagen tomada de un objeto real; se obtengan los parámetros ópticos que produzcan una imagen con un algoritmo de renderización, el cual tomará parámetros de un algoritmo biosinspirado y gradualmente generará imágenes que se aproximen cada vez más a la imagen del objeto real. A diferencia de las metodologías anteriores que usan un ciclo abierto para la estimación de los parámetros, la metodología propuesta usa un ciclo cerrado para la estimación de dichos parámetros de renderización por medio del uso de una heurística bioinspirada no supervisada.

En el trabajo se observa que el nivel de exactitud de los parámetros obtenidos es directamente dependiente de la exactitud del modelo de iluminación usado, con un modelo sencillo se obtienen parámetros inexactos, con un mejor modelo se obtienen parámetros con una mejor aproximación. El presente estudio también aporta una función de la medida de similitud entre dos imágenes, así como un conjunto de puntos de referencia para aproximar a una imagen con otra, estos puntos son parte de los parámetros ópticos obtenidos por la metodología.

Finalmente como parte del uso de una arquitectura paralela para mejorar el desempeño en velocidad, se demuestra que la implantación de heurísticas bioinspirada es paralelizable, lo que mejora respecto al nivel de granuralización de la implantación usada y a que tanto del código a ejecutar se programa en el GPU.

Abstract

This work is about a design and development for a parameter estimation methodology from real objects image obtained with a digital camera for, afterward, use these obtained parameters as input data for an illumination model in computer generated graphics rendered images, as close as posible to the real objects image.

We propose an optical parameters estimation metodology from data obtained of an images of a real object, by using these parameters in a CG rendering software we can render an image close enought to the original to be indistinguible. The proposal will take test parameters generated in a bioinspirated algorithm and then render images more and more close to the real object image.

Apart from other methodologies that use an open loop for parameter estimation, the proposed methodology uses a close loop for the estimation of these illimination model rendering parameters, by using a non-supervised bioinspirated heuristic.

In this work we are able to see that the precision level of the obtained parameters is straigthforward dependent of the used illumination model accuracy. The better the illumination model, the better the acuracy for the obtained parameters.

This work also add a function capable to compare two images and assign a difference percentage between both images. Present work adds too a set of image reference points, used to bring near generated images to objetive image, aquired one, this set of point are part of metodology obtained optical parameters.

Finally to improve the performance and speed, we show that a bioinspirated heuristic implementation is parallelizable, to gets better performance at granuralization level respect how much of executing code is running on the Graphics Proccesor Unit.

Capítulo 1

Introducción

1.1. Motivación

La búsqueda de imágenes fotorrealistas, comienza desde los primeros trabajos en 1945 de Jay Forrester del MIT en ASCA (Navy's Airplane Stability And Control Analyzer), creando el sistema de despliegue de texto e imágenes en un osciloscopio, el problema de la graficación ha sido un reto que gradualmente ha ido aumentando, actualmente las gráficas por computadora se usan de forma tan cotidiana que ya son parte de la vida diaria, y han logrado un realismo tal que en la mayoría de las ocasiones no es posible percatarse de que lo que se ve en la televisión o en el cine son gráficas generadas por computadora. Por ejemplo, en la película el Harry Potter 7 parte 2 se usaron más efectos por computadora que en ninguna otra película anterior (julio de 2011). A las gráficas generadas a partir de los datos de un escenario virtual se les denomina gráficas renderizadas.

El termino render se puede traducir como "interpretar", de tal forma que cuando se tienen datos de computadora que representan un escenario, tales datos deben ser interpretados para generar una imagen en dos dimensiones que sea susceptible de ser presentada en un dispositivo de despliegue tal como un monitor de computadora o un cañón proyector.

En el presente trabajo se propone una metodología para que, a partir de una imagen tomada de un objeto real; se obtengan los parámetros ópticos que produzcan una imagen con un algoritmo de renderización, el cual tomará parámetros de un algoritmo biosinspirado y gradualmente generará imágenes que se aproximen cada vez más a la imagen del objeto real. La metodología para estimar los parámetros de uno o varios objetos de un escenario real usa un dispositivo sencillo de adquisición de imágenes y técnicas de inteligencia artificial como son los algo-

ritmos heurísticos poblacionales tales como los algoritmos genéticos, la evolución diferencial y la optimización por cúmulo de partículas, entre otros. Usando esta metodología, es posible obtener los parámetros de un escenario real y aplicarlos en un escenario virtual para renderizar imágenes con una apariencia fotorrealista sin consumir tanto tiempo de cómputo en el cálculo del mapeo de texturas o en un cálculo en tiempo real de texturas simuladas.

El diseñador de los modelos tridimensionales tiene que ajustar los parámetros del modelo de iluminación con el que va a trabajar de forma que las imágenes que se generen a partir de estos datos se parezcan al mundo real desde su punto de vista. Los parámetros a extraer en el presente trabajo son: el color difuso de los objetos, su índice de reflexión, su brillantez, su rugosidad, y la posición de las fuentes de iluminación en el escenario, así como su intensidad.

Nuestra metodología, a diferencia de las existentes, propone el uso de una fase de retroalimentación en la extracción de parámetros para el renderizado, en el que se compara una imagen objetivo con una imagen generada por la metodología y los valores propuestos por una heurística bioinspirada. Para lograr lo anterior, se propone una función de evaluación que compara el grado de similitud entre dos imágenes y lo asigna como una aptitud a un individuo de la heurística. Los parámetros de renderizado de un escenario se codificaron como parte de un individuo de la heurística y los mejores individuos obtenidos generaron las imágenes más aproximadas a la imagen de referencia.

1.1.1. Relevancia

Desde el punto de vista de las aplicaciones, la generación de imágenes fotorrealistas tiene un alto impacto económico a nivel mundial. Como ya se mencionó anteriormente, la industria de la cinematografía se ve en gran medida beneficiada de las imágenes generadas por computadora, pero no es la única, también industrias multimillonarias como la de los juegos de computadoras, las de producción y postproducción de video, las de imágenes médicas, las de televisión (por ejemplo, las productoras de programas infantiles), la investigación en física atómica, las de astronomía, química, entre otras.

La ventaja que presenta esta metodología, respecto a la síntesis puramente modelada, radica en que el diseñador obtiene parámetros de renderizado que empatan con objetos, colores y texturas reales que él desea obtener, por ejemplo: en el diseño de elementos de escenarios, donde se van a mezclar objetos renderizados con objetos reales, el adecuado empalme de los parámetros de renderizado con la imagen real debe ser de una diferencia mínima. En películas como el hombre araña o los hombres X, se conjugan escenarios reales, como las calles, casas, interiores,

autos e incluso algunos actores, con personajes renderizados y se requiere que su respuesta a las condiciones de iluminación, sean idénticas, objetivo que se puede lograr de manera más rápida usando esta metodología, que una aproximación de síntesis pura o una metodología semi-automática.

1.2. Justificación

Actualmente la mayoría de los métodos empleados para la obtención de parámetros, no usan las condiciones actuales del escenario a renderizar ya que se basan en el criterio y la experiencia del diseñador, lo cual tiene una gran parte subjetiva. La solución propuesta es eliminar en la medida de lo posible lo subjetivo, por medio del uso de Inteligencia Artificial y la metodología propuesta, y volverlo objetivo.

1.3. Hipótesis

Es posible realizar una extracción de parámetros ópticos que definen a un objeto por medio del uso de un dispositivo sencillo de captura de imágenes y un algoritmo bioinspirado que buscará los parámetros a encontrar; generando una imagen a partir de un escenario virtual que contenga los mismos elementos que la imagen original y realizando una comparación de la imagen generada contra la imagen obtenida e intentando minimizar la diferencia entre ambas, modificando únicamente los parámetros del modelo de iluminación a usar. Esto se realiza mediante un dispositivo sencillo para capturar una imagen de un objeto con parámetros desconocidos de color, reflexión y absorción de la luz.

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar una metodología de extracción de parámetros de iluminación para renderizado, por medio del uso de una imagen de un escenario real como referencia.

1.4.2. Objetivos específicos

1. Proponer una función que evalúe la diferencia entre dos imágenes de la misma resolución.

2. Desarrollar un algoritmo heurístico poblacional bioinspirado para obtener los parámetros del modelo de iluminación para algoritmos de renderizado.

3. Integrar la función de evaluación de dos imágenes al algoritmo bioinspirado de renderizado de imágenes.

4. Programar el algoritmo en una arquitectura paralela para mejorar el desempeño en tiempo.

1.5. Alcances del trabajo

1. El desarrollo de una nueva metodología para la extracción de parámetros en un ciclo cerrado, a diferencia de los desarrollos actuales los cuales usan ciclo abierto.

2. El desarrollo de una versión paralela del algoritmo obtenido, para su ejecución en una arquitectura de Unidad de Procesamiento Gráfico (Graphics procesing Unit, GPU).

1.6. Contribuciones del trabajo

1. El aporte principal de este trabajo fué el desarrollo de una metodología de extracción de parámetros de renderizado de escenarios en tres dimensiones, de ciclo cerrado, a diferencia de las actuales que son de ciclo abierto.

2. Se desarrolló un algoritmo heurístico de optimización de parámetros físico ópticos de renderizado que usa un algoritmo de traza de rayos para obtener la aptitud de cada individuo. Dicho algoritmo los hemos denominado "Rendering in the loop".

3. Se desarrolló una de las primeras implantaciones del algoritmo de evolución diferencial en una arquitectura paralela de GPU.

4. Se demostró que la metodología propuesta obtiene mejores resultados que una persona y en un tiempo significativamente menor, en un orden de magnitud.

5. Se desarrolló un algoritmo para el cálculo de puntos óptimos para la extracción de parámetros.

6. Se demostró que la metodología obtiene mejores resultados que la mayoría de las metodologías comparables, y que rivaliza con el mejor método comparado.

7. Se demostró que una programación paralela basada únicamente en las características multihilos de un GPU, resulta en una aceleración proporcional al número de núcleos de un GPU, obteniendo el poder computacional de un cluster en una computadora personal convencional.

8. Se demostró que el desempeño total de un GPU mejora conforme aumenta la cantidad de tareas sencillas distribuidas en los hilos del mismo.

1.7. Resultados obtenidos

1. Se desarrolló un algoritmo heurístico poblacional bioinspirado para la extracción de los parámetros de iluminación de un ambiente virtual.

2. Un algoritmo que pruebe el nivel de aproximación entre las imágenes obtenidas por el sistema de adquisición y el desarrollado en la presente tesis.

3. Se desarrolló un sistema sencillo de adquisición de imágenes de referencia.

4. Se apoyó en la generación de una base de conocimientos en el área de las gráficas por computadoras para el Centro de Investigación en Computación en particular y para el Instituto Politécnico Nacional en general.

5. Se publicó a nivel internacional resultados obtenidos, tanto en dos revistas de prestigio así como en congresos internacionales.

1.8. Organización del trabajo

La organización del presente documento es la siguiente:

En el capítulo 2 se presenta una investigación del estado del arte de la graficación por computadora y los desarrollos actuales para la generación de imágenes fotorrealistas. Por otra parte se presenta un estado actual de las investigaciones relacionadas con la extracción de parámetros físico ópticos para la síntesis de imágenes por computadora.

El capítulo 3 presenta los fundamentos teóricos necesarios de la graficación por computadora, específicamente el modelo de iluminación y sus parámetros de trabajo. Se explica la percepción del color y los diferentes modelos de color, así como la corrección gamma. Finalmente, se explican los fundamentos de las heurísticas bioinspiradas a usar, en particular el algoritmo de optimización por cúmulo de partículas (PSO) y el algoritmo de evolución diferencial (DE).

En el capítulo 4 se realiza una descripción de la propuesta para la extracción de parámetros físico ópticos usando un dispositivo sencillo de adquisición de imágenes. Pruebas de la función objetivo y la propuesta del algoritmo extracción de parámetros basado en una heurística poblacional bioinspirada.

En el capítulo 5 se describe el proceso de la implantación de la metodología de extracción de parámetros en la arquitectura del GPU. Primero se realiza una

implantación de un algoritmo PSO y se evalúa su desempeño. Posteriormente se hace un comparativo de la implantación de un algoritmo PSO y de un algoritmo de Evolución Diferencial y se reporta su comportamiento en el GPU. Finalmente, con base en los resultados de los experimentos anteriores se implanta la metodología propuesta en el GPU.

El capítulo 6 se presentan pruebas y resultados obtenidos con diferentes imágenes objetivo, se destaca la obtención de la posición de la fuente de iluminación así como su intensidad. Asimismo se obtiene un color aproximado de los objetos en el escenario y una huella de la imagen objetivo representada por un conjunto de puntos de interés.

El capítulo 7 Se presenta el conjunto de conclusiones, trabajo futuro y productos de la investigación.

Capítulo 2

Trabajos relacionados

2.1. Antecedentes

Desde el desarrollo de la primera técnica para la interpretación de datos de escenarios virtuales [3], que consumía mucho poder de cómputo, se han desarrollado técnicas que gradualmente fueron mejorando la apariencia de la imagen final, primero haciendo una interpolación bilineal de intensidades para tener curvas más suaves [37], después realizando mapeos de textura [12], posteriormente agregándole brillos especulares a los objetos [3] hasta que finalmente se llegan a técnicas de generación fotorrealistas que brindan imágenes que usan un modelo que simula el comportamiento de los rayos de luz [92] el cual obtiene algunas de las imágenes más impresionantes generadas por computadora [69] y [11]. Por desgracia, estas técnicas consumen una gran cantidad de poder de cómputo, por lo que la comunidad de síntesis de imágenes por computadora está en constante búsqueda de nuevas técnicas que permitan obtener imágenes fotorrealistas a un menor costo computacional. Muchas de las propuestas para resolver la generación de gráficas por computadora van desde el tipo de estructura para el almacenamiento de la información [29] [81], pasando por la arquitectura empleada para el mejor desempeño de los algoritmos de renderizado [54], y por la subdivisión espacial del entorno virtual [51] [41], y finalizando hasta la forma de la distribución de los rayos de luz [15]. Algunas de estas aproximaciones usan novedosas técnicas de transformaciones geométricas [80] como parte de su esquema para lograr una mejora en el desempeño. Existen también otras soluciones en las cuales técnicas conocidas como la traza de rayos son modificadas para usar algoritmos evolutivos con la finalidad de obtener mejores imágenes en menor tiempo con el mismo poder de cómputo [90] y [69].

Las mejoras para la generación de gráficas fotorrealistas puede ser en cualquiera

de las tres grandes etapas del algoritmo de renderización: aplicación, geométrica y rasterizado, ver apéndice B.
En la etapa de aplicación se realiza optimización usando por ejemplo:

1. Nivel de detalle
2. Objetos precalculados
3. Detección de colisiones
4. Secuencias precalculadas

Los algoritmos para el manejo del nivel de detalle, así como los objetos precalculados, la detección de colisiones y las secuencias precalculadas son operaciones que tienen la finalidad de evitar cálculos innecesarios de objetos o escenarios en los cuales se pueden encontrar redundancias y tiempo de procesamiento perdido que no van a tener relevancia en la imagen final; por ejemplo, si se tienen objetos que no cambian de posición, orientación o tamaño, dichos objetos pueden ser precalculados de forma que no impacte en el tiempo de generación de una imagen donde existen otros objetos que si tienen variantes. En el caso del nivel de detalle, se hace que los objetos a grandes distancias tengan poco detalle (pocos polígonos) y a distancias cortas muy buen detalle (muchos polígonos por objeto), lo que aumenta considerablemente el desempeño del algoritmo de cálculo de la imagen, ya que en objetos lejanos con cientos o miles de polígonos sólo se calculan unas pocas decenas de polígonos y no existe impacto en la imagen final generada.

Por parte de la etapa geométrica se realiza optimización usando:

1. División del espacio
2. Árboles binarios [29]
3. Árboles octales [36]
4. Subdivisión adaptiva [81]
5. Álgebras geométricas [69]
6. Cuaterniones [47] [80]
7. Uso de Voxels [51]
8. Intersecciones triángulares [41]
9. Intersecciones cuadrilaterales [41]
10. Intersecciones jerárquicas [45]

En la parte de procesamiento geométrico es posible apreciar que a nivel mundial se realiza un gran esfuerzo por reducir tanto el número de polígonos como las estructuras necesarias para la descripción de los objetos y la relación que existe entre ellos y el escenario donde todos se encuentran. Por ejemplo Han-Wei propone una nueva estructura de almacenamiento de información [40], pero aún es necesaria una traducción, debido a que al tratar los polígonos se debe realizar una descripción de ellos en una forma que la computadora sea capaz de usar. Esta descripción es comúnmente a través de vectores y su manejo es típicamente por medio del uso de matrices, pero no es ésta la única manera, existen otras formas de lograr una reducción por medio del uso de metodologías diferentes a las comunmente usadas para el manejo de los descriptores básicos de los objetos y el universo donde ellos se desenvuelven.
En el campo de las transformaciones geométricas también se han usado las soluciones heurísticas como parte de nuevas aproximaciones para resolver problemas tales como la programación geométrica, específicamente la solución del sistema inmune [55]. También es posible una reducción si se usa el álgebra de Clifford también conocida Algebras Geométricas. Ya se han realizado desarrollos usando los cuaterniones para resolver las rotaciones de una manera diferente al uso de las matrices [47] y [80], estos desarrollos han resuelto las rotaciones en objetos y escenarios tridimensionales de una manera más eficiente que la solución tradicional con matrices aunque sólo acelera la transformación de las rotaciones. Los cuaterniones se pueden considerar una pequeña parte de las Álgebras Geométricas (AG), por lo cual es probable que usando las AG se puedan acelerar otras transformaciones geométricas como son la traslación, el escalamiento, y la deformación. De hecho, ya se han desarrollado algunas soluciones con AG para la renderización, tal es el caso de un sistema de renderización por traza de rayos que se puede tener al final del libro de Daniel Fontijne [59].

En la etapa de rasterizado se realiza optimización usando:

1. Jerarquías Simples [25]
2. Metajerarquías [4]
3. Distribución guiada de rayos [15]
4. Traza de rayos paralelos [32]
5. Clasificación de rayos [49]
6. Conos de rayos [2]
7. Traza de rayos por hipercubo [54]
8. Evolutiva de rayos [69]

A nivel de rasterizado también es posible usar una aproximación diferente para lograr combinar las dos grandes áreas de la resolución de visión y de iluminación, la traza de rayos y la radiosidad. Este concepto ya fue manejado por James T. Kajiya en su propuesta de la ecuación de renderización [56], en la que propone que ambos métodos mencionados anteriormente son, de hecho, una descripción diferente de la misma ecuación y propone la unión de ambos métodos a través de su ecuación ya sea por métodos convencionales o por medio de heurísticas, es aquí donde pueden entrar los algoritmos evolutivos para lograr encontrar la mejor solución práctica a la ecuación de renderizado. Y es que en la renderización ya se han usado heurísticas para solucionar algunos problemas específicos, por ejemplo para la distribución de rayos se usan mutaciones logrando una mejora notable en la imágen en el mismo tiempo [90] y [69]. Pero un algoritmo de generación de imágenes tiene una gran cantidad de parámetros a considerar y asimismo se han propuesto soluciones a cada uno de ellos en particular, tales como: la difracción de la luz en medios turbios [10], el transporte superficial de la luz para obtener una adecuada representación de subsuperficies [44], un manejo adecuado de la sombra y la penumbra [62], una adecuada iluminación global basada en la emisión de luz como una radiación [25], y la representación de texturas [12], entre muchos otros. Todos estos parámetros se van a solucionar de manera diferente dependiendo de la aproximación para la representación de la luz en el entorno de la computadora, las dos grandes tendencias para dicha representación son que la luz se comporte como una partícula representada por un rayo, es decir, una reflexión perfectamente especular (traza de rayos) y que la luz se comporte como una onda una reflexión perfectamente difusa (radiosidad), pero como menciona Kajiya [56] ambas soluciones son parte de una misma ecuación ya que en la realidad se tiene tanto una reflexión especular así como una reflexión difusa y es factible de realizar una solución general a la ecuación de renderizado, algunas propuestas son realizar de forma tradicional la solución combinando ambas técnicas como en el caso de la radiosidad con dos trayectorias y en el caso de la traza de rayos con un muestreo estocástico o con una generación de rayos hijos. Pero la solución de Kajiya propone el uso de un algoritmo muy parecido a la traza de rayos que genera reflexión difusa, reflexión especular y refracción, por medio del uso de un rayo que genera hijos de forma estocástica.

2.2. Revisión del estado del arte

Es posible apreciar que a nivel mundial se ha tenido un gran avance en el área de graficación por computadora, la optimización es parte fundamental de estos avances, ya que a medida que se tiene un mejor uso de los recursos, se pueden usar técnicas más complejas que aproximan las imágenes finales más a las reales. A pesar de esto todavía son posibles una gran cantidad de mejoras, por mencionar

algunas están:

1. Usar otras álgebras geométricas además de los cuaterniones.
2. Usar algoritmos evolutivos para: hibridación de algoritmos, finalización de rayos inútiles, distribución de los rayos, área de sombreados y selección de la estrategia de sombreado.
3. Selección del algoritmo de rasterizado: Traza de rayos, Radiosidad, Buffer Z, Buffer A, etc.
4. Estimación de los parámetros de renderizado.

De lo anterior se puede ver que las mejoras en el área de optimización de gráficos por computadora es una promesa de obtención de buenos resultados en la investigación de las ciencias computacionales.

2.3. Trabajo relacionado

Los esfuerzos se han enfocado en desarrollar algoritmos de renderizado eficientes y exactos basados en la estimación Montecarlo [69] y aproximaciones analíticas. Dichos algoritmos tienen parámetros que directa o indirectamente representan las propiedades físicas ópticas de los objetos que componen el escenario. Para lograr que la imagen generada tenga una apariencia realista es necesario que dichos parámetros sean exactos. A medida que han avanzado y mejorado los algoritmos de renderizado, el realismo de una imagen es altamente dependiente de la exactitud de los parámetros físico ópticos de los objetos. Ya que actualmente existen relativamente pocos trabajos para medir o estimar las propiedades físico ópticas de los objetos, relativo a la graficación por computadora, los parámetros normalmente son definidos por el diseñador del ambiente virtual, con toda la subjetividad que esto conlleva.

Los métodos existentes para la medición de los parámetros físico ópticos requieren del uso de equipo muy costoso por ejemplo el que se usa para medir las propiedades de dispersión para química coloidal [26], por medio del uso de un Goniometro ALV modelo SP-86, un fotomultiplicador QB EMI 9863, y un correlacionador-estructurador ALV 3000. Como fuente de iluminación usaron un Laser de Argon-ion operado a una longitud de onda de 514.5nm, el análisis de los datos obtenidos se proceso en varias computadoras personales (PC) con procesadores 80286 y 80386. El experimento obtuvo datos muy útiles para su investigación, sin embargo entrega pocos datos utilizables para la graficación por computadora. Más recientemente se han realizado esfuerzos para la medición y estimación de propiedades físicas ópticas de los objetos con la finalidad de usar

estos parámetros en la graficación por computadora en los modernos algoritmos de renderizado.

Existen trabajos destacables en la medición o estimación de los parámetros de los objetos virtuales. De acuerdo a Srinivasa G. Narasimhan et al. [82] se tienen dos métodos para la obtención de las propiedades ópticas de los objetos: la medición directa y la estimación indirecta. La estimación indirecta hace uso de aproximaciones analíticas [46], primero midiendo algunos parámetros de objetos de prueba y después proponiendo efectos de la función de distribución de dispersión de superficie que no es posible capturar.
El uso de soluciones numéricas para el transporte de luz [10], en el que pro-

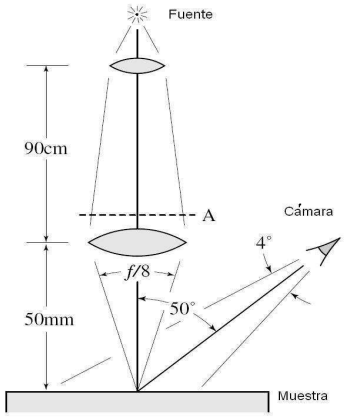


Figura 2.1: Un modelo práctico para el transporte de luz subsuperficial, Henrik Wann Jensen et al.

ponen una integración analítica de las ecuaciones de transporte de luz para la dispersión sencilla para una fuente de luz isotrópica en un medio homogéneo, para su procesamiento en tiempo real, obteniendo resultados impresionantes. Sin embargo, para usar este método es necesario tener tablas con valores de parámetros precapturados, lo que nos conduce, en primer lugar, a la obtención de dichos parámetros.

Para la medición directa se han usado algunos métodos, entre ellos la Goniofotometría la cual mide la función de fase para la dispersión en medios traslucidos [28]. En esta técnica se usa óptica microscópica convencional pero se reemplaza al sistema de iluminación al usar una hoja de luz Laser, lo cual proporciona una mejor medición del medio, ya que mejora la nítidez de la imagen al mismo tiempo que disminuye el nivel de ruido de fondo.

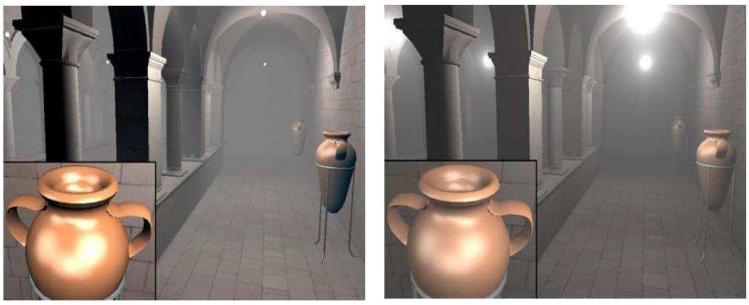


Figura 2.2: Un modelo practico analitico singular de dispersión para el renderizado en tiempo real, Bo Sun et al.

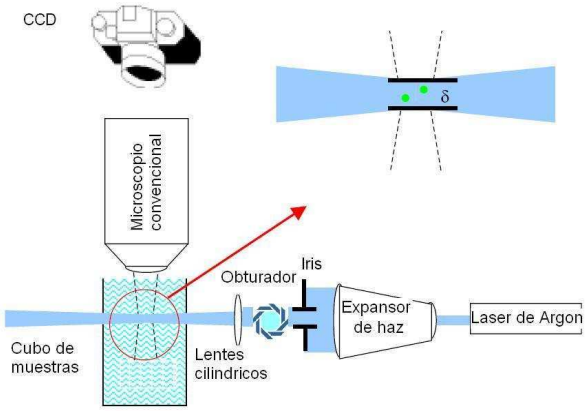


Figura 2.3: Un microscopio de hoja delgada de luz laser para oceanografía microbiana, Fuchs et al.

Una aproximación es la que realizan Fukawa et al. [30] en la cual usan un dispositivo de adquisición, basado en escaners laser de rango, para obtener el modelo 3D (tridimensional) de la textura de los objetos, la cual después aplicarán en sus modelos. Esta plataforma especializada para captura de parámetros, funciona en dos etapas: primero obtiene la información geométrica de los objetos y su conjunto indexado de texturas 4D o BTF (Funcion de Textura Bidireccional, bi-directional texture functions), y después comprime la base de datos de la textura usando un producto tensor de expansión. Esta información le permite a los diseñadores renderizar los objetos desde puntos de vista arbitrarios, con iluminación y deformaciones también arbitrarias.

De forma similar Gero Muller et. al [35] usan un sistema de adquisición de parámetros el cual está formado por una semiesfera de cámaras y luces fijas que a diferencia de otros diseños usa un paralelismo masivo y no tiene partes móviles, esta aproximación obtiene la textura tridimensional de los objetos así como su

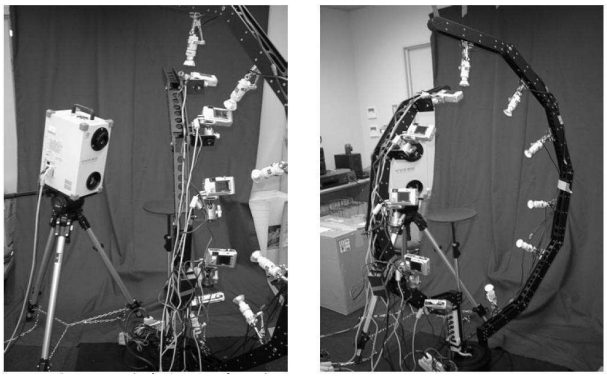


Figura 2.4: Modelado de objetos basado en apariencia usando una base de datos de texturas: captura, compresión y renderizado, Furukawa et al.

color e índice de reflexión, sin embargo el equipo y el poder de cálculo necesarios son costosos.

Wojciech Matusik et al. [93] usan un equipo con 6 cámaras y un arreglo de luces como dispositivo de captura, colocando los objetos en una mesa giratoria y hacen uso de técnicas multifondo "mate" para la adquisición del canal alfa y el ambiente mate para múltiples puntos de vista, con lo cual reconstruyen la apariencia tridimensional del objeto con los índices de color, reflexión y difracción que usaran en sus modelos, sin embargo el equipo y el procesamiento sigue siendo costoso.

G. Müller et. al [31] diseñan un dispositivo con una sola cámara y una fuente de luz, pero la fuente de luz se queda fija, mientras que la cámara se desplaza en un semicírculo para obtener los parámetros de reflexión y color de un objeto. El sistema requiere un ambiente medido y de control complejo ya que adquiere una función bidireccional de textura (BTF) para la adecuada representación de la textura del objeto en el modelo tridimensional, esta técnica brinda unos resultados impresionantes, aunque sigue siendo un sistema costoso por el equipo, el control y el cálculo asociado. Asimismo la gran cantidad de datos generados por un sistema como éste requiere que los datos adquiridos sean comprimidos ya que dichos datos sin comprimir pueden tener hasta 1 Gigabyte de tamaño para 81 vistas de 81 luces, con imágenes de 256x256, según Wai Kit Addy Ngan [64].

Se han desarrollado otras aproximaciones más sencillas para medir un conjunto de parámetros, por ejemplo Srinivasa G. Narasimhan et. al. [82] desarrollaron un pequeño dispositivo de adquisición el cual envía un haz de luz a través de un medio y con una cámara en la parte inferior del medio miden los parámetros

de dispersión del medio, así como su fase y su índice de absorción de la luz, entre otros; posteriormente usan estos parámetros en un algoritmo Motecarlo para generar imágenes fotorrealistas con muy buenos resultados.

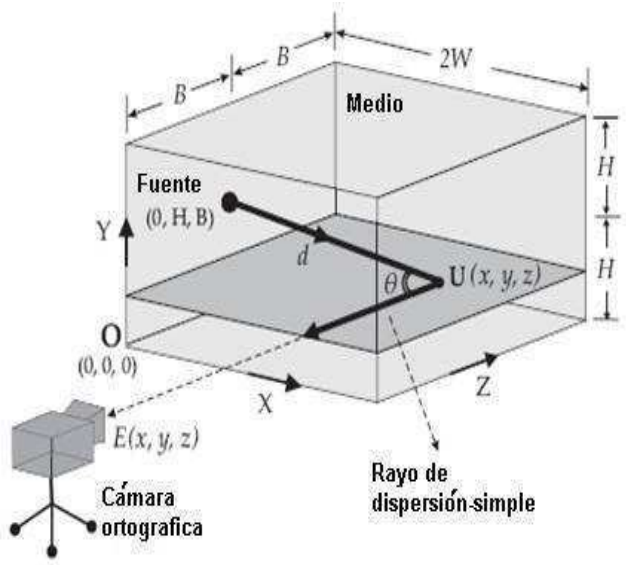


Figura 2.5: Esquema de dispositivo de adquisición de propiedades de dispersión en medios participantes diluídos

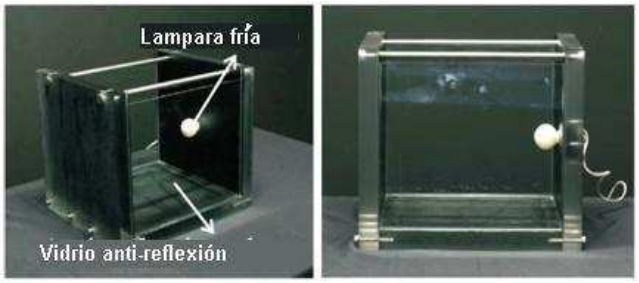


Figura 2.6: Dispositivo de captura de propiedades de dispersión en medios participantes diluidos

La mayoría de estas técnicas requieren un costoso hardware especializado y los resultados que buscan son de propósito específico, en nuestro caso se propone una aproximación diferente, usar un dispositivo de adquisición sencillo y técnicas heurísticas poblacionales para la obtención de los parámetros de iluminación que requiera el usuario.

La diferencia fundamental de nuestra aproximación es que las técnicas revisadas para la extracción de parámetros se basan en un procedimiento sin retroalimentación 2.7, mientras que nuestra propuesta tiene una fase de retroalimentación propia del algoritmo bioinspirado y que permite que los parámetros se ajusten y se vayan optimizando a medida que avanzan las iteraciones, 2.8.

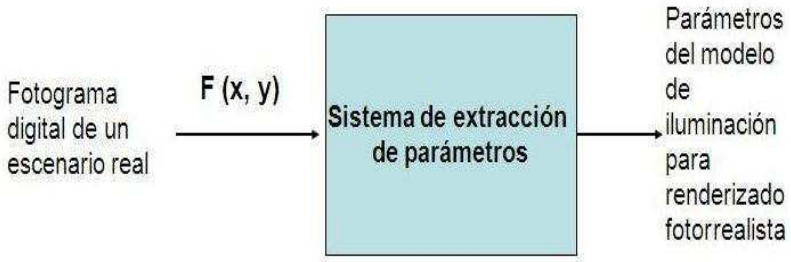


Figura 2.7: Metodología actual de obtención de parámetros

2.3.1. Renderizado evolutivo

Los algoritmos evolutivos para el renderizado de imágenes ya han sido usados con anterioridad en una variedad de aspectos. En su trabajo de renderizado de pinturas 2D, John Collomose comenta, "aplicamos Algoritmos Genéticos para la selección del estilo usando una evaluación interactiva estética y se le permite al usuario seleccionar estilos visuales diferentes, tales como expresionismo, puntillismo, impresionismo y abstracto", [14]. Ellos usan una selección de paráme-

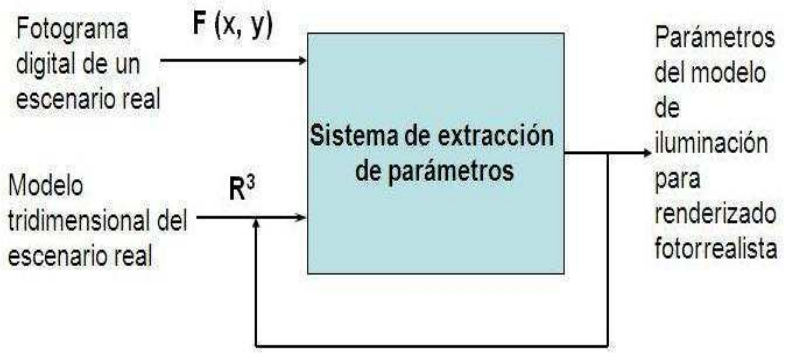


Figura 2.8: Metodología propuesta de obtención de parámetros

tros de bajo nivel para ajustar el estilo visual. Asimismo Collomose y Hall [53], han desarrollado un sistema automático para pintura adaptiva aplicando técnicas de búsqueda evolutiva para el aprendizaje de las máquinas, usando Algoritmos Genéticos como un mecanismo para el control automático del detalle en la realización de las pinturas, pero únicamente en un sólo estilo de pintura.

Tatsuo Unemi es un artista que usa expresiones matemáticas para generar imágenes por medio del uso de distintas versiones de su software BART [84] y [89].

En la generación de imágenes 3D artísticas los científicos y artistas recurren a una gran variedad de técnicas para evolucionar la geometría 3D en varios dominios. William Latham y Stephen Todd trabajaron en unas de las primeras formas orgánicas creadas usando su software en 1990[87]. Posteriormente, Rowbottom desarrolló una implementación, para máquinas PC, de la aproximación de Latham a la cual llamó Form [74]. Latham y Todd desarrollaron un sistema llamado PC Mutator para permitir que su aproximación genética interactiva se pudiera comunicar con otros paquetes de PC, tales como programas de dibujo [88].

Para el proceso de renderizado, los algoritmos evolutivos han sido usados para obtener una mejor distribución de los haces en un algoritmo de renderizacion por algoritmo de trazador de rayos, usando el supermuestreo de un haz de luz, como el sistema usado por Dutre, Suykens, y Willems [69]. Beyer y Lange aplicaron algoritmos genéticos para solucionar el problema de la integración de la radianza en una semiesfera [7]. Ellos usaron los individuos como rayos, los cuales evolucionaron para mejorar su distribución en la superficie con respecto a la radianza incidente. Otra aproximación es la usada por Veach y Guibas, donde las rutas de los haces son seleccionadas usando un método Metropolis para mutar las rutas en lugar de la más comúnmente usada técnica Montecarlo. Su algoritmo comienza con unas cuantas trayectorias de transporte de la luz y les aplica mutaciones aleatorias. Como resultado “En el estado fijo, las cadenas de Markov resultantes, visitan cada trayectoria con una probabilidad proporcional a la contibución de dicha trayectoria a la imágen” [90].

Nuestra propuesta es diferente, ya que nosotros usamos la evolución y el algoritmo de renderizado, como una herramienta para la estimación de los parámetros ópticos con el objeto de lograr una mejor aproximación a una imagen de referencia de un escenario real. En nuestro caso no se usa para la distribución de trayectorias ni para la generación de imágenes artísticas.

2.3.2. Optimización de parámetros de renderizado

Existe poco trabajo previo que se relacione con lo realizado en la presente tesis, son trabajos que abordan la optimización de parámetros para el renderizado usando técnicas de optimización. A continuación lo describiremos brevemente.

El primero es una propuesta para diseñar la iluminación de un ambiente usando técnicas de optimización aplicadas a un sistema de renderización que usa Radiosidad basada en un sistema de síntesis de imagen, realizada por Kawai, Painter y Cohen [52]. Esta propuesta se basa en la optimización de los parámetros de la iluminación y trabaja basándose en restricciones y objetivos especificados por el usuario para modificar la iluminación del ambiente. Este sistema de Radioptimización resuelve las "mejores" posibles configuraciones para: emisividad de la fuente de luz, reflexividad de los elementos y parámetros de direccionalidad de la fuente de luz. Esta propuesta ya tiene un escenario prediseñado donde los parámetros de los objetos tales como el color, la dureza, el índice de reflexión especular y la difracción los propone el usuario de antemano, a diferencia de nuestra propuesta que los propone durante la extracción de los parámetros. El objetivo de la Radioptimización es obtener condiciones de iluminación tales que minimice la energía usada para la iluminación del escenario a renderizar o para proporcionar a la escena la impresión de "privacidad". Como método de optimización usan un sistema de optimización con restricciones que usa el algoritmo BFGS (BroydenFletcherGoldfarbShanno), el cual evalúa la función objetivo y el gradiente en el paso corriente en el diseño del espacio para calcular una dirección de búsqueda. El método de radioptimización explora únicamente una trayectoria posible en la aplicación de las técnicas de optimización para los problemas de diseño de síntesis de imagen.

Otra propuesta es la de Yu, Debeverec, Malik y Hawkins [96] así como la de Yu [97], en la cual se estudia el problema de recuperación de modelos de reflectancia para escenas realistas a partir de fotografías. Este método recupera las propiedades de reflectancia de todas las superficies de una escena real a partir de un conjunto de fotografías y después las usa para renderizar, con métodos de renderizado "convencionales", una versión virtual de la escena, en la que se han mapeado las texturas correspondientes a la escena real, pero que responde a las condiciones virtuales de iluminación. El objetivo es encontrar los parámetros de la BRDF (función bidireccional de distribución de la reflectancia) para usarse en una renderización. Este método permite agregar modificaciones arbitrarias a la estructura e iluminación, por ejemplo objetos extra. La entrada al sistema es un modelo geométrico de la escena y un conjunto de fotografías de alto rango dinámico tomadas con una iluminación directa conocida. Una vez obtenidos los parámetros y renderizadas las imágenes, se procede a comparar las imágenes gen-

eradas con imágenes del escenario real tanto en condiciones originales como en nuevas condiciones, el resultado es que el método predice adecuadamente la imagen resultante bajo condiciones nuevas de iluminación. Los parámetros que se obtienen son el índice de reflexión difusa y el índice de reflexión especular para los colores rojo, verde y azul.

El problema de la iluminación inversa tambien ha sido abordado por Schonenman et al. en su articulo "Painting with ligth", [78]. Donde se propone una solución en la cual el diseñador del escenario usa una "brocha de luz" con la cual puede especificar las áreas de una imagen renderizada de la escena en las cuales él desea que sean iluminadas con cierto nivel de intensidad, su sistema busca la mejor configuración de luces para iluminar la escena por medio de la minimización de la diferencia entre la escena renderizada y la iluminación deseada. Como algoritmo de renderización se usa la traza de rayos.

Finalmente Elorza y Rudomin [21], desarrollan una propuesta donde se realiza un diseño de iluminación por medio de síntesis de imagen en ambientes cerrados, basado en una solución del problema de la iluminación inversa, aplicando un algoritmo genético como técnica de optimización y un algoritmo de radiosidad como técnica de renderizado. Los parámetros que se optimizan son: la cantidad, la posición y la intensidad de las fuentes de iluminación usadas en la escena. El objetivo es encontrar los parámetros que generen una imagen minimizando la energía usada y maximizando la iluminación.

Nuestra propuesta, a diferencia de las anteriores, obtiene un conjunto más amplio de parámetros como son: los índices de reflexión difusa, de difracción, de rugosidad, y de brillantez, así como los parámetros de la fuente de iluminación como son posición e intensidad de la fuente.

Capítulo 3

Marco teórico

3.1. Introducción

Este capítulo está dedicado a proporcionar conceptos básicos de dos temas fundamentales de este trabajo de tesis, el proceso de renderizado y la computación heurística bioinspirada. Estos temas se consideran importantes para establecer claramente el problema que se desea solucionar. Inicialmente se comienza con los conceptos fundamentales del proceso de renderizado y de la radiometría. También se revisa el proceso de la percepción del color. Posteriormente se pasa a las heurísticas poblacionales bioinspiradas como una herramientas que se propone para solucionar la obtención de los parámetros ópticos de renderizado. Finalmente se hace una revisión de la arquitectura de una Unidad de Procesamiento Gráfico (GPU, Grapchics Processor Unit) y las características del sistema de pruebas.

3.2. Antecedentes

Las gráficas por computadora tienen una relevancia muy grande en la vida diaria, desde la medicina, la geografía, la exploración, las simulaciones de física de partículas de alta energía hasta el entretenimiento [95], Anexo A.

A las gráficas generadas a partir de los datos de un escenario virtual se les denomina gráficas renderizadas. El término render se puede traducir como "interpretar", de tal forma que cuando se tienen datos de computadora que representan un escenario estos datos deben ser interpretados para generar una imagen en dos dimensiones que sea susceptible de ser presentada en un dispositivo de despliegue tal como un monitor de computadora o un cañón proyector.

3.3. El proceso de renderizado

En general el proceso de renderizado se puede dividir en 3 fases: aplicación, geometría y rasterizado. Como se puede apreciar, figura 3.1, se tiene un archivo con la definición de un escenario, con sus objetos, texturas, fuentes de iluminación, etc. El archivo es la entrada del proceso de renderizado y la salida resultante es la imagen generada o renderizada.

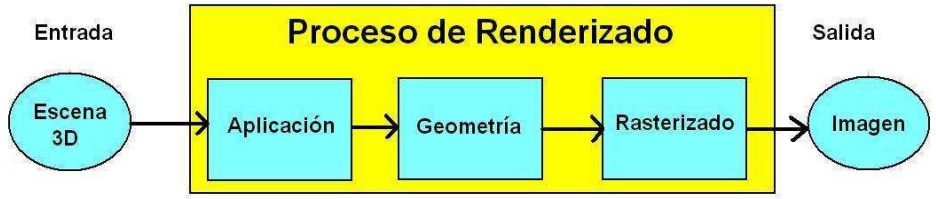


Figura 3.1: Etapas del proceso de renderizado

La computadora *interpreta* una escena 3D y la plasma en una imagen 2D. La renderización se desarrolla con el fin de imitar un espacio en 3D formado por estructuras polígonales, comportamiento de luces, texturas, materiales, animación, simulando ambientes y estructuras físicas verosímiles, etc.; Esto es, renderizar es pasar información del formato que maneja la computadora (VRML, JAVA3d, estructuras en ensamblador, objetos en código máquina, etc.), a una imagen en 2 dimensiones que dé la apariencia de 3 dimensiones, [60]. En la fig. 3.2 se puede apreciar un segmento de código VRML y del lado derecho se observa el resultado de la renderización de dicho código.

3.4. Radiometría

La radiometría es el campo de la ciencia encargado del estudio de la medición de la energía electromagnética, se encarga del estudio del transporte de la energía, en el caso de la luz visible se encarga del estudio del transporte de la luz. En las gráficas por computadora, el transporte de la luz es simulado únicamente en el espectro visible de la luz. Algunos fenómenos del comportamiento de la luz, respecto su análisis como una propagación de onda, son ignorados por ejemplo: la fosforescencia, la fluorescencia, la interferencia y la polarización.

En las gráficas por computadora un modelo de iluminación corresponde a la forma en la que será tratado el fenómeno de la incidencia de la luz visible sobre

```
#Edificio
Transform {
  translation 10 8 0
  children Inline {url
"edificio00.wrl"}
}

#Acera
Transform {
  translation 10 0.075 5
  children Shape{
    geometry Box {size 6
0.1 10}
    appearance
Appearance {
      texture
ImageTexture {url
"acera.jpg"}
      textureTransform
TextureTransform{
        scale 2 2
      }  } } }
```

Figura 3.2: Código de VRML que genera una imagen 3D

un punto y como va a reaccionar dicho punto dependiendo de las propiedades del objeto donde se encuentra ese punto. El material del objeto, desde el punto de vista de las gráficas por computadora, presenta una coloración dependiente de las propiedades de absorción y emisión del objeto para cada longitud de onda así como de las propiedades de la fuente de emisión de luz y del conjunto de elementos que componen el escenario. Por lo anterior el color de un punto depende de 3 factores:

- La aportación de la fuente
- La aportación del objeto
- La aportación de los otros objetos

La ecuación de renderizado, James T. Kajiya [56], dice que la energía que fluye en una pequeña región del espacio debe ser igual a la energía que sale, calculando:

$$SALIENTE - ENTRANTE = EMITIDA - ABSORBIDA \tag{3.1}$$

El proceso de renderizado se puede apreciar más claramente si se toma como base la ecuación de renderizado definida por Kajiya

$$L_0(x, \omega_0) = L_e(x, \omega_0) + L_r(x, \omega_0) \tag{3.2}$$

Donde:

- $L_0(x, \omega_0)$ es el total de la luz saliente
- $L_e(x, \omega_0)$ es la suma de la luz emitida

- $L_r(x, \omega_0)$ es la suma de la luz reflejada

La ecuación indica que la energía que abandona el punto x, en dirección ω_0 será igual a la energía emitida del punto x en dirección ω_0 más la totalidad de todas las aportaciones de energía circundantes.

$$L_0(x, \omega_0 = L_e(x, \omega_0) + \int_{H^2} f_r(x, \omega_i \rightarrow \omega_0) L_i(x, \omega_i) \cos\theta_i d\omega_i \tag{3.3}$$

Donde:

- $L_0(x, \omega_0)$ es la luz saliente en una dirección y posición particular
- $L_e(x, \omega_0)$ es la luz emitida
- $\int_{H^2} f_r(x, \omega_i)$ es la suma de todas las aportaciones de la luz reflejada
- $L_i(x, \omega_i) \cos\theta_i d\omega_i$ es la suma la luz incidente de todas las direcciones multiplicada por la superficie de reflexión y el ángulo de incidencia

Las aportaciones circundantes estarán compuestas por las propiedades del material definidas por fr y la integración de las energías que lleguen al punto x desde la dirección considerada multiplicado ω_i por el coseno entre la dirección de incidencia y a la normal del punto x, para cada una de las direcciones de incidencia.

Basado en lo anterior se puede observar que la energía que abandona un punto está compuesta por la suma de las componentes locales y globales de iluminación. Donde las componentes locales se refieren a las propiedades del material como índice de emisión, de reflexión y de refracción. Y las componentes globales se refieren a: la luz ambiental, la luz emitida por otros objetos y la luz emitida por las fuentes de iluminación.

El cálculo del color del punto entonces se compone de las siguientes aportaciones:

- La reflexión difusa de la luz ambiental
- La reflexión difusa y especular de las fuentes sobre el objeto
- La reflexión especular en otros objetos
- La transmisión especular en otros objetos
- La luz emitida por el propio objeto

La ecuación que lo define:

$$\begin{aligned} I(Q) = & ka \times Ia \times C \\ & + SUMi : 1..m(faten(Q)i \times (kd \times Idi \times (N\Delta Li) \times Cd \\ & + ks \times Isi \times (V\Delta Ri)^n \times Cs)) \\ & + kr \times Ir \\ & + kt \times It \\ & + Ce \end{aligned} \tag{3.4}$$

Donde:

- $I(Q)$ es el color del punto a calcular
- ka es la reflexión difusa de la luz ambiental
- Ia es la intensidad de la luz ambiental
- C es el color del objeto
- kd es la constante difusa del objeto
- Idi es la intensidad de la iluminación difusa del objeto
- ks es la constante especular del objeto
- Isi es la intensidad de reflexión especular del objeto
- $(V\Delta Ri)$ es el ángulo de incidencia de la luz en el objeto
- Cs es el color de la reflexión especular
- kr es la constante de reflexión del objeto
- Ir es la intensidad de reflexión del objeto
- kt es la constante de transmisión del objeto
- It es la intensidad de transmisión del objeto
- Ce es la luz emitida por el propio objeto

Esta ecuación define el modelo de iluminación a usar para calcular el color de cada punto en la imagen a renderizar. Para que este modelo de iluminación genere imágenes realistas es necesario conocer los mejores parámetros ópticos que se usarán para definir a las aportaciones. La estimación de algunos de estos parámetros es el objetivo de la metodología desarrollada en la presente tesis.

3.4.1. Percepción del color

El ojo humano tiene su mayor sensibilidad al color en la longitud de onda de 555 nm que corresponde a un color entre amarillo y verde y su mínima sensibilidad la tiene en un color entre el rojo y el violeta. En la luz del día o con buena iluminación se tiene lo que se denomina "Visión fotópica" donde actúan ambos fotorreceptores del ojo humano, los conos y los bastones.

En condiciones de iluminación pobre, como en la noche se tiene la "Visión escotópica" donde se produce el efecto Purkinjen [85] y [91], en el cual la curva, de sensibilidad de los receptores, se desplaza hacia las longitudes de onda más bajas modificando la sensibilidad máxima en la longitud de onda de 507nm, donde no trabajan los conos. Lo que significa que ya no hay visión de color. Sin embargo el ojo se vuelve muy sensible a la energía en el extremo azul del espectro y casi ciego al rojo.

Todo lo anterior debe ser considerado cuando se elige la fuente de iluminación a usar, ya que se debe seleccionar una fuente de iluminación de luz diurna o nocturna, dependiendo de las condiciones del objeto, donde se está analizando, y en que condiciones se va a renderizar.

Rendimiento del color

Una fuente de emisión de luz puede ser considerada de un alto rendimiento de color si emite la mayoría de los colores de espectro visible, una luz con un rendimiento ideal debería emitir todos los colores visibles. Si falta un color, la fuente de emisión disminuye su rendimiento de color.

El CRI (Color Rendering Index, Índice de Reproducción Cromática) indica las propiedades de una fuente de luz, a los efectos de la reproducción de los colores [76]. Este factor es determinado al comparar el aspecto cromático que presenta un objeto iluminado por una fuente de luz determinada, con respecto a una fuente de referencia. En el caso de la luz de día se denomina como continua y se dice que tiene un CRI = 100. Aunque en realidad le falta componente de luz Roja, lo mismo sucede con la luz incandescente que también se considera con un CRI de 100. Si se usan lámparas de descarga (fluorescentes), estas lámparas generan un espectro discontinuo, ya que presenta líneas espectrales que son propias del material emisor. Adicionalmente presenta la falta de componente Azul y tienen un CRI máximo de 20. El caso de las fuentes de emisión LED es similar a las lámparas fluorescentes ya que para generar luz blanca los LED generan una luz azul y después pasa a un material similar al de las lámparas fluorescentes para generar la luz blanca, por lo cual también tiene una banda de absorción propia

de dicho material y se puede considerar que su emisión es discontinua. Sin embargo y a diferencia de las lámparas fluorescentes, la emisión de luz generada por un LED no tiene una frecuencia de presentación, como si la tienen las lámparas fluorescentes. Los LED de luz blanca presentan un CRI de hasta 90, de acuerdo a [68] y [34], los LEDs presentan una mayor fidelidad al color que las lámparas fluorescentes no generan, y aunque tengan un mismo CRI. Y esto debe ser considerado al seleccionar la fuente de iluminación a usar.

Modelos de color lineales contra logarítmicos

Si bien es cierto que el ojo humano recibe y procesa la información de forma logarítmica y que sería más lógico trabajar con modelos de color logarítmicos como el CMY, es necesario aclarar que el ojo humano no interviene en el proceso de optimización. La metodología propuesta trabaja con una ecuación que mide la suma de las diferencias totales de la imagen y la suma de las diferencias de un conjunto de puntos, dicha diferencia es una relación lineal y como tal el algoritmo se maneja en un espacio lineal, no logarítmico. También se debe recordar que los algoritmos de optimización usados, PSO y Evolución Diferencial, trabajan en un espacio de búsqueda lineal y hacerlos que trabajen en un espacio de búsqueda logarítmico y verificar sus resultados queda fuera del objetivo del presente trabajo.

3.5. Heurísticas bioinspiradas

Los algoritmos evolutivos se pueden comprender conociendo el hecho de que son técnicas bioinspiradas que trabajan en un espacio de soluciones finito, por medio de la modificación de elementos denominados individuos, los cuales pueden ser modificados a través de dos operadores básicos, la mutación y la cruza.
En realidad es un conjunto de agentes con reglas de comportamiento basadas en algunas reglas básicas de la evolución. El algoritmo genético simple es desarrollado por John Holland y se describe a continuación.

1. Generar aleatoriamente la población inicial
2. Calcular aptitud de cada individuo
3. Seleccionar probabilísticamente en base a la aptitud
4. Aplicar cruza a los padres para producir la población de hijos
5. Aplicar mutación a los hijos
6. Los hijos forman la nueva población aplicando elitismo
7. Repetir desde el paso 2 un número predeterminado de veces

A partir del desarrollo de Holland se abre un campo de investigación el cual crea técnicas de optimización inspiradas en su trabajo, este nuevo campo genera posteriormente nuevas propuestas como son: las estrategias evolutivas, la programación genética, la optimización por colonia de hormigas, etc. En este trabajo se aplican dos de las técnicas más rápidas para encontrar buenas soluciones, la evolución diferencial y la optimización por cúmulo de partículas.

3.5.1. Evolución diferencial

El algoritmo de evolución diferencial ED es una heurística poblacional de optimización bioinspirada, desarrollada por Storm y Price en 1995, se desarrolló en un principio para resolver polinomios complejos de Chevichev. Los principios de su desarrollo son la eficiencia, simplicidad y el uso de variables de punto flotante en lugar de números binarios. La idea básica surge cuando Ken Price propone usar diferencias entre vectores para perturbar los vectores de las poblaciones. Al igual que otros algoritmos evolutivos, ED usa operadores de mutación, cruza y de selección, para evolucionar a sus individuos.

Al igual que los algoritmos evolutivos, la ED comienza con una población aleatoria la cual es mejorada al paso de las generaciones, usando la selección, cruza y mutación. Existen diversas formas para determinar el criterio de parada, de las cuales se pueden usar: la generacional ó el alcance del óptimo. Los parámetros de control para la ED son el parámetro de control de cruce CR, el factor de mutación F y el tamaño de la población NP.

En cada generación G, la ED pasa a través de cada vector de decisión $x_{i,G}$ de la población y crea un vector de prueba correspondiente a UiG, el cual se compara con $x_{i,G}$ y el de mejor aptitud es el que pasa a la siguiente generación.

ED genera vectores de prueba sumando la diferencia ponderada entre los vectores de población. Price y Storm llaman a esta operación mutación. Los parámetros mutados son entonces combinados con los parámetros de otro vector, el vector objetivo, para obtener lo que se denomina un vector de prueba. Si el vector de prueba obtiene un valor de costo de función menor que el vector objetivo, un vector de la población previamente seleccionado, el vector de prueba substituye al vector objetivo como individuo de la nueva población.

El vector objetivo puede ser parte de una generación de vectores, aunque no necesariamente. Según Storn y Price [83], la extracción de información de distancia y dirección de una población para generar desviaciones aleatorias proporciona un esquema adaptativo con excelentes propiedades de convergencia.

El funcionamiento básico del algoritmo de ED es de la siguiente forma:
Se define una población inicial con valor NP.

Mutación

Para cada vector $x_{i,G}$, donde i=$0, 1, 2, ..., NP-1$ un vector de prueba v es generado de acuerdo a:

$$v_{i,G+1} = \qquad x_{r1,G} + F * (x_{r2,G} - x_{r3,G}) \tag{3.5}$$

Considerando $r1$, $r2$, $r3$ $[0, NP-1]$ enteros mutuamente diferentes y $F > 0$. F es un factor real constante el cual controla la variación de las diferencias. Los enteros $r1$, $r2$, $r3$ deben ser seleccionados aleatoriamente del intervalo $[0, NP-1]$ y deben ser diferentes del índice i actual. De forma tal que NP debe ser mayor o igual a 4 para permitir esta condición.

Cruza

Con la finalidad de lograr un incremento en la diversidad de los vectores de parámetros, se define el vector u:

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, , u_{Di,G+1}) \tag{3.6}$$

Por lo que para:

$$\mathrm{u}_{ji,G+1} = \begin{cases} \mathrm{v}_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ o } j = rnbr(i) \\ \mathrm{x}_{i,G} & \text{if } (randb(j) > CR) \text{ y } j \neq \text{rnbr(i)} \end{cases}$$
$$\mathrm{j} = 1, 2, ..., \mathrm{D} \tag{3.7}$$

$randb(j)$ es un número aleatorio entre 0 y 1. $rnbr(i)$ es un índice seleccionado aleatoriamente que va desde 0 hasta D, el cual asegura que $u_{i,G+1}$ tenga por lo menos un parámetro de $v_{i,G+1}$ $CR[0, 1]$ es la probabilidad de cruce y constituye una variable de control, la cual ha demostrado ser muy importante para encontrar el óptimo en funciones con variables separables cuando CR tiene valores pequeños.

Selección

Para decidir si el vector u se convertirá en un miembro de la población $G+1$, se compara con el vector $x_{i,G}$. Si el vector u tiene un valor menor para su función objetivo que $x_{i,G}$. Entonces $x_{i,G+1}$ será el valor de u, de otra forma el valor anterior de $x_{i,G}$ será conservado. Es necesario tener en cuenta que para diferentes problemas, se deberán tener diferentes valores de NP, CR y F, con el fin de obtener los mejores resultados.

Existen diferentes variantes al algoritmo original, la nomenclatura usada para identificar a las variantes parte de un identificador inicial que comúnmente es DE para Differential Evolution. A continuación viene el tipo de selección de los individuos que generarán a la descendencia, éste puede ser aleatorio ($rand$) o el mejor de la población ($best$). El siguiente número es la cantidad de pares de soluciones escogidos. Finalmente se indica el tipo de recombinación usado que puede ser binario (bin) o exponencial (exp). Por ejemplo, la variante más comúnmente usada es la $DE/rand/1/bin$, la cual podemos saber es un algoritmo de evolución diferencial (DE), que genera a su descendencia usando una selección aleatoria ($rand$), con un par de soluciones escogidas (1) y usa una recombinación del tipo binaria (bin).

3.5.2. Optimización por cúmulo de partículas

El algoritmo Particle Swarm Optimization (PSO) ha tenido gran aceptación como una de las alternativas de optimización global basadas en heurísticas bio-inspiradas. Sus principales ventajas son su buen desempeño, baja complejidad computacional y el mínimo de parámetros requeridos para su operación. El algoritmo PSO intenta representar el movimiento de un conjunto de partículas o individuos que "vuelan" en un espacio n-dimensional en busca del óptimo global en forma colaborativa. En cada iteración del algoritmo se sigue una regla simple para actualizar la posición de tal manera que el movimiento, aunque esencialmente aleatorio, se ve influenciado por su propia experiencia (aprendizaje individual) y por el entorno (influencia social) [57].

El algoritmo PSO está basado en el movimiento de partículas o individuos que vuelan en un espacio de búsqueda de n dimensiones, intentando encontrar un óptimo global de forma colaborativa. La posición de la partícula es actualizada en cada iteración del algoritmo siguiendo una sencilla regla, de tal forma que el movimiento del individuo, esencialmente aleatorio, es influenciado por su propia experiencia y por el ambiente o la experiencia grupal. Originalmente el algoritmo fue propuesto por Kennedy y Heberhart en 1995 [57], basado en la posición y el cambio de posición para cada partícula. Posteriormente, el algoritmo fue mejo-

rado por Shi y Heberhart en 1998[79], por medio de la introducción del concepto de inercia. El algoritmo PSO es descrito a continuación:

Algoritmo de optimización por cúmulo de partículas:

1. Inicializar cada partícula de la población de forma aleatoria, generando así los valores para los vectores de posición y velocidad.
2. Calcular la aptitud respecto a la posición de cada partícula. Si la aptitud actual es mejor que la anterior, entonces actualizar con la mejor.
3. Determinar la localización de la partícula con la mejor aptitud global y comparar con la propia.
4. Para cada dimensión de cada partícula, la velocidad debe ser actualizada respecto a la siguiente ecuación:

$$v_{i,d}(t+1) = \omega \times v_{i,d}(t) + c \times r1 \times (pbx_{i,d} - x_{i,d}(t)) + c2 \times r2 \times (gbx_{i,d} - x_{i,d}(t)) \tag{3.8}$$

   donde ω es la inercia del sistema $c1$ y $c2$ son constantes que determinan la influencia del aprendizaje individual y la influencia social respectivamente; $r1$ y $r2$ son valores aleatorios, entre 0 y 1, que representan el libre movimiento de cada partícula.
5. Actualizar la posición x de cada partícula de acuerdo a la siguiente ecuación:

$$x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1) \tag{3.9}$$

6. Repetir los pasos 2 al 5 hasta alcanzar la condición de finalización (número de iteraciones o precisión).

Ambas técnicas, ED y PSO, usan una población inicial con individuos que representan soluciones al problema propuesto, estas soluciones deben ser representables en forma de un vector de números reales. Para el estudio del presente trabajo ambas técnicas de optimización son prometedoras debido a su naturaleza poblacional, y a su capacidad de exploración y explotación del universo de soluciones. Encontrar una solución óptima para los parámetros de renderización por medio de una búsqueda exhaustiva es impráctico, sin embargo el uso de PSO y ED ofrece una posibilidad de encontrar buenas soluciones.

3.6. Unidad de procesamiento gráfico

El sistema seleccionado para realizar las pruebas es un GPU (Graphics Processor Unit, Unidad de Procesamiento Gráfico) utilizado normalmente como coprocesador gráfico para sistemas de video de alto desempeño específicamente

en videojuegos y producción de video. Un GPU es una arquitectura de procesadores vectoriales capaces de ejecutar operaciones matemáticas sobre múltiples datos de forma simultanea, a diferencia de los procesadores escalares comunes en los CPU los cuales manejan una operación por dato. La gran mayoría de los CPU son procesadores escalares o algo muy similar. Los procesadores vectoriales eran comunes en computadoras de investigación científica [48] pero eventualmente fueron desplazados por arquitecturas escalares multinúcleo. Sin embargo no fueron eliminados ya que muchas arquitecturas de gráficos por computadora usan los procesadores vectoriales como base de su hardware. Al parecer, hoy la tendencia en el desarrollo de las unidades de procesamiento gráfico posibilita el retorno del cómputo científico hacia arquitecturas íntimamente relacionadas con los procesadores vectoriales, en los cuales tienen su origen.

De forma general un procesador tiene que buscar la instrucción para ejecutarla, este proceso toma tiempo, dicho tiempo genera latencia en la ejecución de las instrucciones. Para reducir este tiempo los procesadores modernos ejecutan un conjunto de instrucciones en forma concurrente realizando lo que se conoce como "instruction pipelining" (entubamiento de instrucción), en el cual las instrucciones pasan a través de varias subunidades en turnos, el primer elemento lee la dirección y la decodifica, el siguiente elemento obtiene (fetch) los valores contenidos en esas direcciones y la siguiente hace el cálculo matemático según la instrucción recuperada. Con el "instruction pipelining" se inicia la decodificación de la siguiente instrucción incluso antes de que la primera haya salido del procesador, de tal forma que el decodificador de instrucciones esté en un uso constante y se disminuyan las latencias. El tiempo de ejecución de una sola instrucción es el mismo, pero un conjunto de instrucciones se ejecuta en menor tiempo aumentando con esto el desempeño general del procesador.

En los procesadores vectoriales este concepto se lleva aún mas lejos, de forma tal que en vez de sólo entubar las instrucciones, también se entuban los datos, de manera que en lugar de estar decodificando instrucciones constantemente y obteniendo los datos necesarios para completarlas, se lee una sola instrucción y a continuación una gran cantidad de datos para operar esta instrucción sobre todos ellos, esto permite un gran ahorro de tiempo por la parte de la decodificación de instrucción.

Debido a lo anteriormente expuesto es posible visualizar un gran poder de cómputo en los procesadores vectoriales, aunque se tienen algunas reglas que seguir, una de las más importantes es que la instrucción a seguir debe ser la misma para una gran cantidad de datos, esto es especialmente cierto para operaciones con vectores o matrices. En las gráficas por computadora existe una gran

cantidad de operaciones matriciales, de hecho las imágenes se pueden visualizar como un conjunto de matrices de valores de intensidades o de profundidades. De forma tal que, cuando se va a realizar el cálculo de una imagen, se puede operar sobre conjuntos de matrices aplicando una sola instrucción a múltiples datos. Debido a lo anterior los procesadores vectoriales encontraron un nicho en la graficación por computadora y el hardware diseñado para la generación de imágenes en producción de video y videojuegos se basa en esta arquitectura.

Gracias a una gran demanda por juegos y producciones de video de apariencia fotorrealista en tiempo real, la producción de GPU cada vez más poderosos y con arquitecturas con mejor desempeño ha conducido al mercado de las tarjetas de video a generar propuestas multinúcleo, multihilo altamente paralelas, las cuales tienen un tremendo poder de cálculo que rememora a aquellas primeras supercomputadoras Cray de procesamiento vectorial.

Los GPU se han especializado en un cómputo intensivo altamente paralelo, debido a esto un gran porcentaje de los transistores usados para fabricar al GPU están dedicados al procesamiento de datos, más que al manejo y control de flujo de dichos datos, como se puede apreciar en la figura 3.3. Sin embargo,

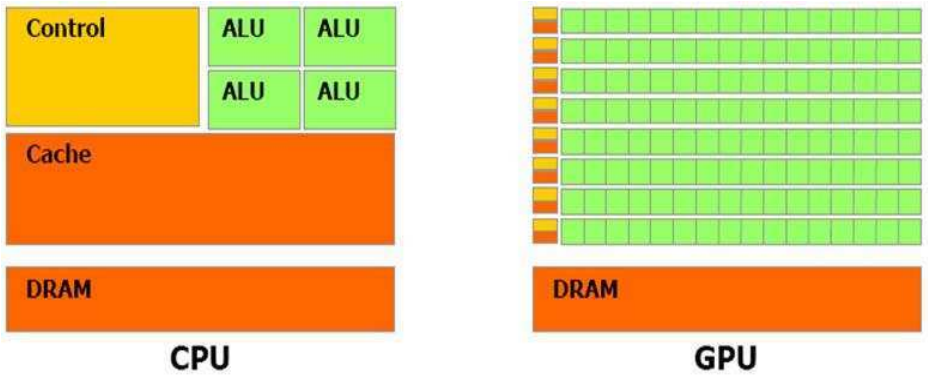


Figura 3.3: El porcentaje de transistores usados para procesamiento de datos en CPU y en GPU.

hoy la tendencia en el desarrollo de los GPU permite preveer la consolidación de un nuevo modelo de programación paralela, donde el GPU tiene un papel más preponderante como administradores de hilos al mismo tiempo que mantiene, e incluso aumenta, sus capacidades de cómputo paralelo:
"Debido a que el mismo programa se ejecuta para cada elemento de datos, existen bajos requisitos para el control de flujo y debido a que este procesamiento se ejecuta sobre una gran cantidad de datos con una alta intensidad aritmética la latencia en los accesos a memoria puede ser ocupada en cálculos en lugar de

caches de datos" [16].

Es evidente la ventaja que este nuevo enfoque en la arquitectura aporta para el procesamiento de imágenes, sobre todo debido al potencial de paralelizar cada pixel de una imagen en núcleos de procesamiento diferentes, pero no es el único tipo de cálculos que se puede beneficiar de esta arquitectura, de hecho casi cualquier tarea susceptible de ser paralelizada puede sacar ventaja de esta arquitectura, desde la simulación de partículas físicas, procesamiento de señales, detección de virus en redes, computación biológica, simulaciones de desplazamientos de grandes multitudes, etc.

El dispositivo a usar es un GPU Nvidia 8600GT, es un arreglo de procesadores masivamente multihilos, que el fabricante denomina arquitectura Tesla de cómputo y gráficas unificadas. Este diseño se construye alrededor de un arreglo escalable de multiprocesadores para flujo multihilo. Cada multiprocesador consiste de ocho núcleos de procesadores escalares, dos unidades especiales para trascendentes, una unidad de instrucciones multihilo y una unidad de memoria compartida en el GPU.

El multiprocesador se encarga de la creación, manejo y ejecución de los hilos actuales en el hardware, para poder manejar una cientos de hilos (teóricamente hasta 512) ejecutando diferentes programas, el multiprocesador puede hacer uso de lo que el fabricante denomina SIMT (Single Instruction Multiple Thread). El multiprocesador entonces debe mapear cada hilo a un núcleo de procesamiento escalar y cada hilo escalar debe ejecutarse de forma independiente de los demás, con su propia dirección de instrucción y registro de estado. Esto hace que la programación sea orientada al manejo de los hilos y la optimización en su uso.

Se usa el modelo de programación CUDA (Computer Unified Device Architecture, Arquitectura Unificada de Dispositivos de Cómputo), es un modelo de programación paralela para dispositivos GPU, desarrollado por NVIDIA. Consiste en un conjunto de programas y compiladores que facilitan la programación de programas en forma paralela usando GPUs de la marca NVIDIA.

3.6.1. Características del sistema.

Las pruebas se realizaron en una computadora con procesador Intel Core Duo, a la que denominaremos "Anfitrión", con un disco duro de 160 Gigabytes, memoria de acceso aleatorio (RAM) de 512 Mbytes, con un monitor LCD de 19 pulgadas. La tarjeta de video es una Tarjeta NVIDIA 8600GT, a la que denominamos GPU, con 256 Mbytes de memoria de trabajo, el procesamiento de

la tarjeta está integrada por 4 multiprocesadores de 8 núcleos de “Shading” los cuales suman en conjunto 32 núcleos de trabajo que pueden ser programados a través del sistema CUDA para funcionar como un cluster de coprocesamiento matemático.

El sistema de cómputo tiene un arranque dual de sistema operativo, una partición de Windows XP y otra de Linux. Las pruebas se realizaron en ambos sistemas operativos pero en Linux se obtuvieron menores tiempos (mejores) que en Windows, de forma que los experimentos se realizaron en Linux. 0

Capítulo 4

Metodología de extracción de parámetros

4.1. Introducción

El presente capítulo comienza con una breve descripción de la metodología actualmente usada para la obtención de parámetros ópticos para su uso en la renderización y posteriormente explica la metodología que el presente trabajo propone como una alternativa para la extracción de parámetros.

4.1.1. Metodología actual

Los trabajos presentados en la sección de estado actual, tienen en común una metodología de extracción ó cálculo de los parámetros ópticos a ser usados en la renderización, esta metodología obtiene los parámetros a usar y los aplica en modelos previamente preparados y renderiza imágenes con ellos. La imagen generada muestra el efecto del uso de los parámetros obtenidos. Sin embargo no se realiza una comparación de la imagen obtenida y la imagen generada en el proceso de la extracción/cálculo de los parámetros, figura 4.1.

La metodología se puede visualizar desde el punto de vista de la teoría del control, como un sistema de lazo abierto, en el que el proceso obtiene los parámetros y los usa. Se puede pensar que el usuario cierra el lazo al comparar la imagen renderizada con el objeto real o una imagen del objeto real y modificar, en respuesta a lo observado, el sistema de adquisición. Sin embargo el enfoque de nuestra propuesta es diferente del método usado actualmente, ya que incluye una retroalimentación dentro del proceso de extracción de parámetros, figura 4.2.

Figura 4.1: Metodología actual de obtención de parámetros

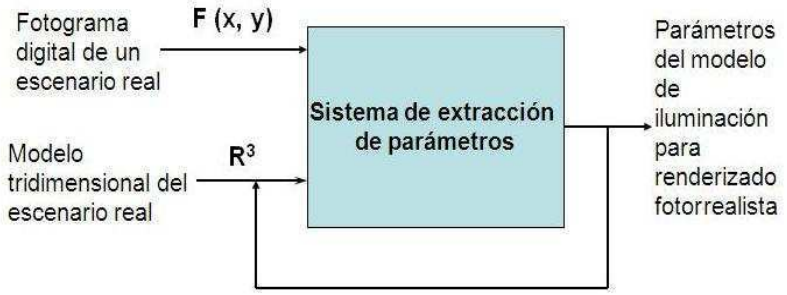


Figura 4.2: Metodología propuesta de obtención de parámetros

Por lo anterior se propone una metodología de extracción de parámetros físico ópticos para renderización de imágenes en la graficación por computadoras, que reciba un fotograma digital de un escenario real, como imagen de referencia, y una descripción de ese escenario en una mundo virtual, figura 4.3. La metodología propone el uso de un algoritmo de optimización bioinspirado el cual tiene como función objetivo la comparación de dos imágenes, una imagen de referencia o imagen objetivo y la otra imagen generada por cada individuo del algoritmo bioinspirado. Cada imagen generada se prueba contra la imagen de referencia y se obtiene un valor de proximidad, el cual se usa como aptitud que se asigna al individuo que generó dicha imagen, figura 4.4.

4.1.2. Propuesta de solución

Propuesta de algoritmo bioinspirado

1. Tomar una imagen de referencia con una cámara digital

   a) La imagen estará con condiciones mínimas de entorno externo

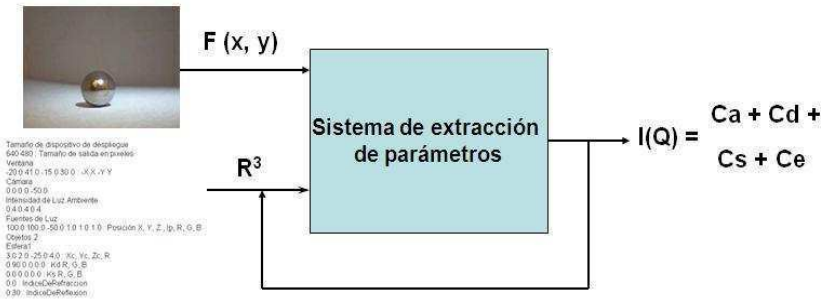


Figura 4.3: Entrada y salida de la metodología propuesta de obtención de parámetros

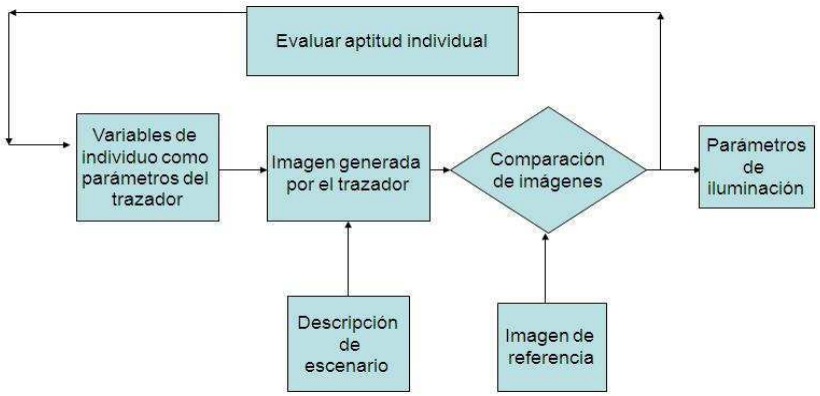


Figura 4.4: Diagrama de la metodología propuesta de obtención de parámetros

b) Se guardará en formato BMP

2. Usar una función objetivo que compara dos imágenes y muestra su diferencia en valores RGB por separado

4.2. Adquisición de imagen objetivo

El dispositivo de adquisición de imagen figura 4.5a, consiste de una caja de 15x15x20 cms, el cual tiene 4 leds de alta luminosidad como fuentes de luz, colocados en la parte superior de la caja, así como un orificio para una cámara web para la adquisición de las imágenes. A 6 cm de la base de la caja se coloca una sección para soportar los objetos a capturar. Cada led puede ser controlado de

manera independiente, de forma que se puede tener uno, dos, tres o los cuatro leds encendidos en cualquier orden.

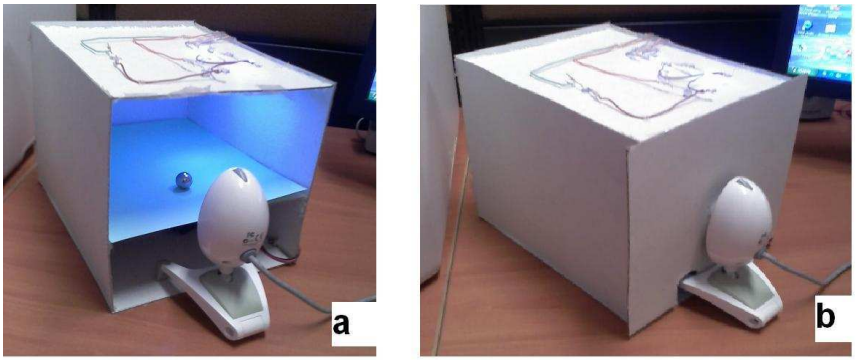


Figura 4.5: Sistema de adquisición de imágenes de referencia

Se eligió como fuente de emisión de luz a un Diodo LED debido a su dispersión uniforme de la luz, a diferencia de una fuente fluorescente, asimismo la emisión de luz blanca del LED presenta una menor dominancia de los colores verdes que la que presenta la fuente fluorescente y a diferencia de la luz natural no presenta una dominancia con los colores rojos. Se pueden conseguir hasta 115 lúmenes por LED a 350mA. El LED blanco puede alcanzar hasta un CRI de 90, de acuerdo a [68] y [34] los LEDs presentan una mayor fidelidad al color que las lámparas fluorescentes no generan, y aunque tengan un mismo CRI, lo que que no sucede ya que las lámparas fluorescentes alcanzan como máximo un CRI de 20. Como ya se mencionó anteriormente la emisión de luz generada por un LED no tiene una frecuencia de presentación, como sí la tienen las lámparas fluorescentes.

Como fuente de alimentación se seleccionó un conjunto de tres baterías como fuente de corriente continua, para que no presente variaciones con respecto a una fuente de energía alterna. El suministro de las baterias se controla con un multímetro para mantener un rango constante de 4.5 volts. Se usa un fondo blanco como revestimiento interior de la caja de captura y un suelo de color verde claro para apreciar la diferencia del suelo respecto al resto del entorno.

Para dispersar la luz de los LED, que son direccionales, en la parte superior de la caja se colocó un plástico blanco traslucido para dispersar de manera uniforme la luz de los LED. Para minimizar las reflexiones externas se coloca una tapa en la caja, figura 4.5b para que solo sea visible el lente de la cámara. Con este dispositivo se adquieren las imágenes de los escenarios a ser usados como referencia de la función objetivo.

Como objeto principal de los experimentos se eligieron unas esferas de plástico de varios colores, de un tamaño de 2.5 cm. El motivo de la elección es que las superficies curvas son de las más difíciles de aproximar en una renderizacion, por la variación gradual de los colores a lo largo de toda la superficie.

4.2.1. Condiciones de adquisición de imagen

El revestimiento interno de la caja es un material blanco mate para minimizar las reflexiones internas.

Los objetos a capturar siempre se encuentran a la misma distancia y con la misma iluminación, asimismo el lente de la cámara se coloca en un enfoque fijo.

Como sistema de adquisición de imágenes se eligió una cámara web marca Creative, modelo WebCam Instant, con capacidad de captura de imágenes a una resolución de 320 x 240 o de 648 x 480. Puede guardar sus imágenes en formato BMP o en formato JPG. Se seleccionó este modelo por tener la capacidad de almacenar sus imágenes en formato BMP. El tiempo de exposición se deja fijo para que no cambien las imágenes adquiridas respecto a las condiciones de iluminación, figura 4.6. El conjunto de los parámetros de la cámara es el siguiente:

- Exposición = 48
- Brillo = 20
- Contraste = 20
- Hue = 18
- Saturación = 50
- Sharpness = 3
- Gamma = 3
- Habilitar color = 1
- Compensación de luz posterior = 0

Formato de imagen

Se almacena la imagen de referencia en un formato BMP, el cual es un formato sin compresión, y que almacena la información en forma de matriz RGB, con valores de 0 a 255. El formato BMP tiene, en su cabecero, la información correspondiente a la resolución de la imagen a lo ancho y a lo alto (en pixeles), el número de planos a usar (normalmente cero), el número de bits por pixel, (en

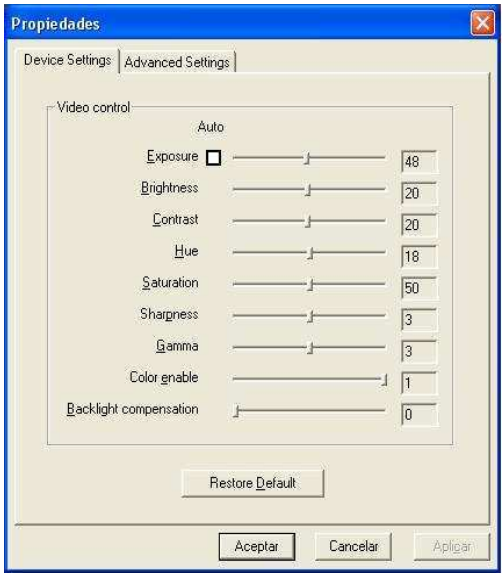


Figura 4.6: Parámetros usados en la cámara de adquisición

este caso 24), el tipo de compresión (normalmente a 0, en este caso 0), el número de datos en bytes de la imagen, etc. Sin embargo el formato BMP almacena la información invertida en el eje Y, esto significa que una imagen adquirida por el dispositivo de captura, se presentará invertida en el eje Y, esto debe ser considerado al momento de la comparación de las imágenes. También se debe tomar en cuenta que las imágenes se almacenan en bytes pero cada renglón debe tener un número de elementos correspondientes a un múltiplo de 4, de otra forma el algoritmo del formato BMP rellena con ceros el final del renglón. Estos ceros deben ser descartados en la lectura de la imagen para que no interfieran con la comparación respecto a la imagen generada por el algoritmo de renderizado.

La imagen capturada tiene una resolución de 320 x 240 píxeles y se le aplica un algoritmo de reducción de resolución para obtener una imagen de 128 x 96 píxeles. Al realizar le lectura del archivo BMP los bytes de información de cada píxel se almacenan en una matriz de 128 x 96 x 3. Donde 128 es la resolución horizontal en el eje X, 96 es la resolución vertical en el eje Y, y 3 es el valor para cada componente de color del modelo RGB (R-Rojo, G-Verde, y B-Azul).

4.2.2. Función objetivo

Para comparar ambas imágenes se propone la sumatoria del valor absoluto de la resta del valor de cada posición xy de la imagen objetivo (RGBObj) respecto a la misma posición de la imagen generada por el renderizador basado en trazador

de rayos (RGBTraza), eq. 6.4. La función de comparación se propusó en este trabajo de tesis pero después se encontró que es la misma que se propone como la primera de una serie de funciones para comparar las diferencias de dos imágenes en el trabajo de H. Rushmeier [38], Rushmeier et al. lo expresan como M(A;B) pero es la distancia entre las imágenes A y B. Midiendo la diferencia entre cada píxel de la imagen generada con respecto al píxel correspondiente en la imagen de referencia. A medida que esta diferencia disminuye, las imágenes son más parecidas entre ellas, cuando la diferencia es cero las imágenes son idénticas.

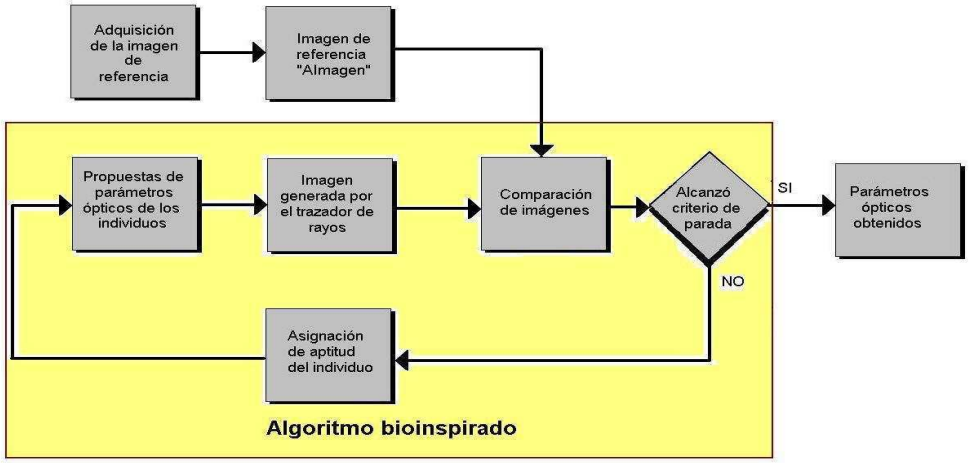


Figura 4.7: Metodología de obtención de parámetros

F01- Función de evaluación para asignar aptitud, considerando todos los píxeles de la imagen:

$$Dif = \sum_{i=0}^{i=n} |RGBObj_i - RBGTray_i| \tag{4.1}$$

Donde:

- n = Xmax * Y max
- RGBObji = Valor de cada componente del píxel i de la imagen objetivo
- RGBTrazai = Valor de cada componente del píxel i de la imagen generada por el trazador de rayos

La función objetivo entrega un valor al recibir dos imágenes, con un valor elevado la diferencia de las imágenes es grande, con un valor reducido la diferencia de las imágenes es pequeña. Esta función, permite conocer la diferencia entre dos imágenes en un valor de hasta una unidad de intensidad en una sola

componente del modelo RGB. La función se aprecia en el diagrama de bloques de la metodología, etiquetada como "comparación de imágenes", figura 4.7

F02- Función de evaluación para asignar aptitud, considerando sólo los puntos de interés sobre los píxeles de la imagen:

$$Dif = \sum_{i=0}^{i=n} |RGBObj_i - RBGTraz_i| \tag{4.2}$$

Donde:

- n = Puntos de interés.
- RGBObji = Valor de cada componente del píxel i de la imagen objetivo
- RGBTrazai = Valor de cada componente del píxel i de la imagen generada por el trazador de rayos

Prueba de función objetivo

Se probó la función con una imagen adquirida contra la misma imagen con un píxel de diferencia. Dos imágenes iguales con una variación de un píxel dan una diferencia de: 147+ 137+ 134 = 418

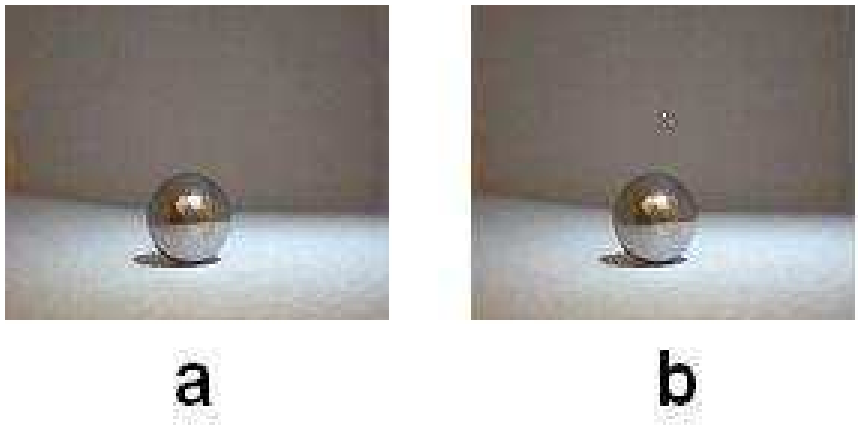


Figura 4.8: Imágenes con un píxel de diferencia, a) imagen de referencia b) imagen modificada

4.3. Algoritmo propuesto

El algoritmo propuesto recibe una imagen de referencia, la que el sistema de adquisición obtuvo, y la almacena en una matriz de 128x96 denominada Matriz-

Obj, posteriormente inicializa las partículas del algoritmo PSO (Particle Swarm Optimización) y genera una población de partículas, posteriormente evalúa la aptitud de cada partícula. La evaluación se divide en dos fases:

- 1.- Usando el vector X de la partícula seleccionada, se asigna cada variable del vector a un parámetro de un escenario virtual, el escenario virtual se usa como entrada para un algoritmo de traza de rayos el cual genera una imagen de 128 x 96 que se almacena en una Matriz llamada MatrizTraza.

- 2.- Se llama la función de comparación de las dos imágenes recibiendo como entrada la MatrizObj y la MatrizTraza, generando como resultado un valor numérico correspondiente a la diferencia entre ambas imágenes, este valor es el que se guarda como aptitud de la partícula.

El PSO evalúa todas las partículas y selecciona la partícula de mejor aptitud para cada vecindario, posteriormente compara la aptitud entre las mejores partículas de cada vecindario y selecciona la mejor. La información de la mejor partícula se usa para que el PSO realice sus cálculos de actualización de velocidad y el nuevo valor de cada variable del vector X, asimismo se almacenan los valores de la mejor partícula de cada iteración en un archivo en disco. Finalmente los parámetros representados por los valores de la mejor partícula se usan para renderizar una imagen que se guarda en un archivo BMP de 128x96, en el directorio correspondiente a esa iteración. Solo se guarda una imagen cada 5 iteraciones para no saturar al dispositivo de almacenamiento.

Debido a la naturaleza de la heurística bio-inspirada poblacional que se usa como parte de la metodología, el algoritmo tiene una naturaleza adaptativa, ya que a diferentes condiciones de los modelos, a los cuales se les extraerán sus parámetros, el proceso de extracción se adaptará para obtener los mejores resultados. Si se usa un objeto rojo, la metodología se comporta de diferente forma que para un objeto Azul o uno Verde. El pseudocódigo del algoritmo se presenta a continuación.

4.4. Algoritmo

```
ImagenO = Obtención de  Imagen
   Iniciar partículas
   Para cada partícula
      Para cada Variable
         Selecciona un valor aleatorio entre 0.0 y 1.0
      Crea vecindarios
      Mientras (el criterio de parada no se cumpla)
       Para partícula=1 a numero de partículas
```

```
 Calcula Aptitud
  Usa variables de la partícula como parámetros del trazador
     Ejecuta trazador de rayos y genera imagenT
     Compara ImagenO con ImagenT
  Calcula Velocidad
      Si Aptitud de (Xp > Mejor de vecindario)
         Actualiza Mejor de Vecindario con aptitud de Xp
      Calcula Mejor de Vecindario
      Calcula Mejor de los vecindarios
 Fin de ciclo de partícula
Próxima Iteración hasta criterio de parada
```

Parámetros del PSO

- Partículas = 25
- Variables = 30
- Vecindarios= 5
- Grupos = 5
- Iteraciones= 302
- VMax = 0.020000
- VMin = -0.020000

Resultados esperados

- Encontrar un conjunto de índices de color, reflexión y difracción que sirvan para proponer un modelo de iluminación óptimo para las imágenes generadas por computadora.
- Lograr un balance adecuado de los parámetros de iluminación local y global como resultado de las aproximaciones por algoritmos bioinspirados.

4.4.1. Generación de archivo de escena

Para generar la imagen por traza de rayos, es decir la imagen generada, es necesario que el algoritmo de traza de rayos tenga como entrada un archivo que describe la escena en el entorno virtual, el archivo contiene los elementos correspondientes al fondo de la escena, al piso, al objeto central a renderizar, a la fuente de luz y a la cámara virtual que sirve como observador. El tamaño y la posición de todos los elementos, menos la fuente de iluminación, son ajustados a mano, esto significa que la posición del suelo, y el tamaño y la posición del objeto así como la distancia de la cámara así como su apertura son ajustados por el usuario, para

que concuerden en tamaño y en posición con los objetos equivalentes de la imagen de referencia. El formato del archivo de escena se presenta a continuación:

```
Tamaño de dispositivo de despliegue
128 96 : Tamaño de salida en pixeles
Ventana
-20.0 41.0 -15.0 30.0 : -X X -Y Y
Cámara  X, Y, y Z
0.0 0.0 -50.0
Intensidad de Luz Ambiente
0.4 0.4 0.4
Fuente de Luz
100.0 100.0 -50.0 1.0 1.0 1.0 : Posición X, Y, Z, Ip, R, G, B
Objetos 2
Esfera1
3.0 2.0 -25.0 4.0 : Xc, Yc, Zc, R
0.90 0.0 0.0 : Kd R, G, B
0.0 0.0 0.0 : Ks R, G, B
0.0 : IndiceDeRefracción
0.30 : IndiceDeReflexión
Plano1
0.0 -10.0 0.0 -2.0 : Pn, Pn, Pn, X
0 30.60 0 : P0, P1, P2
0.99 0.99 0.99 : Kd R, G, B
1.0 1.0 1.0 : Ks R, G, B
0.0 : IndiceDeRefracción
0.0 : IndiceDeReflexión
```

4.4.2. Codificación de los parámetros de renderización

El vector de variables de cada individuo tiene correspondencia con los parámetros de iluminación del escenario. Las tres primeras variables son la posición X, Y y Z de la fuente de iluminación, Pos_L. La cuarta variable es la intensidad de la fuente. Los valores siguientes corresponden a los valores RGB de la esfera (esf) y del plano (pla), como se puede apreciar en la tabla 4.1.

$$[Xpos_L, Ypos_L, Zpos_L, Potencia_L, R_{esf}, G_{esf}, B_{esf}, R_{pla}, G_{pla}, B_{pla}]$$

4.4.3. Aproximación de parámetros por PSO

A continuación se muestra el resultado de la aproximación de los parámetros por medio del algoritmo de "rendering in the loop", usando como técnica de optimización un algoritmo PSO.

El conjunto de parámetros de PSO usados en esta sección se obtuvo de forma experimental y fueron los que obtuvieron los mejores resultados. Se realizaron

Tabla 4.1: Codificación de parámetros para el modelo de color RGB

Parámetro	Descripción
X_Pos_L	Coordenada x de la fuente
Y_Pos_L	Coordenada y de la fuente
Z_Pos_L	Coordenada z de la fuente
Potencia_L	Intensidad de la fuente
R_esf	Componente rojo de color de la esfera
G_esf	Componente verde de color de la esfera
B_esf	Componente azul de color de la esfera
R_pla	Componente rojo de color del plano
G_pla	Componente verde de color del plano
B_pla	Componente azul de color del plano

10 corridas del algoritmo, con un criterio de parada de 200 iteraciones, con 25 partículas y vecindarios de 5 partículas por vecindario. Con una velocidad máxima de 0.08 y una velocidad mínima de -0.08. A continuación se muestran los resultados de una corrida, figura 4.9.

Figura 4.9: Evolución de las imágenes generadas, se presenta la mejor imagen de cada 5 generaciones, se puede apreciar el número de cada iteración, después de las letras PSOR

4.4.4. Evolución diferencial

Debido a que con PSO no se obtuvieron los resultados esperados, se decidió probar con un algoritmo de evolución diferencial. El algoritmo de evolución diferencial se implementó de la siguiente manera:

```
ImagenO = Obtención de  Imagen
   Iniciar individuos
   Para cada individuo
      Para cada Variable
         Selecciona un valor aleatorio entre 0.0 y 1.0
       Mientras (el criterio de parada no se cumpla)
       Para individuo=1 a número de individuos
         Calcula Aptitud
          Usa variables del individuo como parámetros del trazador
             Ejecuta trazador de rayos y genera imagenT
             Compara ImagenO con ImagenT
          Calcula aptitud de hijo
              Si Aptitud de (Xp > Padre)
                 Sustituye Hijo por Padre
              Calcula Mejor individuo
        Fin de ciclo de individuo
      Próxima Generacion hasta criterio de parada
```

El uso de Evolución Diferencial permitió obtener mejores parámetros que los obtenidos por medio de PSO. Por lo cual el desarrollo de las etapas siguientes se realizó usando a Evolución Diferencial como algoritmo bioinspirado. Se mejoró en los parámetros de la fuente de iluminación (posición X, Y y Z, así como en la intensidad), asimismo el color del objeto, se acercó al color del objetivo, figura 4.10.

4.4.5. Separación de tareas de optimización

Debido a la complejidad del problema, y a los resultados obtenidos hasta el momento, se decidió dividir la optimización de parámetros en dos partes, en la primera fase se realiza la optimización de los parámetros de la fuente de iluminación (posición x, y, z, así como la intensidad) y en segundo lugar los parámetros de los objetos color, índice de reflexión, índice de difracción, e índice de reflexión especular, 4.11.

4.4.6. Modelo de color HSV

Se implantó una función para usar los valores obtenidos del individuo como componentes de color HSV en lugar de usarlos como componentes RGB, las pruebas iniciales mostraron una ligera mejoría en el color de los objetos obtenidos,

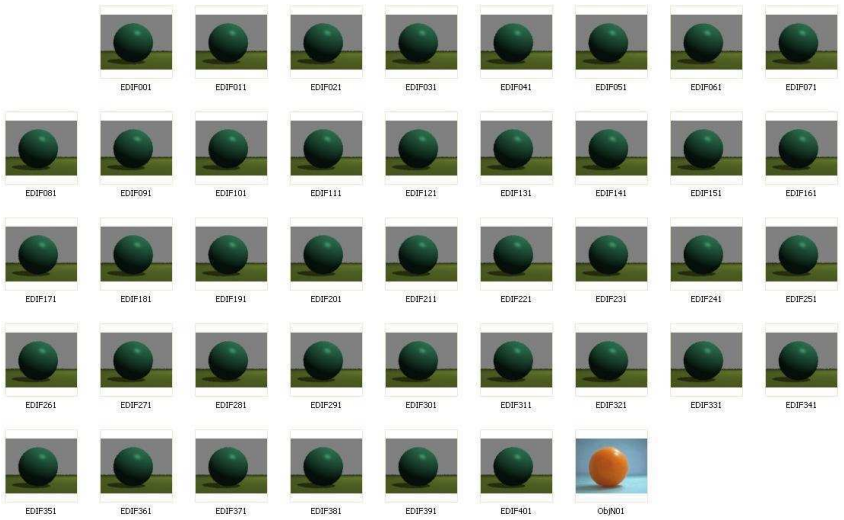


Figura 4.10: Resultados obtenidos por evolución diferencial, un mejor conjunto de parámetros para la fuente de iluminación y una mejora respecto al color del objeto

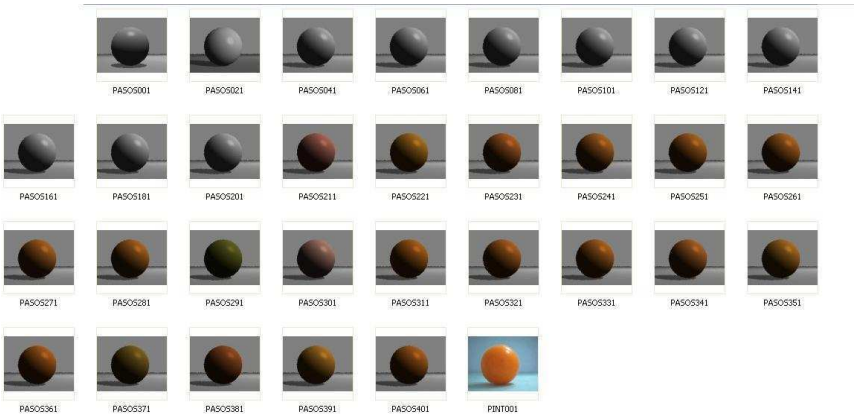


Figura 4.11: Separación de tareas de optimización, la primera etapa es la optimización de los parámetros de la fuente de iluminación, la segunda son los parámetros de los objetos, como el color y los índices de reflexión y difracción.

figura 4.12. La adecuación que se realizó fue en la etapa de la obtención de los datos del individuo de evolución diferencial como parámetros de los objetos en el algoritmo de traza de rayos. Al leer los parámetros del objetos y leerlos como valores RGB, se modificó la función de renderizado para que los valores leídos de cada individuo se trataran como parámetros HSV, después se pasan a una función llamada HSVaRGB la cual los convierte en valores RGB que ya son usados

por el algoritmo de traza de rayos, por lo tanto la codificación del individuo se incorpora en un arreglo donde cada elemento es un valor real, tabla 4.2.

Tabla 4.2: Codificación de parámetros para el modelo de color HSV

Parámetro	Descripción
X_Pos_L	Coordenada x de la fuente
Y_Pos_L	Coordenada y de la fuente
Z_Pos_L	Coordenada z de la fuente
Potencia_L	Intensidad de la fuente
H_esf	Componente tono de color de la esfera
S_esf	Componente saturación de color de la esfera
V_esf	Componente valor de color de la esfera
H_pla	Componente tono de color del plano
S_pla	Componente saturación de color del plano
V_pla	Componente valor de color del plano

Se puede apreciar una ligera mejora en el color del objeto central, esfera, con respecto a lo obtenido al usar el modelo RGB, figura 4.12.

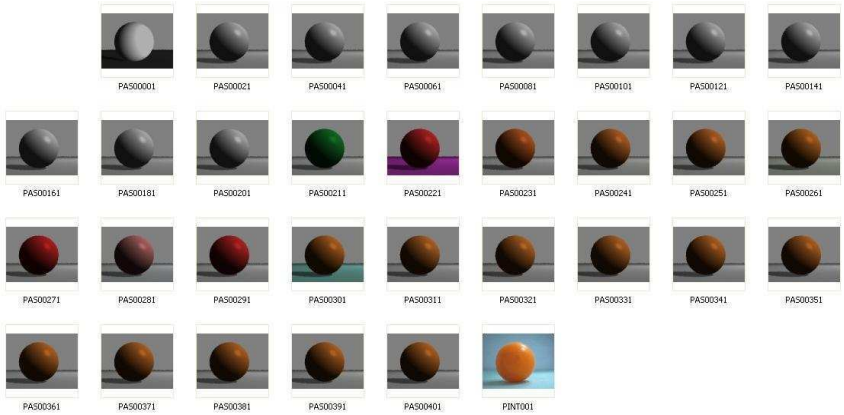


Figura 4.12: Ejemplo de evolución usando modelo de color HSV

Se uso del modelo de color HSV, como una posibilidad de mejora a los resultados de las búsquedas de los individuos. Esto se justifica si se toma en cuenta el espacio de búsqueda de un modelo RGB que puede ser conceptualizado como un cubo el cual tiene algunos puntos que pueden ser de difícil acceso para los individuos de ED, tal como las esquinas del cubo, figura 4.13.

En el caso del modelo HSV, cuyo espacio de búsqueda se conceptualiza como un cílindro o un cono, figura 4.14, se tienen menos esquinas que en el caso del

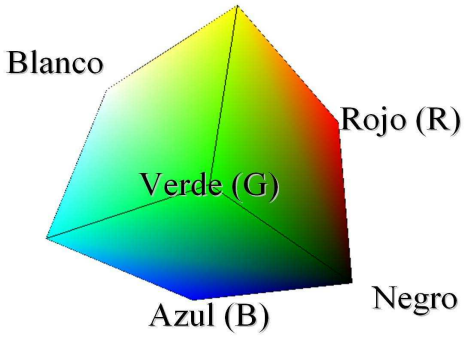


Figura 4.13: Espacio de color RGB

espacio RGB.

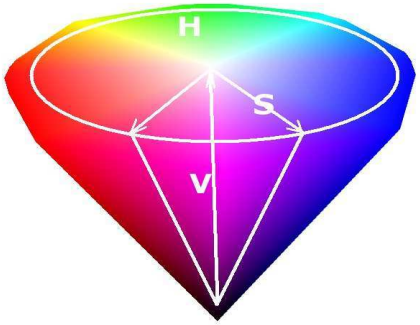


Figura 4.14: Espacio de color HSV, se conceptualiza como una forma cónica

El uso del modelo HSV mejoró el tono del color de la esfera con respecto a los modelos RGB usados con anterioridad. Por lo cual es lógico pensar que a la Evolución Diferencial le benefició el espacio de búsqueda HSV ya que pudo realizar una mejor exploración de parámetros.Para una información adicional de los modelos de color se puede revisar el apéndice C.

4.4.7. Modelo de iluminación mejorado

Debido a que el modelo de iluminación usado en un principio es un modelo simplificado basado en el modelo de Phong [72], el resultado de las búsquedas no encontró el resultado esperado así que se incorporó un modelo más comple-

to el cual adicionalmente del color del objeto, el índice de reflexión, el índice de refracción y el índice de reflexión especular, se agrega un factor de distancia con respecto a la superficie y la dirección de la cual viene la luz, de forma tal que se generan reflexiones de diferentes intensidades reflejadas en dirección del observador, es el modelo Phong-Blinn, [8]. Se considero necesaria una modificación del modelo de iluminación, debido a lo limitado de la representación final de la imagen final que logra el modelo anterior. La aplicación de dicho modelo mejoró notablemente la apariencia final de la imagen con respecto a los intentos anteriores, figura 4.15. Con lo cual se comprobó que a medida que mejora el modelo de iluminación, se obtiene una imagen con una diferencia menor respecto a la imagen original.

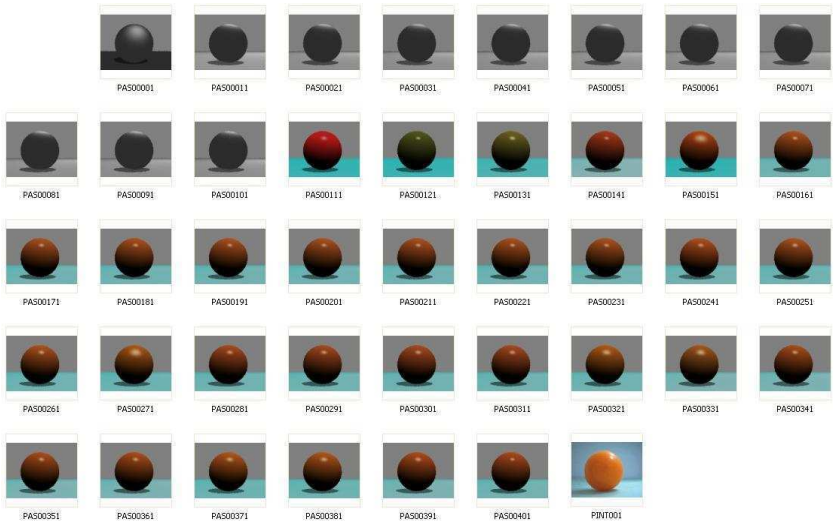


Figura 4.15: Modelo de iluminación mejorado, se puede apreciar la variación del tamaño de la reflexión especular

4.4.8. Corrección gamma

Para mejorar el aspecto de la imagen generada, se agregó una corrección gamma a la parte de la traza de rayos, esta corrección gamma ayuda a que la imagen se vea más brillante y proporciona un control sobre el contraste de las imágenes generadas, figura 4.16.

Es posible hacer que el algoritmo encuentre una corrección gamma para la renderización, pero no es un parámetro propio de los objetos, por lo tanto se buscó un parámetro de corrección gamma que entrega buenos resultados y se

dejó fijo.

Figura 4.16: Tanto el color del suelo como el objeto se ven con más brillo que en las partes anteriores, la corrección gamma empleada es de 1.2 para todas las componentes

4.4.9. Aproximaciones sucesivas

Una vez que los algoritmos de evolución diferencial obtuvieron una buena solución, se realiza una búsqueda en el rango próximo de los valores de cada componente de color para aproximar más los parámetros para la imagen final, figura 4.17.

El objetivo fue que la evolución diferencial buscará buenos valores y a través de una búsqueda con 10 valores hacia arriba y 10 valores hacia abajo del valor encontrado, se realizará una evaluación y comparación del resultado de estos valores, si los parámetros propuestos son mejores que el mejor individuo de la Evolución Diferencial, entonces se sustituyen los parámetros propuestos en el mejor individuo de la Evolución Diferencial y se renderiza la imagen.

4.4.10. Puntos de interés

Ya que la segunda función de evaluación requiere de n puntos de interés, los cuales deben ser fijados por el usuario, estos puntos son muy importantes para la obtención de la imagen final. Debido a esto se planteó una estrategia, en la cual los puntos de interés también sean buscados por una heurística. Por lo tanto se propuso que en un algoritmo PSO cada partícula entregue un conjunto de 10 puntos de interés y que cada propuesta ejecute un algoritmo de evolución

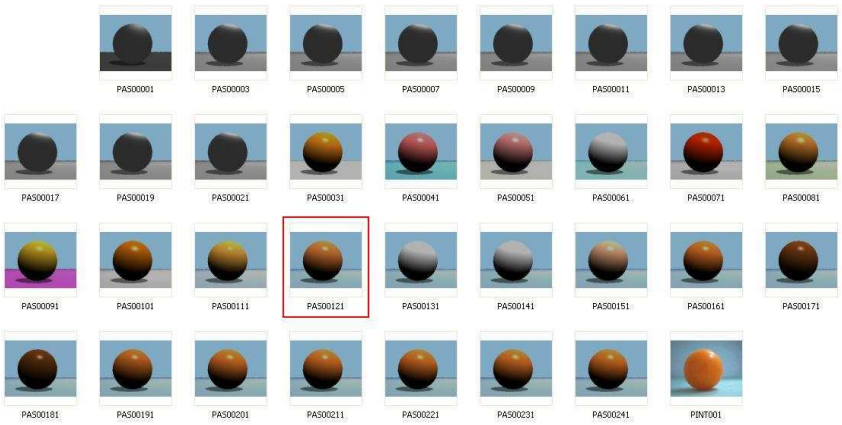


Figura 4.17: La Evolución Diferencial funcionó desde la figura PAS00000 hasta la PASS00121, recuadro rojo, a partir de la imagen 131 se realizó una búsqueda con una variación en los parámetros de color para cada componente, con ±10

diferencial con 100 generaciones, y el mejor individuo obtenga un valor de aptitud basado en la proximidad con la imagen de referencia, el objetivo es encontrar la mejor disposición de puntos de interés en la imagen para obtener las mejores imágenes posibles. Las imágenes generadas obtendrán una mejor aptitud con mejores puntos de interés, esta aptitud pasa a la partícula de PSO, y después se prueba la siguiente partícula. Se puede apreciar como en la evolución de las partículas, los individuos van eligiendo los puntos donde existen más cambios en el objeto central de la imagen, figura 4.18.

De esta forma hasta terminar con todas las partículas de la iteración, durante 90 iteraciones, evidentemente los mejores puntos de interés van a permanecer. Se puede ver la distribución inicial que se tiene al principio de la optimización, es una distribución aleatoria, figura 4.19. Y la distribución final, más organizada que se tiene en la iteración 100, figura 4.20.

Figura 4.18: Puntos de interés encontrados por la metaheurística Optimización por Cúmulo de Partículas -Evolución Diferencial

Figura 4.19: Puntos de interés encontrados por la mejor partícula en la generación inicial

Figura 4.20: Puntos de interés encontrados por mejor partícula de la generación 100

Capítulo 5

Implantación en la arquitectura paralela

5.1. Introducción

En el presente capítulo se describe el proceso de la implantación de la metodología de extracción de parámetros en la arquitectura del GPU. Primero se realiza una implantación de un algoritmo PSO y se evalúa el desempeño de sus diferentes variantes para la implantación paralela. Posteriormente se hace un comparativo de la implantación de un algoritmo PSO y de un algoritmo de Evolución Diferencial y se observa y reporta su comportamiento en el GPU. Finalmente, en base a los resultados de los experimentos anteriores se implanta la metodología propuesta en el GPU.

5.2. Estudio comparativo de implantaciones paralelas en una arquitectura multihilos

Los algoritmos Optimización por Cúmulo de Particulas, PSO (Particle Swarm Optimization), han tenido gran aceptación dentro de las alternativas de optimización global basadas en heurísticas bio-inspiradas. Sus principales ventajas son su simplicidad de programación y el mínimo de parámetros requeridos para su operación. En general, las técnicas heurísticas han tenido un gran auge en los últimos veinte años gracias al impresionante poder de cálculo desarrollado por las computadoras personales en el mismo periodo. Sin embargo, aún resulta atractivo estudiar alternativas tecnológicas que permitan acelerar la velocidad de cómputo de estos algoritmos para aplicarlos a problemas mucho más grandes y complejos. En este trabajo se presenta un estudio empírico sobre la implantación

de algunas variantes paralelas de un algoritmo PSO empleando un dispositivo de procesamiento gráfico, GPU, con capacidad multi-hilos y el más reciente modelo de programación paralela para estos casos.

5.2.1. Introducción

Las técnicas heurísticas bio-inspiradas, tales como el cómputo evolutivo, colonia de hormigas y PSO, se propusieron como alternativas para resolver problemas de optimización difíciles obteniendo soluciones aceptablemente buenas en un tiempo razonable. Estas técnicas, al ser poblacionales, ensayan diferentes soluciones con base a ciertas reglas y al uso extensivo de gran cantidad de procesos estocásticos subyacentes. Estas herramientas han encontrado aplicación en prácticamente todos los campos del conocimiento, gracias a la disponibilidad de la velocidad y poder de cálculo de las actuales computadoras personales. Las técnicas heurísticas obtienen soluciones aceptablemente buenas en un tiempo "razonablemente corto", en comparación con los métodos determinísticos y enumerativos tradicionales que exploran cada una de las posibilidades y que resultan imprácticos en los problemas de optimización difíciles. Sin embargo, el tiempo "razonable" que pueden consumir las técnicas heurísticas es de varios segundos, minutos, e incluso horas, dependiendo del problema y de la técnica heurística que se emplea. En este sentido, las técnicas como colonia de hormigas y estrategias evolutivas pueden llegar a ser especialmente demandantes de recursos computacionales. Por ello, los algoritmos heurísticos simples, como lo es el de optimización con cúmulo de partículas, PSO, se han vuelto muy atractivos en virtud de que su relativa simplicidad se traduce en un tiempo de ejecución proporcionalmente corto. No obstante, cuando se requiere obtener una buena solución en "tiempo real", incluso los algoritmos simples pueden resultar lentos. Todo lo anterior ha motivado la búsqueda de nuevas formas que aceleren el desempeño las técnicas heurísticas.

Recientemente se consideró la posibilidad de aprovechar el gran poder computacional disponible en las tarjetas gráficas, GPUs, para resolver problemas de propósito general [23], concibiendo con ello el concepto del Cómputo de Propósito General en Unidades de Procesamiento Gráfico. Desde entonces, tanto fabricantes como desarrolladores han considerado esta nueva área de la computación como una vertiente con un prometedor futuro, en función de la amplia gama de posibles aplicaciones que pueden obtener una ventaja de la natural paralelización de operaciones de los GPU.

En este trabajo se presentan las principales consideraciones de implantación, así como los resultados obtenidos, durante la paralelización de un algoritmo PSO empleando un nuevo enfoque de programación de GPU basado más en el empleo de múltiples hilos que en el tradicional empleo de las operaciones paralelas sobre

la memoria de despliegue. Por esta razón se seleccionó el dispositivo NVIDIA GeForce 8600GT, el cual cuenta con capacidad para administrar múltiples hilos concurrentes, además de la tradicional paralelización de operaciones, bajo el concepto SIMT (instrucción simple múltiples hilos) [65].

5.2.2. Trabajo relacionado

Desde la aparición de los primeros algoritmos poblacionales y bio-inspirados pareció natural su traslado hacia arquitecturas paralelas y distribuidas, sólo las limitaciones tecnológicas de los primeros tiempos acotaron el avance en este sentido. El algoritmo PSO no fue la excepción y poco después de su aparición se presentaron los primeros intentos por explotar su natural propensión al paralelismo, como en [24], interés que se mantiene vigente hasta el día de hoy a fin de aplicar variantes paralelizadas del algoritmo PSO para resolver problemas de optimización muy complejos (ver, por ejemplo, [50], [17] y [61]).

Respecto a la forma de paralelizar al algoritmo PSO, en la comunidad del cómputo evolutivo existe el consenso de que, siendo ambas técnicas poblacionales, la clasificación de las variantes paralelas para los algoritmos evolutivos aplica igualmente para los algoritmos PSO [6], a saber:

- Implantaciones paralelas con el enfoque global. Donde existe un procesador maestro y varios esclavos, de tal manera que el maestro distribuye en los esclavos principalmente el trabajo relativo a la evaluación de la aptitud, ya que tradicionalmente se ha considerado que las secciones con cálculos consumen la mayor parte del tiempo de procesador.

- Implantaciones paralelas con enfoque migratorio o de islas. En este enfoque, los diferentes procesadores corren instancias del algoritmo con sus propias poblaciones en forma independiente durante cierto tiempo, después del cual pasan a una etapa de comunicación e intercambio de información donde comparte las mejores soluciones encontradas hasta ese momento. El proceso se repite varias veces hasta cumplir un criterio de terminación.

- Implantaciones paralelas con enfoque difuminado. Puede verse como un caso límite del modelo de islas, en donde la población de cada isla es uno y en donde existen tantos procesadores como individuos.

Las implantaciones paralelas tradicionalmente se han concebido para ejecutarse en sistemas distribuidos, en donde los procesadores forman una red y en donde el tiempo de sobrecarga acumulado (overhead) por concepto de comunicación es un factor que impacta considerablemente el desempeño de la implantación paralela. Por ello no es raro que el análisis de las implantaciones PSO paralelas usualmente se haga explícitamente en función de estrategias de comunicación como en [13],

[9] y [63]. Otras propuestas para paralelizar los algoritmos PSO se han hecho con base al concepto de procesamiento vectorial paralelo, como en [70] y en [77], pero también se han hecho propuestas con base a procesos concurrentes en un solo procesador como en [5].

Aunque en la literatura existe una bibliografía significativa de esfuerzos enfocados en paralelizar algoritmos de cómputo evolutivo, sólo algunos se enfocan sobre la paralelización de algoritmos PSO. Lo anterior probablemente se deba a la posterior y relativamente reciente aparición de los algoritmos PSO, pero lo cierto es que similar situación se presenta en la literatura referente a la programación del algoritmo PSO en GPUs. Por ejemplo, tenemos en [43], [42], [58] y [73] trabajos muy recientes en que tratan sobre la implantación paralela de algoritmos genéticos y evolutivos en GPUs, mientras que solo en [22] los autores aplican, para el caso PSO, su experiencia con previas implantaciones para genéticos en GPUs.

Nuestras implantaciones del PSO paralelo para GPU difieren enormemente de la presentada en [22], en virtud de que en nuestro caso no empleamos el tradicional enfoque de programación del GPU basado en operaciones paralelas sobre múltiples datos de la memoria gráfica (texturas, etc.), que se conoce con el termino SIMD (instrucción simple múltiples datos), sino en un nuevo modelo de programación paralela concebido para aprovechar la capacidad de algunos GPU para administrar múltiples hilos concurrentes en forma muy eficiente, además del tradicional paralelismo de operaciones sobre memoria, en lo que se conoce como SIMT (instrucción simple múltiples hilos) [65]. Este nuevo enfoque y las herramientas de programación asociadas permiten hacer programación paralela en los dispositivos GPUs de una manera más natural y para aplicaciones de verdadero propósito general. De hecho, se prevé que este nuevo modelo de programación paralela sea el referente para la eventual especificación de un modelo de programación universal que permita hacer programación paralela, no sólo en GPUs, sino en cualquier arquitectura multi-núcleo o plataforma de súper cómputo que aparezca en el futuro.

Por ahora, hasta donde sabemos, no existe en la literatura un estudio empírico relativo a la implantación de algoritmos heurísticos bio-inspirados en un GPU con este nuevo modelo de programación paralela orientada a múltiples hilos. Esto lo atribuímos al hecho de que las herramientas de programación son de reciente introducción [39]. Por lo tanto, este trabajo intenta ser un primer estudio empírico que muestre el efecto sobre el desempeño de un algoritmo poblacional, en particular un PSO, programándolo en forma paralela en un GPU con un enfoque basado en multi-hilos.

5.2.3. Implantación de variantes del algoritmo PSO paralelo en el GPU

Se seleccionó al dispositivo NVIDIA GeForce 8600GT, a fin de implantar el algoritmo PSO con programación paralela en un GPU con soporte multi-hilos, debido a que el fabricante proporciona gratuitamente el software CUDA (Compute Unified Device Architecture, Arquitectura Unificada de Dispositivo de Cómputo) [67] que es precisamente el entorno para poder programar con el nuevo modelo de programación paralela. El objetivo era realizar una implantación paralela en un GPU sin recurrir al tradicional y laborioso método basado únicamente en operaciones paralelas sobre memoria gráfica. Lo anterior a fin de probar el rendimiento que podía alcanzarse con el GPU mediante una programación paralela relativamente simple centrada en distribuir el trabajo en múltiples hilos.

Así se decidió realizar la implantación de algunas variantes paralelas del algoritmo PSO canónico, y medir su desempeño en función del número de partículas y del número de iteraciones. Las implantaciones, a su vez, serían probadas para la obtención del óptimo global de tres conocidas funciones de referencia:

F05- Función Rosenbrock generalizada:

$$f_5(\vec{x}) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{5.1}$$
$$-30 < x_i < 30$$
$$\text{mín}(f_5) = f_5(1, \ldots, 1) = 0$$

F09- Función Rastrigin's generalizada:

$$f_9(\vec{x}) = \sum_{i=1}^{n-1}[x_i^2 - 10cos(2\pi x_i) + 10] \tag{5.2}$$
$$-5{,}12 < x_i < 5{,}12$$
$$\text{mín}(f_9) = f_9(0, \ldots, 0) = 0$$

F11- Función Griewank's generalizada:

$$f_{11}(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{5.3}$$
$$-600 < x_i < 600$$
$$\text{mín}(f_{11}) = f_{11}(0, \ldots, 0) = 0$$

Tabla 5.1: Proporciones del costo computacional en función del número de partículas

30 dimensiones, 20 vecinos, 2000 iteraciones						
	Partículas					
	64 partículas		256 partículas		1024 partículas	
Función	Evalúa	Actualiza	Evalúa	Actualiza	Evalúa	Actualiza
F05	2.84 %	96.59 %	3.03 %	97.25 %	2.03 %	97.28 %
F09	27.54 %	70.34 %	26.93 %	69.38 %	27.16 %	69.43 %
F11	39.86 %	55.24 %	41.31 %	55.81 %	14.88 %	35.93 %

En particular se emplearon estas funciones con suficientes dimensiones (más de 30) a fin de probar el desempeño de las implantaciones para problemas con complejidad significativa. La distribución del algoritmo en forma paralela tomó como referencia los modelos de programación paralela más conocidos: global, islas y difuminado. Para la implantación con el enfoque global se programaron dos variantes:

- Global_ev: Con la distribución al GPU de únicamente la evaluación de la función objetivo.
- Global_ev+up: Con la distribución al GPU de todos los cálculos, tanto de evaluación de la función objetivo como de actualización de velocidad, posición e inercia.

Lo anterior se hizo para mostrar que, a pesar de que tradicionalmente se hace especial énfasis en el hecho de distribuir los cálculos referentes a la función objetivo, cuando los problemas son n-dimensionales la evaluación de la aptitud puede llegar a representar una menor proporción del tiempo de procesamiento, respecto de los bloques de actualización de velocidad y posición (ver cuadros 5.1 y 5.2). Esto se debe a que las operaciones de actualización, aunque de menor complejidad aritmética, se vuelven de un volumen mayor respecto de las operaciones de evaluación de la aptitud. En particular, la obtención de los números aleatorios puede llegar a ser muy demandante de tiempo de procesamiento.

Respecto de las implantaciones con modelo paralelo de islas y difuminado, al considerar que el modelo difuminado es un caso límite del de islas, optamos por englobar estas últimas variantes bajo el término embebido (embedded), ya que en cualquiera de ellas todo el algoritmo PSO corre en la tarjeta GPU.

La estrategia de programación paralela para todas las implantaciones consistió en crear un hilo por cada partícula, a fin de poder distribuir las tareas del algoritmo PSO. La táctica para obtener la versión paralela del código consistió en remplazar todo lazo que estuviera en términos del número de partículas por una

Tabla 5.2: Proporciones del costo computacional en función del número de dimensiones

256 partículas, 20 vecinos, 2000 iteraciones						
	Dimensiones					
	30 dimensiones		60 dimensiones		120 dimensiones	
Función	Evalúa	Actualiza	Evalúa	Actualiza	Evalúa	Actualiza
F05	3.03 %	97.25 %	3.26 %	96.74 %	3.38 %	96.66 %
F09	26.93 %	69.38 %	30.51 %	67.81 %	33.56 %	66.53 %
F11	41.31 %	55.81 %	44.9 %	53.55 %	46.95 %	49.49 %

llamada a un kernel multi-hilo. Adicionalmente, se implantó el algoritmo PSO secuencial como referencia para el cálculo de las métricas de desempeño. Se pueden apreciar los siguientes bloques funcionales [57]:

- Inicialización de población.
- Evaluación de la aptitud mediante el cómputo de la función objetivo.
- Comparación para determinar si un individuo tiene mejor aptitud que el registrado como el mejor, ya sea local o global.
- Imitación, donde cada individuo actualiza su posición, influenciado por su propia experiencia y por el entorno social.

En la versión de PSO Secuencial todos los bloques funcionales son ejecutados en el procesador anfitrión. En la versión PSO Global_ev, lo que se distribuye hacia el GPU es el bloque de evaluación de la aptitud, remplazando el lazo correspondiente por una llamada a una función kernel que distribuye el trabajo a múltiples hilos dentro de la tarjeta del dispositivo GPU. En la versión PSO Global_ev+up, se distribuye hacia el GPU toda operación que implique cálculos, de tal manera que tanto el bloque de evaluación de la aptitud como el de actualización de la posición se paralelizan mediante las correspondientes llamadas a funciones kernel multi-hilos. Finalmente en la versión PSO Embebida, el único bloque funcional que queda en poder del procesador central es el de inicialización de la población, ya que se llama a una función kernel que ejecuta las tareas asociadas con la evaluación, la comparación y la imitación, hasta cumplir con el criterio de terminación.

Para concluir con esta sección, sólo resta mencionar algunas consideraciones prácticas que deben tomarse en cuenta para lograr una implantación funcional de un algoritmo PSO paralelo en un GPU:

- Tiempo de sobrecarga (overhead). En el GPU también existe un tiempo adicional acumulado (overhead) debido a los traslados de memoria entre el

anfitrión y el dispositivo GPU, lo cual es necesario para el intercambio de datos. Debido a que estos traslados son relativamente lentos, el diseño para una implantación paralela en un GPU debe minimizar su empleo. Por lo antes dicho, es de esperar que las implantaciones PSO globales sean más lentas que la embebida debido al mayor intercambio de datos entre el anfitrión y el GPU durante todo el algoritmo, ya que para que el procesador central pueda realizar las comparaciones requiere conocer los datos producidos por las secciones de evaluación y de actualización.

- Sincronización. Todos los hilos que estén ejecutándose deben de sincronizarse justo antes de continuar con los puntos donde se requiere que toda la información esté lista para la toma de alguna decisión. Este punto es particularmente importante cuando los hilos tienen que comunicarse entre ellos para compartir información, en donde lo ideal, además de sincronizar los hilos, es implantar una estrategia de comunicación maestro-esclavo, de tal manera que sólo uno de los hilos sea el que actualiza variables globales con base a la información de todos los demás.

- Contención. Hay que tener cuidado con el manejo de variables globales que pueden ser actualizadas y leídas por varios hilos a la vez, incorporando las técnicas que resuelvan este problema. Por ejemplo, en el algoritmo PSO esto ocurre con la variable que contiene el índice a la mejor solución global hasta el momento.

- Generación de números aleatorios. Este es un punto fundamental para el buen funcionamiento del algoritmo PSO y se vuelve crítico cuando se trata de generar los aleatorios al interior del GPU. Una implantación que asuma una generación de aleatorios de buena calidad en el GPU (diferentes en cada llamada rand() y para cada hilo) tomando las medidas pertinentes, puede repercutir en un algoritmo que no converja debido a la falta de diversidad. Por ello, en nuestra implantación del PSO embebida el bloque de inicialización se mantiene fuera del GPU y, en especial, la generación de semillas para los aleatorios, que en nuestro caso es de una semilla por hilo.

5.2.4. Experimentos y resultados

Métricas para el desempeño del procesamiento paralelo

Para evaluar el desempeño de las diferentes variantes del algoritmo PSO paralelo, se definen algunas métricas, usadas comunmente para medir el desempeño en algoritmos paralelos, las cuales nos son más que medidas para comparar el desempeño entre las variantes:

- Costo computacional

- Rapidez
- Ventaja (speedup)
- Eficiencia

El costo computacional, C, lo definimos como el tiempo de procesamiento consumido (en segundos) que le toma a cada variante ejecutar el algoritmo PSO, con los parámetros de prueba dados (iteraciones, partículas, dimensiones etc.). Definimos la rapidez, V, como el inverso del costo computacional:

$$V = \frac{1}{C} \tag{5.4}$$

La ventaja (speedup) es la tasa que resulta de dividir la rapidez de la variante de interés entre la rapidez de la variante de referencia (por ejemplo, la variante secuencial).

$$S = \frac{V_{obj}}{V_{ref}} \tag{5.5}$$

Finalmente, pero no menos importante, definimos a la eficiencia paralela, E, como la tasa que resulta de dividir la ventaja obtenida entre el número de unidades de procesamiento:

$$E = \frac{S}{n} \tag{5.6}$$

donde n es igual al número de núcleos de procesamiento del GPU (en nuestro caso 32).

5.2.5. Desarrollo experimental

Las pruebas se realizaron en el sistema mencionado en la sección 3.6.1.

Se llevaron a cabo una serie de experimentos para medir el desempeño de cada una de las variantes implantadas del algoritmo PSO paralelo, a saber:

1. Variante secuencial. Un PSO que se ejecuta totalmente en el procesador del anfitrión.
2. Variante Global_ev. Ejecuta sólo la función de evaluación en el GPU y el resto del algoritmo lo ejecuta en el anfitrión.
3. Variante Global_ev+up. Un PSO que ejecuta la evaluación de aptitud y la actualización de la velocidad en el GPU, mientras que el resto del algoritmo se ejecuta en el anfitrión.
4. Variante Embebida. Es un programa PSO donde se han delegado al GPU la mayoría de los bloques funcionales del algoritmo PSO, dejando al anfitrión únicamente la inicialización de la población y el despliegue de los resultados.

Tabla 5.3: Comparativo de referencia para tres funciones clásicas

10,000 iteraciones, 256 partículas, y 30 dimensiones						
	Función					
	F05		F09		F11	
Variante PSO	óptimo	Tiempo segs.	óptimo	Tiempo segs.	óptimo	Tiempo segs.
Secuencial	0.0198489	20.832	2.0894139	28.553	0.0009857	35.934
Global_ev	0.0023021	25.125	1.8904221	22.747	0.0663081	22.914
Global_ev+up	20.8692016	8.456	4.1788275	6.5	0.01083621	6.663
Embebida	31.7282912	7.568	5.2254372	5.958	0.2614562	6.186

Las funciones que se usaron fueron la F05, F09 y F11, ya que presentan una complejidad significativa al ser evaluadas [1]. El óptimo para todas las funciones es de 0.

El primer grupo de experimentos se realizó para medir el desempeño en términos de la complejidad de las funciones objetivo. Se fijaron los parámetros en 10,000 iteraciones, 256 partículas, y 30 dimensiones. Se realizaron 10 corridas de las 4 variantes del algoritmo PSO para cada una de las 3 funciones de evaluación, obteniendo el promedio del óptimo encontrado y del tiempo consumido en segundos. Los resultados se pueden apreciar en la tabla 5.3. A continuación se realizaron pruebas de desempeño de las implantaciones variando dos parámetros fundamentales: el número de partículas usadas y el número de iteraciones. Se reportó el error respecto del óptimo, así como el tiempo consumido en segundos con centésimas de segundos de precisión. Debido a las restricciones de espacio, y después de observar que la función F11 consume más tiempo de procesamiento en todos los casos, se optó por elegir a las corridas con F11 para ilustrar los resultados experimentales de este trabajo.

Para los experimentos donde se varió el número iteraciones se fijaron los siguientes parámetros:

- Partículas = 256
- Dimensiones = 60
- Función = F11
- Criterio de parada = número de iteraciones.
- Iteraciones = 10,000, 30,000 y 60,000

mientras que para aquellos experimentos donde se varió del número de partículas se fijaron los siguientes parámetros:

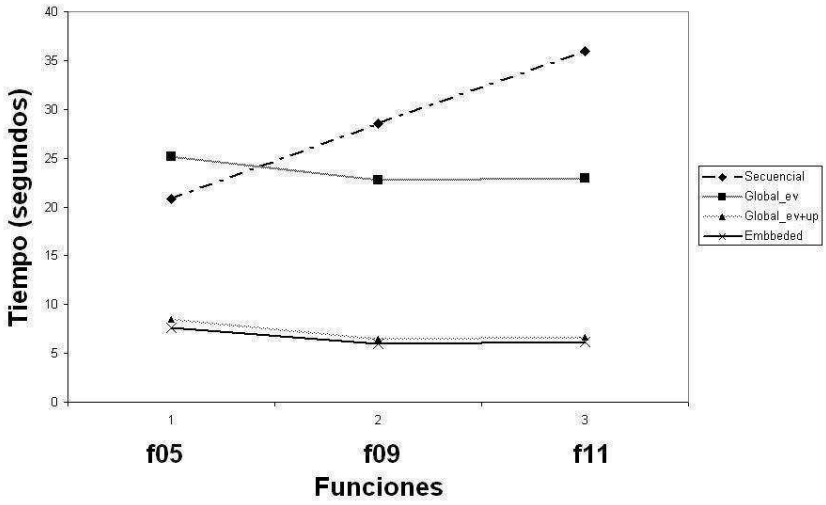


Figura 5.1: Comparativo del desempeño de las variantes PSO en función de la complejidad de la función objetivo

- Iteraciones = 10,000
- Dimensiones = 60
- Función = F11
- Criterio de parada = número de iteraciones.
- Partículas = 64, 256 y 1024

5.2.6. Discusión de resultados

En lo referente al tiempo consumido, experimentalmente se observó que el desempeño, de la variante Global_ev puede llegar a ser peor que el de la variante secuencial si la complejidad aritmética de la función objetivo es poca, pero a medida que esta complejidad aumenta, el desempeño de la variante Gloval_ev mejora en virtud de un tiempo de sobrecarga acumulado (overhead) proporcionalmente menor (ver fig 5.1). Los resultados muestran una tendencia que sugiere una mejora en tiempo, respecto a la ejecución en el anfitrión, para la implantación Global_ev durante la optimización de las funciones F09 y F11. La mejora en tiempo es más notable para la variante Global_ev+up, a pesar de tener resultados pobres para las funciones F05 y F09. La mayor mejora en tiempo, reducción, la logra el algoritmo PSO embebido, el cual tiene un tiempo de cálculo de 6 segundos

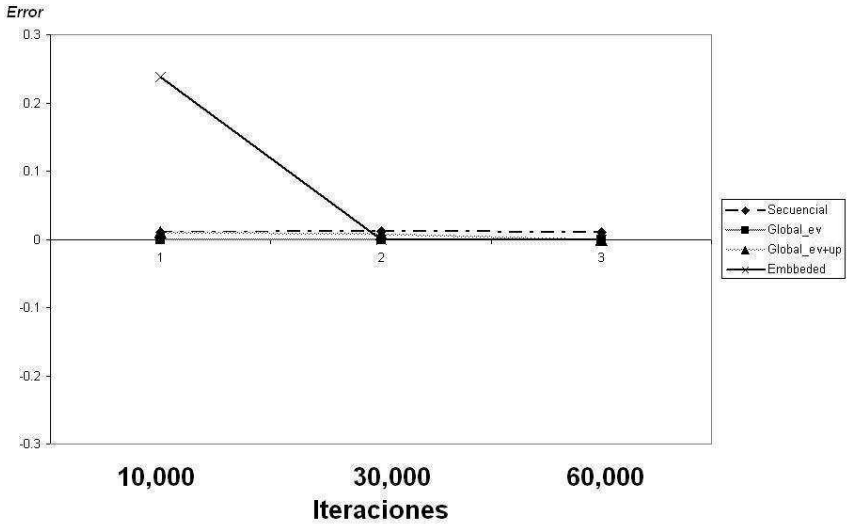


Figura 5.2: Comparativo del error de las variantes PSO, en función del número de iteraciones, para la optimización de F11

en promedio, sin embargo también muestra un desempeño pobre en relación con el algoritmo secuencial para el caso de las funciones F05 y F09. No obstante, la mejora en tiempo es notable: 35.9 seg. de la variante secuencial contra 6.6 seg. de la variante embebida.

Variando el número de las iteraciones se pudo obtener una buena aproximación al óptimo real en todas las implantaciones. Se observó que la implantación secuencial tiene un error bajo y muy uniforme, si se toma como punto de partida 10000 iteraciones. Por su parte, la variante embebida comienza con un error ligeramente mayor al de las demás implantaciones, pero a medida que se incrementa el número de iteraciones su rendimiento mejora y alcanza la misma precisión para 30 y 60 mil iteraciones (ver figura 5.2), lo cual logra en tan sólo una fracción de tiempo consumido por la implantación secuencial, para el mismo número de iteraciones. De hecho, a 60 mil iteraciones la variante embebida tiene un error menor que la secuencial en una razón de 5 a 1 (ver cuadro 5.4).

En la figura 5.3 se puede apreciar el incremento exponencial de tiempo consumido por el algoritmo secuencial a medida que se aumenta el número de iteraciones. Es clara la mejora adquirida a medida que se aumenta el número de operaciones matemáticas distribuidas en el GPU. También es interesante señalar la mejora que se tiene cuando el GPU, además de ejecutar los cálculos de la fun-

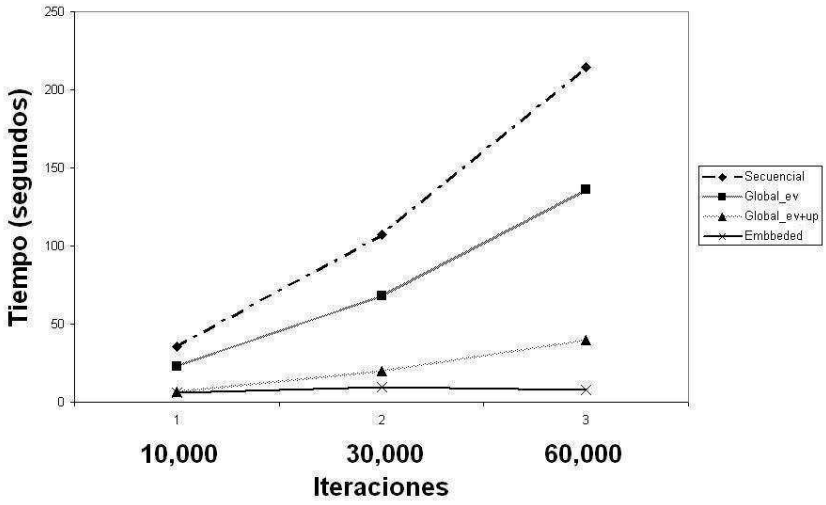


Figura 5.3: Comparativo del tiempo consumido por las variantes PSO, en función del número de iteraciones, para la optimización de F11

Tabla 5.4: Desempeño en función de iteraciones durante la optimización de F11

256 partículas, 60 dimensiones						
	Iteraciones					
	10,000		30,000		60,000	
Variante PSO	óptimo	Tiempo segs.	óptimo	Tiempo segs.	óptimo	Tiempo segs.
Secuencial	0.009865	35.83	0.012321	107.2	0.009857	214.31
Global_ev	0.000000	22.82	0.000000	68.2	0.000000	136.3
Global_ev+up	0.009876	6.67	0.007396	19.89	0.000000	39.89
Embebida	0.237734	6.08	0.000000	9.81	0.000000	7.66

ción objetivo(variante Global_ev), también realiza la actualización de velocidad y posición (variante Global_ev+up). En cuanto a la implantación embebida, es muy interesante observar que el tiempo consumido es prácticamente el mismo, independientemente del número de iteraciones.

Por su parte, los resultados obtenidos al variar el número de partículas mostraron un comportamiento similar (ver figura 5.4): a mayor número de partículas mejor el desempeño de las implantaciones en el GPU (ver Tabla 5.6).

En general, los experimentos mostraron que el desempeño de los algoritmos

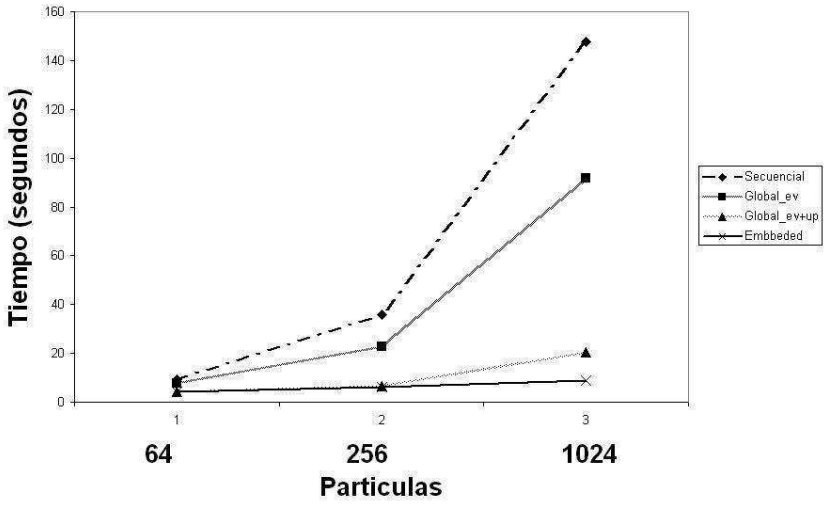


Figura 5.4: Comparativo del tiempo consumido por las variantes PSO, en función del número de partículas, para la optimización de F11

Tabla 5.5: Desempeño en función del número de partículas durante la optimización de F11

10,000 iteraciones, 60 dimensiones						
	Partículas					
	64		256		1024	
Variante PSO	óptimo	Tiempo segs.	óptimo	Tiempo segs.	óptimo	Tiempo segs.
Secuencial	0.012316	9.1	0.012316	35.78	0.000000	147.50
Global_ev	0.007396	7.54	0.0073.96	22.67	0.000000	91.68
Global_ev+up	0.007396	4.33	0.009865	6.66	0.000000	20.53
Embebida	0.238114	4.21	0.277701	6.20	0.000000	8.76

implantados adquiere una notable mejoría a medida que más cálculos son distribuidos para su ejecución en el dispositivo GPU, comportamiento que es prácticamente uniforme para todas las funciones probadas. Las métricas para el desempeño de las diferentes variantes se resumen en los cuadros 5.6 a 5.9 y sus gráficas corespondientes (ver fig 5.5 a fig 5.8), las cuales son elocuentes por sí mismas.

Resumiendo, de los resultados experimentales podemos destacar, en primer

Tabla 5.6: Ventaja de las variantes PSO paralelas, respecto de la variante secuencial, en función del número de iteraciones

F11, 256 partículas, 60 dimensiones			
	Iteraciones		
	10,000	30,000	60,000
Variante PSO	Ventaja	Ventaja	Ventaja
Global_ev	1.57011394	1.57184751	1.57234043
Global_ev+up	5.37181409	5.38964304	5.37252444
Embebida	5.89309211	10.9276249	27.9778068

Tabla 5.7: Ventaja de las variantes PSO paralelas, respecto de la variante secuencial, en función del número de partículas

F11, 10,000 iteraciones, 60 dimensiones			
	Partículas		
	64	256	1024
Variante PSO	Ventaja	Ventaja	Ventaja
Global_ev	1.20689655	1.57829731	1.60885689
Global_ev+up	2.10161663	5.37237237	7.18460789
Embebida	2.16152019	5.77096774	16.8378995

Tabla 5.8: Eficiencia de las variantes PSO paralelas, respecto de la variante secuencial, en función del número de iteraciones

F11, 256 partículas, 60 dimensiones			
	Iteraciones		
	10,000	30,000	60,000
Variante PSO	Eficiencia	Eficiencia	Eficiencia
Global_ev	0.04906606	0.04912023	0.04913564
Global_ev+up	0.16786919	0.16842634	0.16789139
Embebida	0.18415913	0.34148828	0.87430646

Tabla 5.9: Eficiencia de las variantes PSO paralelas, respecto de la variante secuencial, en función del número de partículas

F11, 10,000 iteraciones, 60 dimensiones			
	Partículas		
	64	256	1024
Variante PSO	Eficiencia	Eficiencia	Eficiencia
Global_ev	0.03771552	0.04932179	0.05027678
Global_ev+up	0.06567552	0.16788664	0.22451901
Embebida	0.06754751	0.18034274	0.52618436

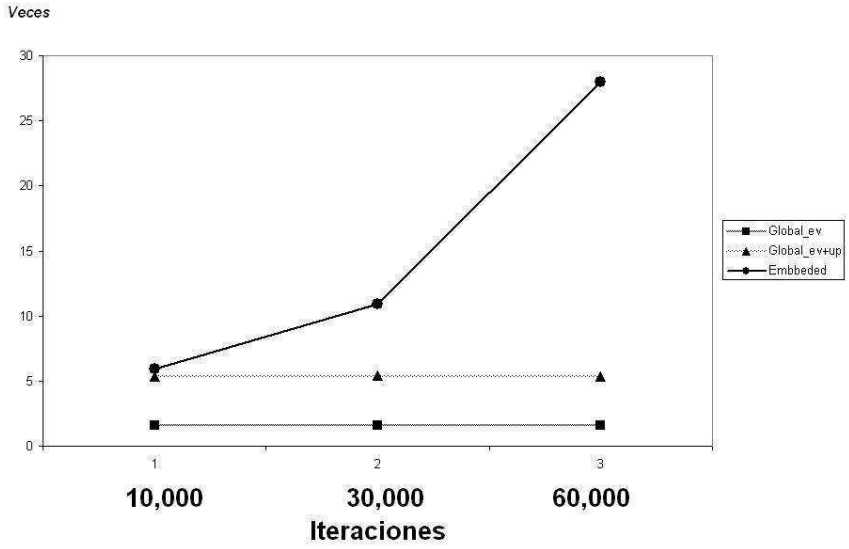


Figura 5.5: Comparativo de la ventaja de las variantes PSO paralelas en función del número de iteraciones

lugar, el buen desempeño que tiene un GPU en relación al procesador anfitrión. Por ejemplo, al optimizar la función Griewank's generalizada (F11, de alto costo computacional con 60 dimensiones y 60,000 iteraciones y 256 partículas), mientras la variante secuencial encontró el óptimo 0.0 en el transcurso de 214.13 segundos (tiempo que por sí solo demuestra la relativa rapidez del algoritmo PSO), la variante Gloval_ev consumió 136.3 segundos y la variante Gloval_ev+up consumió 39.89 segundos, una ventaja de 5.37 veces respecto a la variante secuencial. Finalmente, la variante embebida consumió únicamente 7.66 segundos, es decir una ventaja de 27.97 veces con respecto a la ejecución secuencial, que se traduce en una eficiencia de prácticamente 1 (recuérdese que en nuestro caso el GPU se integra por 32 núcleos).

Nótese como al aumentar el número de partículas, o las iteraciones, la eficiencia para las variantes Global_ev y Global_ev+up se mantiene prácticamente constante, aunque la métrica de ventaja aumente. Esto se debe a que el tiempo de sobrecarga acumulado (overhead) también aumenta en proporción directa al número de partículas o bien de iteraciones. En cambio, la variante embebida aumenta su eficiencia a medida que aumenta el número de las partículas o el número de iteraciones.

Los resultados experimentales también confirman lo siguiente:

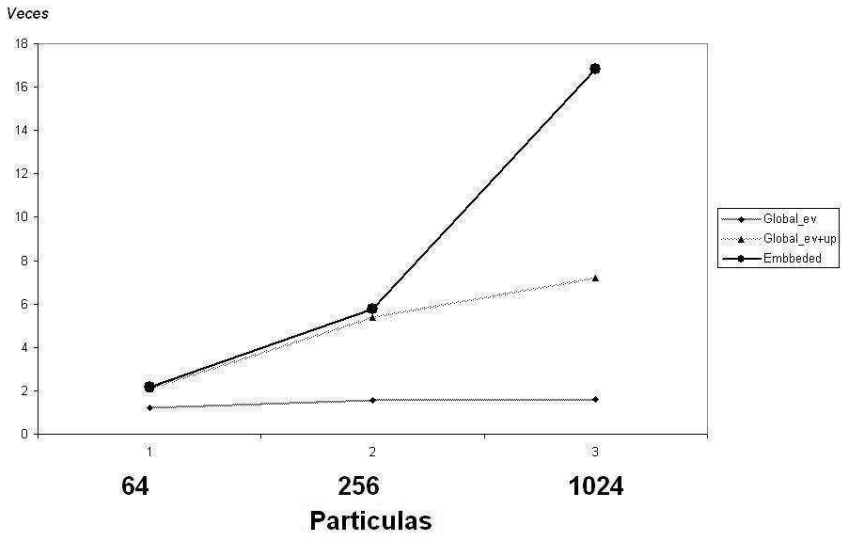


Figura 5.6: Comparativo de la ventaja de las variantes PSO paralelas en función del número de partículas

- En problemas de optimización n-dimensionales el mayor tiempo de procesamiento del algoritmo PSO lo consumen las tareas de actualización (posición y velocidad), y no la de evaluación de aptitud.

- La disminución en el desempeño para las variantes paralelas Gloval_ev y Global_ev+up, respecto de la variante embebida, se debe al tiempo de sobrecarga (overhead) acumulado durante los intercambios de datos entre el anfitrión y el GPU.

5.2.7. Análisis de resultados

En este trabajo se presentó un estudio empírico comparativo sobre el desempeño de tres variantes paralelas del algoritmo de optimización por cúmulo de partículas (PSO) en un GPU con soporte multi-hilos, empleando el más reciente modelo de programación paralela orientado a hilos. Los resultados mostraron que el algoritmo es paralelizable y funciona en un esquema de arquitectura de procesadores del tipo de un GPU.

En general, se demostró que un algoritmo paralelo programado simplemente con base a los hilos de un GPU (i.e. exclusivamente distribuyendo el trabajo en hilos y sin recurrir a un uso intensivo de las capacidades de paralelización de operaciones en múltiples datos), por lo menos, puede alcanzar una ventaja pro-

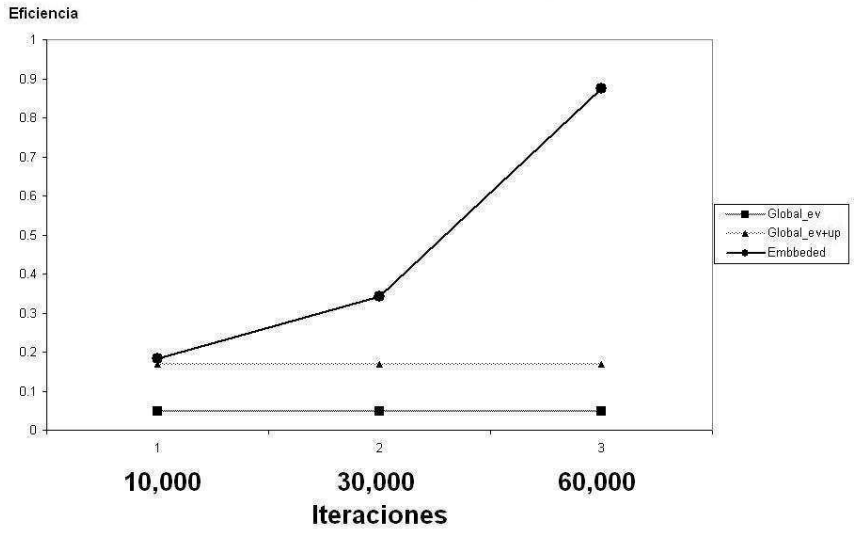


Figura 5.7: Comparativo de la eficiencia de las variantes PSO paralelas en función del número de iteraciones

porcional al número de núcleos del GPU, con lo cual se tiene el poder de cómputo de un *cluster* en una máquina de escritorio. Se intuye que aprovechando todas las capacidad del GPU para realizar operaciones simultáneamente sobre múltiples elementos de datos, además de la eficiente administración de hilos, es posible incrementar las métricas de desempeño.

En particular, se demostró que el desempeño total del GPU mejora conforme aumenta la cantidad de tareas sencillas distribuidas en los hilos del mismo. Esto es más claro en la implantación embebida donde a medida que aumentamos las partículas, o bien las iteraciones, el desempeño mejora en forma significativa.

5.3. Estudio comparativo de algoritmos de optimización por cúmulo de partículas contra evolución diferencial implementados en un GPU multihilos

En esta parte se presenta un estudio empírico de comparación en implementaciones paralelas de PSO y DE en la arquitectura GPU multihilos. La tendencia actual en el desarrollo de GPU NVIDIA nos permite confirmar la consolidación de un nuevo modelo de programación paralela ofrecida a través de la herramienta de programación CUDA, donde el GPU no sólo incrementa su capacidad original

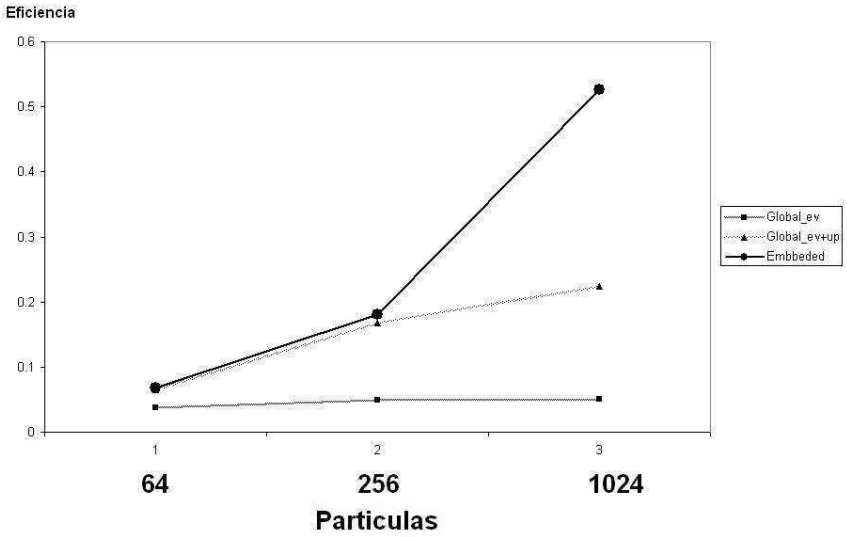


Figura 5.8: Comparativo de la eficiencia de las variantes PSO paralelas en función del número de partículas

de cálculo paralelo, sino que además tiene un rol preponderante como manejador multihilos [16].

5.3.1. Algoritmos PSO contra DE

En fechas recientes, los algoritmos PSO y DE son dos algoritmos poblacionales que han despertado el interés de muchos investigadores. Aunque su estructura básica es muy similar (ambos poseen bloques de inicialización, evaluación de aptitud, comparación y actualización), sin embargo usan diferentes reglas de comparación y actualización. En general el algoritmo PSO es una estrategia colaborativa que encuentra el óptimo como resultado del movimiento de individuos que intentan imitar al mejor individuo de un vecindario (local o global). Por otra parte, en la Evolución Diferencial, los individuos no intentan imitar al mejor individuo de cualquier vecindario, en su lugar, generan conjuntos de tres o más individuos con la finalidad de generar un nuevo individuo (usando operadores de mutación y cruza). Este nuevo individuo intenta mejorar el desempeño del mejor individuo de la actual población.

Partiendo del teorema de "no hay desayuno gratis" de Wolpert [94], se sabe que usando un conjunto limitado de funciones de prueba no se garantiza que un algoritmo con un buen desempeño, no será necesariamente competitivo con un

conjunto diferente de problemas. De hecho, es bien sabido que cada heurística es competitiva en un conjunto específico de problemas. Este estudio intenta obtener algunas pistas acerca de qué algoritmos pueden explotar el potencial del GPU para reducir el tiempo de convergencia y obtener buenos resultados, basado en el tipo de problema y la dependencia de tiempo de ejecución como una función de la cantidad de individuos o de la cantidad de iteraciones.

5.3.2. Implementación con programación paralela en el GPU

En el estudio comparativo de PSO, se demostró que una forma conveniente de paralelización de algoritmos poblacionales, en una arquitectura GPU multihilos, es una variante de la implementación de difusión. donde existe un procesador por cada individuo. En nuestra propuesta nosotros generamos un individuo por hilo, de tal forma que la evaluación y la actualización de un individuo son ejecutadas por un único hilo del GPU, mientras que las comparaciones, entre los individuos, es calculada dentro del GPU después de una necesaria y conveniente sincronización de hilos. A esta implementación le denominamos Embbeded, ya que la mayoría del los bloques funcionales, con excepción de la inicialización, son delegadas al GPU, y es la implementación paralela de los algoritmos de PSO y DE que se usa en este estudio para comparar su desempeño al ejecutarlas en el GPU.

El algoritmo 1, PSO, y el algoritmo 2, DE, fueron paralelizados como una estrategia de programación que consiste en la creación de un hilo por cada individuo. La regla que se siguió, fué reemplazar todos los ciclos secuenciales donde las iteraciones se encontraban en términos del número de individuos por medio de una llamada multihilo a la función kernel. De esta forma, mientras que en los ciclos de un código secuencial, cada ciclo de iteración es independiente de los otros, dentro del GPU todas las iteraciones intentan ser ejecutadas en forma concurrente al mismo tiempo. La herramienta de programación CUDA nos permite realizar una migración de los ciclos en kernels paralelos, transformando a cada iteración de ciclo en un hilo independiente [65]. Los programas CUDA ejecutan kernels paralelos con la siguiente sintaxis, la cual ya se había explicado:

$$kernel <<< dimGrid, dimBlock >>> (listadeparámetros); \qquad (5.7)$$

donde:
dimGrid y dimBlock son parámetros especializados que especifican las dimensiones del la malla de procesamiento paralelo, en bloques, y las dimensiones de los bloques en hilos, respectivamente.

La estructura general de un algoritmo PSO y un algoritmo DE se puede conjuntar en los siguientes bloques funcionales:

- Inicialización de la población. Inicia los individuos de la población con valores aleatorios.
- Evaluación de la función de aptitud.
- Comparación. Determina si un individuo tiene mejor aptitud que que el mejor registrado.
- Actualización. Cada individuo actualiza su posición siguiendo las reglas específicas del algoritmo.

Para ilustrar la paralelización del código, los bloques funcionales del código secuencial han sido reorganizados, resaltando los ciclos que están en términos del número de individuos. La idea principal es crear un hilo por cada individuo. En la implementación secuencial, todos los bloques funcionales son ejecutados en el procesador anfitrión. En contraste, en nuestra implementación paralela, a la que llamamos empotrada (embedded), únicamente el módulo de inicialización se encuentra en el anfitrión, ya que existen llamadas a kernel asociadas con la evaluación, la comparación y la actualización, todas ejecutándose en el GPU hasta que se alcance una condición de finalización. De forma específica, nuestra implementación embebida y su correspondiente módulo de inicialización, la inicialización de las semillas (una por hilo, usadas para la generación de los números aleatorios) es llevada a cabo en el anfitrión y permanece fuera del GPU. La condición anterior, garantiza que la generación de números aleatorios sea diferente, dentro del GPU, para cada hilo de procesamiento.

Es conveniente destacar algunas consideraciones prácticas a tomar en cuenta para obtener una implementación funcional en la paralelización de algoritmos poblacionales en multihilos:

- Tiempo de sobrecarga(Overhead)
- Sincronización (Synchronization)
- Contención (Contention)
- Generación de números aleatorios (Random numbers generation)

5.3.3. Experimentos y resultados

Los experimentos se llevaron a cabo en el sistema de la sección 3.6.1.

Procedimiento experimental

El objetivo fue evaluar el desempeño de nuestras implementaciones paralelas para PSO y DE, comparando una respecto a la otra. La dependencia del desempeño fue medida para un conjunto de funciones de prueba bien conocidas [20], en función del número de individuos y del número de iteraciones. Se seleccionaron 4 funciones multimodales, ya que ellas presentan una significativa complejidad de optimización. Siendo F01 y F03 funciones separables, mientras que F02 es no separable, entendiéndose como funciones separables aquellas que pueden ser escritas como combinaciones lineales de funciones de variables individuales:

F01- Función Rosenbrock generalizada:

$$f_1(\vec{x}) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{5.8}$$
$$-30 < x_i < 30$$
$$\min(f_1) = f_5(1, \ldots, 1) = 0$$

F02- Función Rastrigin generalizada:

$$f_2(\vec{x}) = \sum_{i=1}^{n-1}[x_i^2 - 10cos(2\pi x_i) + 10] \tag{5.9}$$
$$-5{,}12 < x_i < 5{,}12$$
$$\min(f_2) = f_9(0, \ldots, 0) = 0$$

F03- Función Griewank generalizada:

$$f_3(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{5.10}$$
$$-600 < x_i < 600$$
$$\min(f_3) = f_{11}(0, \ldots, 0) = 0$$

Los experimentos se llevaron a cabo para medir el desempeño de cada una de las implementaciones paralelas:

- Experimento 1. Para las implementaciones paralelas de PSO y DE, la medición del desempeño en términos de número de iteraciones. El número de iteraciones se hizo variable (1000 a 31000, en incrementos de 2000), los siguientes parámetros se fijaron:

- Individuos = 128
- Dimensiones = 30
- Criterio de parada = número de iteraciones

- Experimento 2. Para implementaciones paralelas de PSO y DE, mediciones de desempeño en términos de número de individuos. Mientras que el número de individuos se hizo variable (64, 128 hasta 1024, en incrementos de 128 individuos). Observese que en este experimento, el número de individuos se incrementó en pasos que son múltiplos de 64, los cuales son resultado de que en nuestras implementaciones paralelas, se creó al menos un hilo de procesamiento por individuo, y también debido a que NVIDIA recomienda la construcción de bloques de hilos con un tamaño que sea múltiplo de 64, para lograr explotar de forma adecuada los recursos del GPU [16], los siguientes parámetros se fijaron:

  - Iteraciones = 15000
  - Dimensiones = 30
  - Criterio de parada = número de iteraciones.

El algoritmo 1, PSO, fue implementado en la versión local (p.e. cada partícula tiene un vecindario local y conoce cuál es la partícula con el mejor global) y cuál es la mejor partícula del vecindario (la mejor local) los siguientes valores fueron fijados para todas las funciones probadas:

$$\begin{aligned} c_1 &= 1 \\ c_2 &= 1 \\ v_{max} &= 1 \\ v_{min} &= -1 \end{aligned}$$

Para PSO, el vecindario que se definió para cada partícula fue de 20 vecinos seleccionados de forma aleatoria, el valor es la norma en los esquemas de vecindarios en los algoritmos PSO.

El algoritmo 2, DE, se implementó en su variante rand/1/bin. El parámetro F se estableció a 0.6 para todas las funciones probadas, mientras que el parámetro CR se estableció a un a valor dependiente de la función a analizar, tal y como lo recomienda Mezura [20]: 0.9 para F01 y F03 (las cuales son funciones no separables), y 0.0 para F02 (función separable). Estos valores tambien son la norma para esta implantación de DE, y propician un comportamiento muy parecido entre DE y PSO.

Cada experimento se ejecutó 30 veces para cada función de prueba. Así se encontró el óptimo promedio, su desviación estándar y el tiempo promedio consumido en segundos, valores que se reportan en la sección de resultados.

5.3.4. Métricas de desempeño para procesamiento paralelo

Para poder evaluar, graficar y comparar el desempeño de nuestras implementaciones paralelas, se definieron las siguientes métricas:

- Costo computacional
- Aceleramiento.

El costo computacional C se define como el tiempo de procesamiento (en segundos) que un algoritmo dado consume. La cantidad de trabajo que puede realizar la definimos como rapidez (throughput) V se define como el inverso del costo computacional.

Definimos la rapidez, V, como el inverso del costo computacional:

$$V = \frac{1}{C} \tag{5.11}$$

La ventaja (speedup) es la tasa que resulta de dividir la rapidez de la variante de interés entre la rapidez de la variante de referencia (por ejemplo, la variante secuencial).

$$S = \frac{V_{obj}}{V_{ref}} \tag{5.12}$$

Finalmente, pero no menos importante, definimos a la eficiencia paralela, E, como la tasa que resulta de dividir la ventaja obtenida entre el número de unidades de procesamiento:

$$E = \frac{S}{n} \tag{5.13}$$

donde n es igual al número de núcleos de procesamiento del GPU (en nuestro caso 32).

5.3.5. Discusión de resultados

Debido a que el objetivo es comparar el desempeño de PSO y DE en sus implementaciones paralelas, ejecutándose en una arquitectura multihilos, GPU, con un conjunto seleccionado de funciones de prueba de algoritmos bioinspirados, se presentan a continuación los resultados para ambos algoritmos en la misma tabla y sus respectivas gráficas.

Debido a que es bien sabido que la calidad de convergencia en un algoritmo poblacional dado, durante el proceso de optimización, típicamente es muy

sensible a sus parámetros específicos (p. e. v_{max}, v_{min}, w_{max}, w_{min}, c_1, c_2, y el tamaño del vecindario para PSO, o CR y F para DE) y también es dependiente del problema mismo (p.e. la función objetivo), determinar que algoritmo tiene mejor convergencia para cada tipo de problema está fuera de los alcances de este trabajo. Sin importar que algoritmo tiene mejor convergencia, estamos más interesados en determinar el efecto de la variación del número de individuos e iteraciones en el costo promedio (p.e. tiempo consumido) y en la forma general de la curva de convergencia. Todas las funciones tienen un valor óptimo de cero.

5.3.6. Resultados del experimento 1

En esta sección se presentan las mediciones de desempeño para las implementaciones paralelas de PSO y DE, en términos de iteraciones variables.

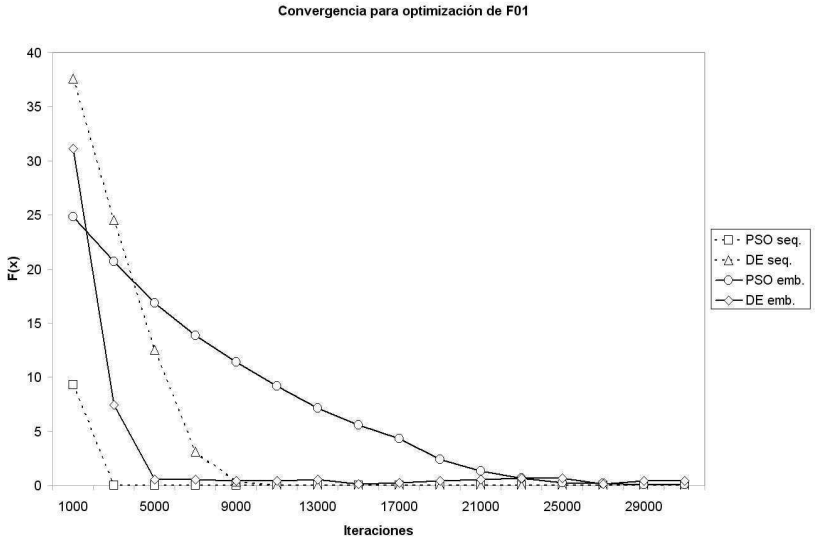


Figura 5.9: Convergencia para optimización en la función F01, con un número fijo de 128 individuos y con iteraciones variables

Como se puede apreciar, figuras 5.9, 5.10 y 5.11, el efecto de la variación del número de iteraciones, con un conjunto fijo de parámetros de 128 individuos y F = 0.6 para todas las funciones, y cambiando el parámetro CR dependiendo si son funciones separables o no separables. El algoritmo DE converge a un buen valor en las primeras 1000 a 5000 iteraciones. Incrementando el número de iteraciones no muestra una mejora significativa. Asimismo, para este conjunto de experimentos, DE tiene un mejor desempeño que PSO desde el inicio hasta el fin en todas las funciones, excepto en la función F01 (Rosenbrock genralizada) figura 5.9, donde

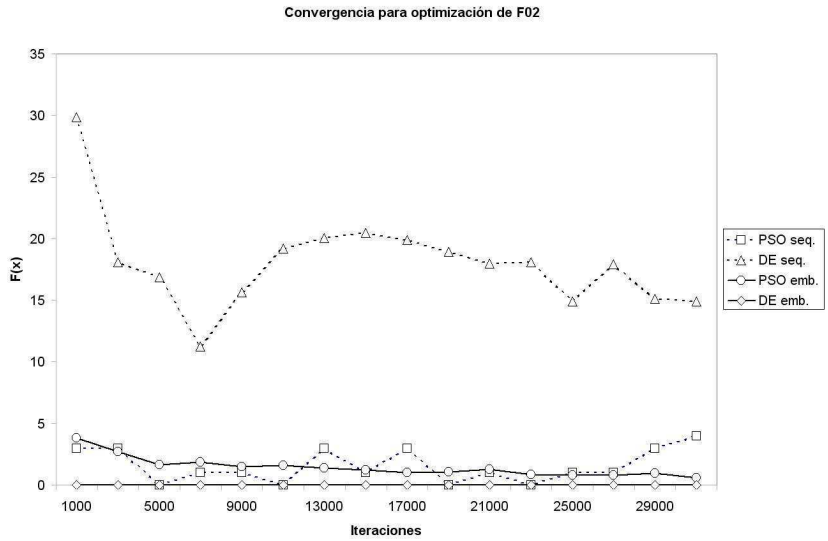


Figura 5.10: Convergencia para optimización en la función F02, con un número fijo de 128 individuos y con iteraciones variables

al inicio (iteración 1000) PSO tiene un mejor desempeño que DE, pero en la iteración 3000 y de ahí en adelante, hasta las 23,000 iteraciones, DE es mucho mejor que PSO.

En cuanto al tiempo consumido (costo), pudimos observar que al aumentar las iteraciones da como resultado en un crecimiento aproximadamente lineal para PSO y DE, y ninguno se muestra mejor que el otro, presentando resultados mixtos, por una parte en las funciones F02 y F03, PSO tiene un desempeño ligeramente mejor que DE, figuras 5.13 y 5.14. Por otra parte, en la función F01, DE mostró un desempeño ligeramente mejor que PSO, figura 5.12. Esto resulta en una aceleración diferente para cada función: primero la función F01 tiene una aceleración decreciente de 1.5 a 0.9. La función F02 se observa un factor de aceleración de 1.25 en todas las iteraciones. Finalmente en F03 se tiene una aceleración promedio de 1.24.

5.3.7. Resultados del experimento 2

En esta sección se presentan las mediciones de desempeño para las implementaciones paralelas de PSO y DE, en términos de iteraciones e individuos. Respecto a la dependencia de desempeño debido al número de individuos, los resultados experimentales se presentan en las Tablas 1 a la 4 y se grafican en las

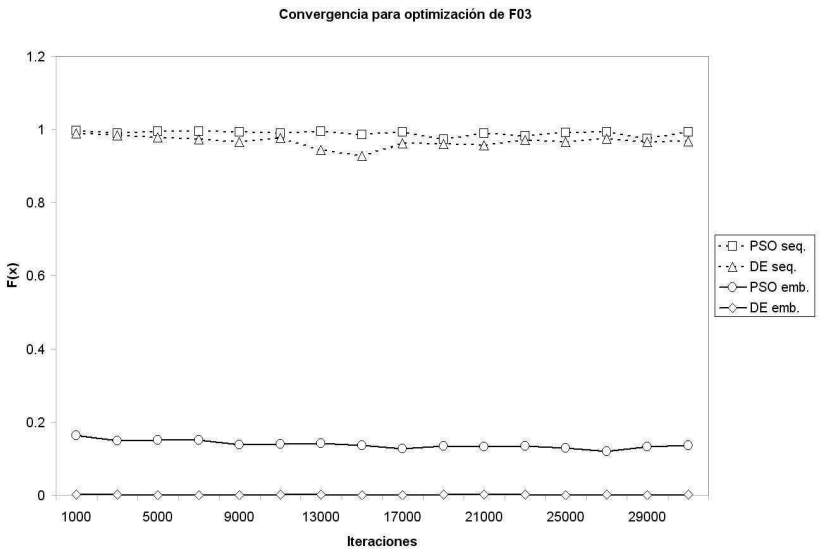


Figura 5.11: Convergencia para optimización en la función F02, con un número fijo de 128 individuos y con iteraciones variables

figuras 3 a la 10. De forma similar, los resultados mostrados en las tablas 5 a la 8, se grafican en las figuras 11 a la 18, y corresponden al desempeño dependiente del número de iteraciones.

En cuanto al efecto de modificar el número de individuos, es destacable que para todas las funciones probadas, con los parámetros especificados, el algoritmo DE converge a un buen valor con sólo 256 individuos. El incremento de individuos con un valor superior a 256 no produce una mejora significativa para el algoritmo DE. Sin embargo el comportamiento general de DE es asintótico respecto al valor óptimo en el rango ilustrado aquí, se ha afirmado que el desempeño de DE puede disminuir si el número de individuos es lo suficientemente grande, [20]. En nuestro caso particular, confirmamos que este comportamiento previsto es evidente con mas de 4096 individuos. La variación de los individuos dentro del rango especificado, nos permitió observar que el comportamiento PSO es menos regular que el de DE. De hecho, para el caso específico de la función F01 (Rosenbrock Generalizada), la convergencia PSO no mejora, pero sin embargo empeora a medida que el número de individuos se incrementa.

Respecto al tiempo consumido (costo), se pudo observar que incrementando el número de individuos resulta en un crecimiento aproximadamente lineal del tiempo consumido, tanto para PSO como para DE. Aparentemente, dentro del rango gráficado, ningún algoritmo muestra una clara ventaja, lo cual se traduce en una aceleración, de PSO, de aproximadamente 1, respecto a DE. Sin embargo,

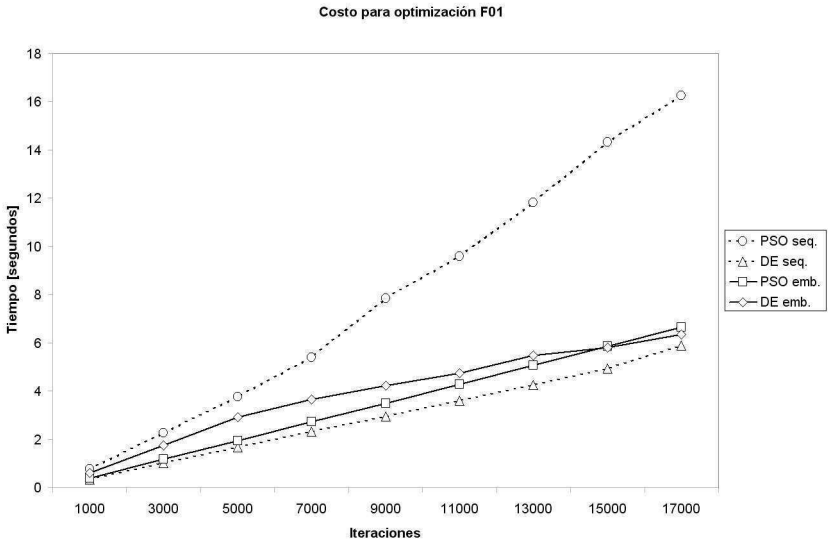


Figura 5.12: Costo en la función F01, con un número fijo de 128 individuos y con iteraciones variables

con un número suficientemente grande de individuos aparece un espacio significativo, medible en segundos, lo cual resulta en una pequeña pero evidente ventaja para uno de los dos algoritmos, dependiente de la función objetivo. Por ejemplo, con 15,000 iteraciones y 8192 individuos, la optimización de la función F01 con PSO consume 175 segundos respecto a los 197 que le toma a DE. Por otra parte, la optimización de la función F02 le toma 168 segundos a PSO mientras que a DE consume 141 segundos. En el primer caso resulta que PSO tiene una ventaja de aceleramiento de 1.2 con respecto a DE. En el segundo caso resulta que DE tiene una ventaja de aceleración de 1.19 con respecto a PSO.

5.3.8. Análisis de resultados

Los resultados experimentales demostraron que el algoritmo paralelo de DE tiene un mejor comportamiento optimizando el conjunto de nuestras funciones de prueba, con una dimensionalidad significativa de 30. Es destacable que DE, usando valores apropiados para los parámetros CR y F que dependen del problema, pueda ser capaz de converger a un óptimo global con tan poco como 64 individuos y 3000 iteraciones, recordando que la dimensionalidad del problema es de 30.

Los resultados experimentales mostraron que con el uso de estas herramientas, CUDA y GPU, es posible tener variantes paralelas para algoritmos poblacionales

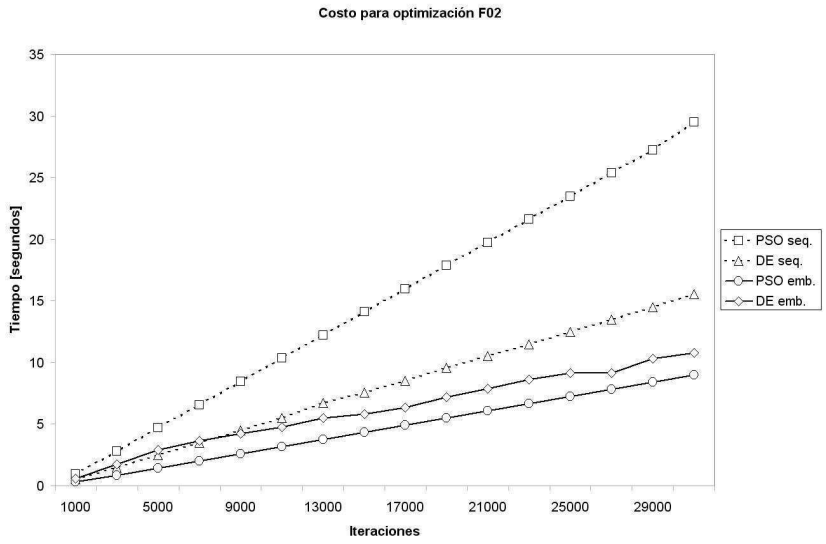


Figura 5.13: Costo en la función F02, con un número fijo de 128 individuos y con iteraciones variables

con una significativa mejora en el desempeño, por medio de una programación paralela simple y directa.

En términos generales, el conjunto de experimentos realizados demostró que una programación paralela basada únicamente en las características multihilos del GPU, cuando menos resulta en una aceleración proporcional al número de núcleos del GPU, obteniendo el poder computacional de un cluster en una computadora personal convencional, esto es parte de la naturaleza de los algoritmos probados, que son de carácter poblacional y con elementos individuales factibles de ser programados en la arquitectura del GPU debido a sus características inherentes como son necesidades reducidas de memoria y un conjunto de funciones iguales pero con diferentes datos. Además, se demostró que el desempeño total del GPU mejora cuando se aumenta la cantidad de tareas simples distribuidas en hilos del GPU. Este comportamiento puede ser apreciado en el desempeño de todas las variantes paralelas probadas, pero es más evidente en la variante incrustada (embbeded), donde el desempeño mejoró significativamente.

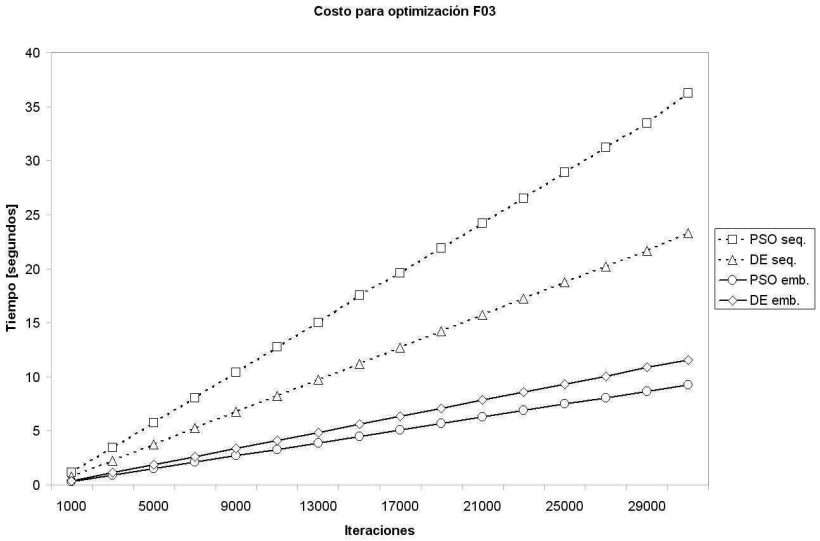


Figura 5.14: Costo en la función F03, con un número fijo de 128 individuos y con iteraciones variables

5.4. Implementación de la metodología propuesta en arquitectura paralela multihilos

En esta sección se presentan las consideraciones de implantación, así como los resultados obtenidos, durante la paralelización de la metodología de extracción de parámetros ópticos empleando el enfoque de programación de GPU basado en el empleo de múltiples hilos. En esta parte se usa el mismo sistema definido en la sección 3.6.1, el cual cuenta con capacidad para administrar múltiples hilos concurrentes, además de la tradicional paralelización de operaciones, bajo el concepto SIMT (instrucción simple múltiples hilos) [65].

La estrategia de programación paralela para la implantación de la metodología consistió, al igual que en las implantaciones de los experimentos anteriores, en crear un hilo por cada individuo, a fin de poder distribuir las tareas del algoritmo de Evolución Diferencial a cada uno de ellos. La táctica para generar la versión paralela del código consistió en remplazar todo lazo que estuviera en términos del número de individuos por una llamada a un kernel multi-hilo. Adicionalmente, se implantó el algoritmo de Extracción Secuencial como referencia.

Finalmente, la versión de Extracción de Parámetros incrustada, el único bloque correspondiente al algoritmo bioinspirado que queda en poder del procesador central es el de inicialización de la población, ya que se llama a una función

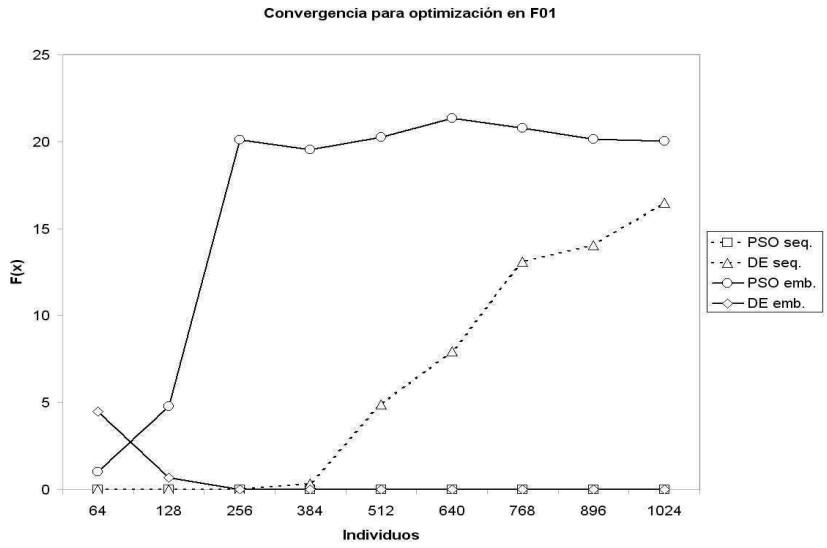


Figura 5.15: Convergencia para optimización en la función F01, con un número fijo de 2000 iteraciones y con número de individuos variable

kernel, Extracción de Parámetros, que ejecuta las tareas asociadas con la evaluación, la comparación y la imitación, hasta cumplir con el criterio de parada.

Cabe recordar algunas consideraciones prácticas que deben tomarse en cuenta para lograr una implantación funcional de un algoritmo PSO o de ED paralelo en un GPU, primeramente la implantación debe considerar tres tipos de funciones:

- *Anfitrión* - las que se ejecutan en el anfitrión
- *Globales* - las que se llaman desde el anfitrión pero son ejecutadas en el GPU
- *Dispositivo* - las que se ejecutan en el dispositivo

La ejecución de la extracción de parámetros en el GPU se lleva a cabo en cuatro partes:

- *Inicialización en el anfitrión* - Se da un inicio general a todas las variables locales, globales y de dispositivo, se inicia una arreglo con números aleatorios que se pasarán al GPU, también se lee la imagen de referencia y se pasan estos datos a matrices del dispositivo mediante la función CudaMemcpy.
- *Llamada de la función Kernel Extracción de Parámetros* - Se envían a la función de kernel el número de hilos a generar así como el número de bloques que éstos requieren.

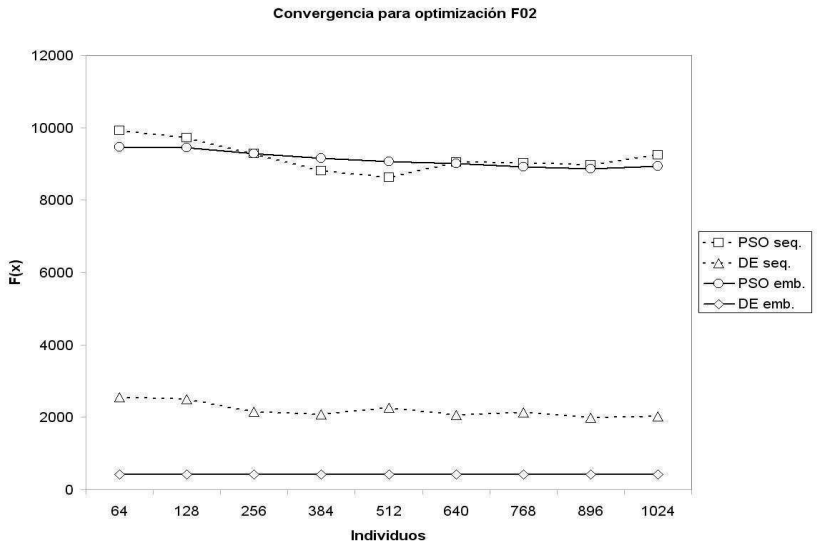


Figura 5.16: Convergencia para optimización en la función F02, con un número fijo de 2000 iteraciones y con número de individuos variable

- *Ejecución de la función Extracción de Parámetros* - Se calcula, mediante el algoritmo de cálculo de aptitud, la aptitud de cada individuo tomando como valores el vector de solución del individuo, esta información se le pasa como parámetros a un trazador de rayos, el cual genera una imagen, esta imagen se compara con la imagen de referencia y basado en el resultado se asigna la aptitud al individuo, así con cada individuo de toda la población y hasta alcanzar el criterio de parada durante n generaciones.

- *Despliegue de resultados en el anfitrión* - Una vez que concluye la función Extracción de Parámetros, la información que obtuvo de los mejores individuos de cada generación es guardada en una matriz, dicha matriz se envía al anfitrión. El anfitrión lee los datos de la matriz y genera las imágenes correspondientes a cada mejor individuo de las generaciones reportadas, para visualizar la información.

Como es bien sabido, la parte que más tiempo de cálculo consume en las heurísticas bioinspiradas es la prueba de la función objetivo, y en este caso no es la excepción. La metodología propuesta usa un algoritmo de traza de rayos para generar una imagen de prueba que se compara con la imagen de referencia, el resultado se almacena como la aptitud de cada individuo de Evolución Diferencial. Debido a lo anterior el número de individuos, así como el número de generaciones, determina la cantidad de llamados a la función objetivo. En este caso se realizaron pruebas con 60 individuos y con con un numero inicial de 150

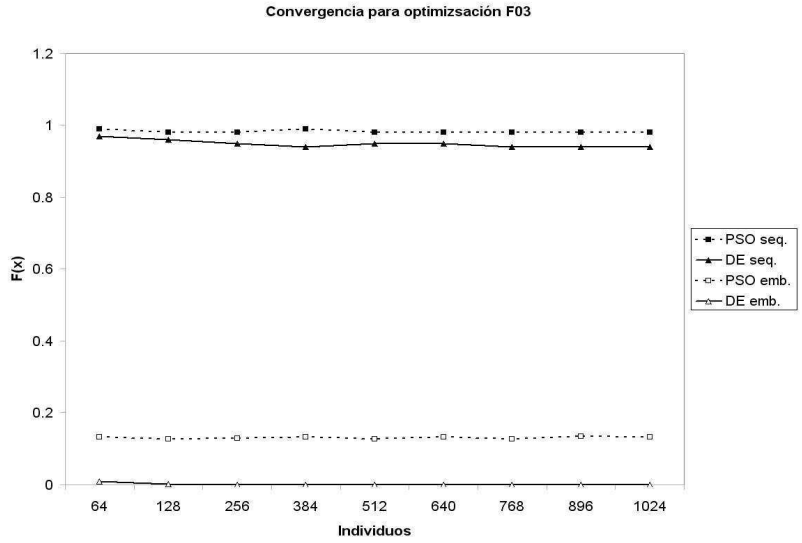


Figura 5.17: Convergencia para optimización en la función F03, con un número fijo de 2000 iteraciones y con número de individuos variable

generaciones y un numero final de 1050, ver tabla 6.4.

Tabla 5.10: Porcentaje de tiempo de ejecución

Desviación		
Generaciones	% tiempo de heurística	% tiempo de traza de rayos
150	10.1	89.9
250	14.3	85.2
350	17.2	82.8
450	17.9	82.1
550	19.3	80.7
650	19.5	80.5
750	20.7	79.3
850	22.8	77.2
950	22.3	77.7
1050	21.8	78.2

Las pruebas de desempeño indican que el tiempo consumido por la traza de rayos es lo que mas tiempo le toma al algoritmo, es lógico debido a que es la parte de cálculo mas intensivo de la metodología, asimismo se observa una ligera disminución en el porcentaje del tiempo usado por la traza de rayos a medida que aumenta el número de generaciones, aunque este se estabiliza cuando se pasa

de 1000. Una vez implantado el algoritmo en el GPU se pasa a la parte de las pruebas de ejecución del capítulo siguiente.

Capítulo 6

Presentación y discusión de resultados

6.1. Introducción

El presente capítulo reporta las pruebas y los resultados obtenidos con diferentes imágenes objetivo, se destacan la obtención de la posición de la fuente de iluminación así como su intensidad. Asimismo, se obtiene un color aproximado de los objetos en el escenario y sus características de material, como dureza y rugosidad.

6.2. Experimento 1

Se tiene un escenario sencillo con un plano que representa el piso, una esfera de diferente material y una fuente de iluminación, se capturan 7 imágenes objetivo de diferentes colores y materiales, figura 6.1.

Como ya se mencionó, el vector de variables de cada individuo tiene una correspondencia con la fuente de iluminación del escenario, de forma tal que las primeras tres variables de cada individuo corresponden a la posición X, Y y Z de la fuente de luz (pos_L), y la cuarta variable corresponde a la intensidad de la fuente ($Potencia_L$), las siguientes variables corresponden a los valores HSV de la esfera (esf), el siguiente valor corresponde al parámetro brillo (dureza) y el último a la chispa (rugosidad) y posteriormente los mismos valores del plano (pla), de forma tal que el vector X de cada individuo de evolución diferencial está de la siguiente forma:

Figura 6.1: Imágenes de referencia para los experimentos

$$[Xpos_L, Ypos_L, Zpos_L, Potencia_L, H_{esf}, S_{esf}, V_{esf}, BR_{esf}, C_{esf}, H_{pla}, S_{pla}, V_{pla}, BR_{pla}, C_{esf}] \quad (6.1)$$

El algoritmo de evolución diferencial es multiobjetivo, para minimizar tres funciones, la diferencia de las componentes de rojo, verde y azul de la imagen objetivo y la generada por traza de rayos. Donde cada componente es un objetivo a minimizar. Se tiene una dominancia suave, con una dominancia de dos o tres elementos.

Se han realizado 10 corridas del algoritmo, por cada objeto, con un criterio de parada de 100 iteraciones para encontrar la posición de la fuente de iluminación y su intensidad, a continuación se realizan 200 generaciones para el cálculo de los componentes del objeto (color HSV, brillo y chispa) y finalmente se realiza la etapa de optimización de color con 20 valores de prueba para H, S y V. El algoritmo de Evolución Diferencial se ejecuta con 60 individuos. Con una variable CR de 0.8 y F de 0.6. A continuación se muestran los resultados de una corrida para una esfera de color, se pueden observar claramente las tres etapas de la metodología, figura 6.2.

- De la imagen PAS00001 a la imagen PAS00091 es la etapa donde se optimizan los parámetros de la fuente de iluminación

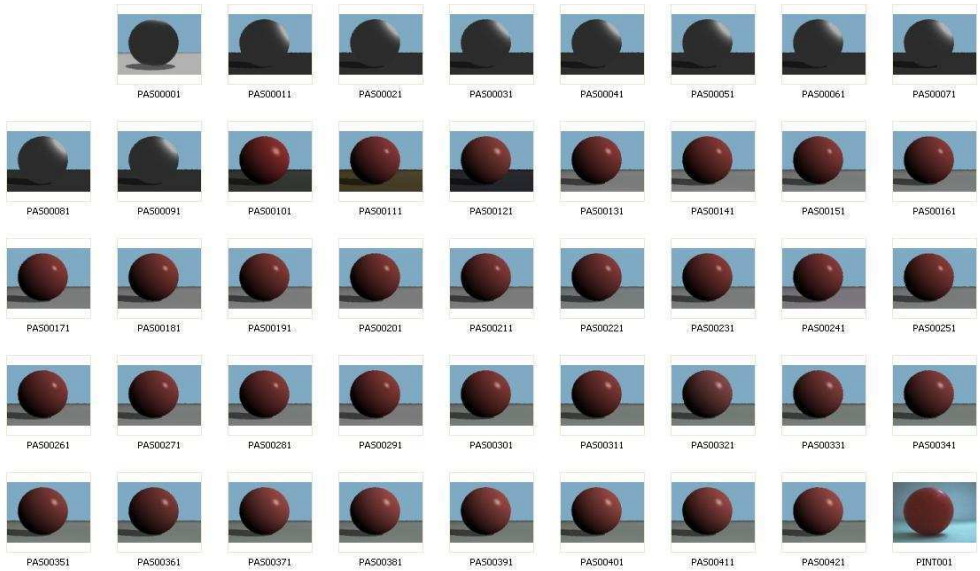


Figura 6.2: Evolución de las imágenes generadas, se presenta la mejor imagen de cada 10 generaciones, se puede apreciar el número de cada generación, después de las letras PAS

- De la imagen PAS00101 a la imagen PAS00291 es la optimización de los parámetros de los objetos, color HSV, Brillo y Chispa
- De la imagen PAS00301 a la PAS00421, es la optimización del color

El promedio de las 10 corridas para la esfera roja entrega la información de la posición de la fuente de iluminación, considerando que el valor entregado se multiplica por 100 y al resultado se le restan 50, para tener un rango de posición de la fuente de luz de ± 500 unidades de distancia para X, Y y Z. Sabiendo que la distancia de la fuente de luz en el escenario real es de:

- X = 10mm
- Y = 80mm
- Z = -35mm

Los valores entregados por el algoritmo son:

- X = 0.521
- Y = -0.02
- Z = 0.73

Transformados en unidades de distancia del mundo virtual son:

- X = 50 - (0.521 * 100) = -2.1
- Y = 50 - (-0.02 * 100) = 54.1
- Z = 50 - (0.73 * 100) = -23.0

Los datos no son iguales a los del mundo real pero están con una aproximación razonable. La búsqueda de la posición de la fuente de iluminacion se muestra en la figura 6.3.

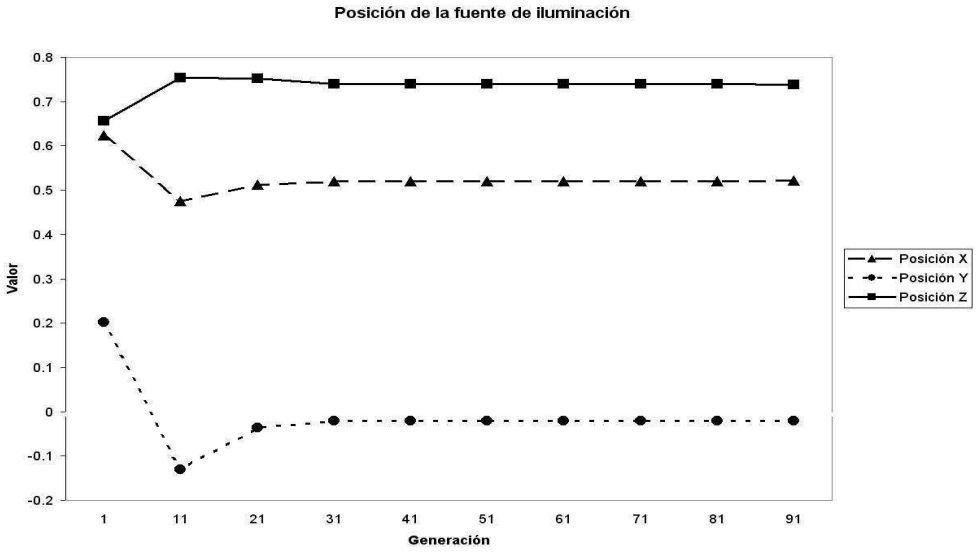


Figura 6.3: Búsqueda de los parámetros de la Intensidad de la fuente de iluminación

La intensidad de la fuente de iluminación también determina la diferencia de una imagen con respecto a la otra, y a medida que pasan las iteraciones el algoritmo encuentra un valor bueno para la intensidad de la fuente de iluminación, figura 6.4.

Para la determinación de los componentes HSV de color de la esfera se ejecutan 200 generaciones, y a lo largo de las 200 se está intentando encontrar buenos valores para el color de la esfera, sin embargo en la generación 300, entra la parte de optimización directa, sin algoritmo poblacional, y al final se encuentra un valor ligeramente mejor que el encontrado por la Evolución Diferencial, figura 6.5. Es posible apreciar que en las primeras generaciones se tiene una propuesta de color de la esfera roja es un color bueno, pero a medida que avanzan las generaciones se va depurando el color, sobre todo por individuos que prueban componentes HSV distintos, sobre todo en el componente H y en el S.

Asimismo se está probando con diferentes valores de brillo y chispa en el material de la esfera, figura 6.2. Esto se puede apreciar en la forma, definición

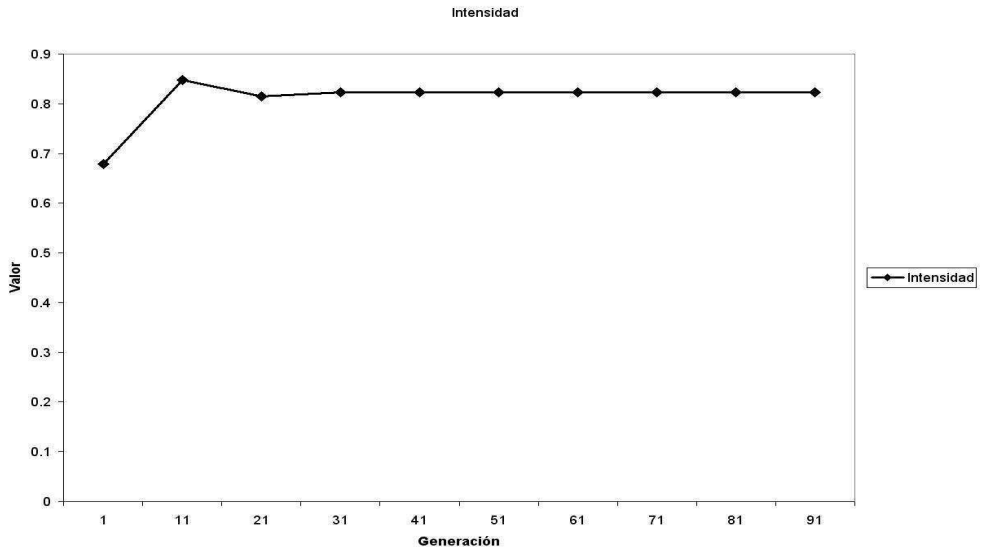


Figura 6.4: Aptitud de cada una de las componentes, a medida que disminuye el valor en la gráfica, disminuye la diferencia entre la imagen generada y la imagen objetivo

y tamaño de la reflexión especular de una generación respecto a otra, lo cual se representa en la figura 6.6.

Se realizaron experimentos con las 7 esferas que se muestran en la figura 6.1. En cada uno de los experimentos se tuvo un desempeño similar al de la esfera roja, siendo el objeto más difícil de optimizar la esfera blanca, y la esfera verde y la azul las que tuvieron una convergencia mayor, es decir, las que en más ocasiones de las 10 corridas llegaron a propuestas muy similares entre cada corrida.

6.2.1. Evaluación de resultados

Para evaluar la similitud de una imagen con respecto a otra se usa la misma función de aptitud que se desarrolló para Evolución Diferencial:
F01- Función de evaluación para asignar aptitud, considerando todos los píxeles de la imagen:

$$Dif = \sum_{i=0}^{i=n} |RGBObj_i - RBGTray_i| \tag{6.2}$$

Donde:

- $n = Xmax * Ymax$
- $RGBObj_i$ = Valor de cada componente del píxel i de la imagen objetivo

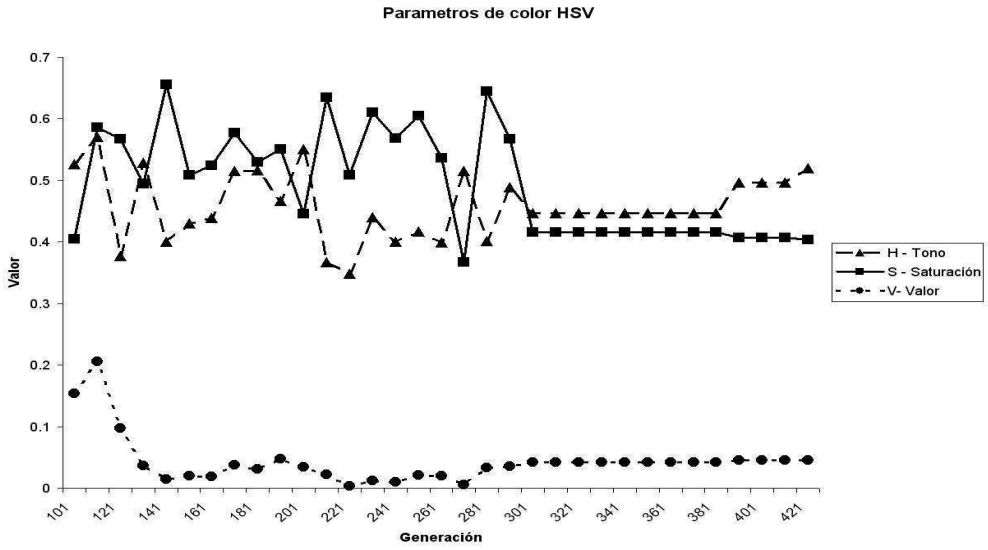


Figura 6.5: Búsqueda y estabilización de valores buenos para el color de la esfera

- $RGBTraza_i$ = Valor de cada componente del píxel i de la imagen generada por el trazador de rayos

Esta función se incluyó en un programa que abre la imagen de referencia (imagen objetivo) y la imagen generada por el trazador de rayos con los parámetros finales del mejor individuo, en la tabla 6.1, se presentan los resultados de los mejores individuos para cada esfera:

Tabla 6.1: Desviación de cada uno de los mejores individuos con respecto a su imagen objetivo

	Desviación			
Esfera	% R	% G	% B	Promedio
Roja	6.2560	5.2532	5.2027	5.5706
Verde	6.9311	6.1894	4.3935	5.8380
Azul	10.8471	9.5256	7.4329	9.2685
Amarilla	7.2940	6.9565	10.1744	8.1416
Blanca	12.6140	10.5589	6.2914	9.8214
Naranja	8.7913	7.2358	15.9613	10.6628
Metal	7.7390	6.4553	4.4356	6.2100

Los resultados del algoritmo se compararon contra otros métodos que obtienen parámetros de renderizado y los implantan en objetos virtuales, la comparativa se realizó obteniendo las imágenes de los artículos de cada método, ya que no

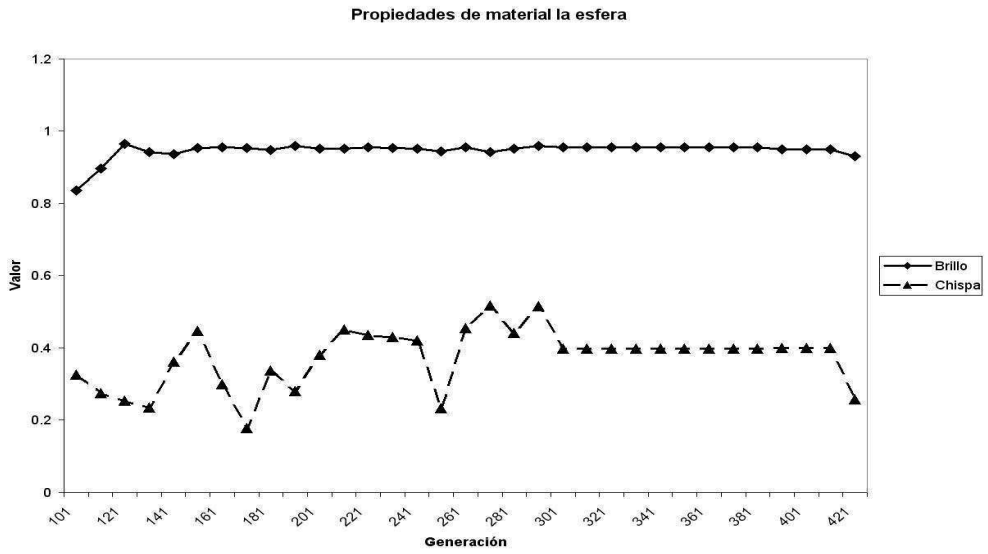


Figura 6.6: Evolución de los parámetros de brillo y chispa, para el material de la esfera roja

fue posible implementar los métodos que los autores proponen, debido a que se requiere equipo especial que ellos diseñaron y con el cual realizaron su proceso de extracción de parámetros, es probable que el desempeño sea diferente si se obtienen imágenes de referencia y generadas directamente de los autores. Los resultados se presentan en la tabla 6.6 Cabe aclarar que en el caso de los

Tabla 6.2: Comparación de la desviación de los diferentes métodos

	Desviación			
Metodo	% R	% G	% B	Promedio
Wai Kit Addy 1	8.92	9.55	8.54	8.97
Wai Kit Addy 2	12.41	9.67	8.58	10.22
Ward	3.30	3.48	3.14	3.30
Ward-Durn	2.20	2.32	2.25	2.25
Nuestro Mejor	6.2560	5.2532	5.2027	5.5706
Nuestro Peor	8.7913	7.2358	15.9613	10.6628

modelos de Wai Kit Adi, no se tiene una imagen del modelo real y del modelo virtual que ajusten en tamaño y forma, así que se tomaron imágenes lo mas parecidas posible. En el caso de los modelos Ward, no se tenía una imagen del modelo real así que se comparó contra la referencia que los autores presentan, la cual también es un modelo virtual y por lo tanto la diferencia entre ambas imágenes es menor.

6.3. Experimento 2

Como un segundo conjunto de experimentos se compara la eficiencia de extracción de parámetros de la metodología respecto a la eficiencia de una persona con conocimientos de renderizado y de modelado de objetos.

Figura 6.7: Imagen de referencia para los ajustes a mano

El objetivo es que el sujeto de prueba intente igualar la imagen objetivo (esfera amarilla adquirida con cámara digital), partiendo del hecho de que ya se tiene un escenario virtual listo y ajustado a las proporciones del escenario real, el sujeto tiene que ajustar los parámetros de la posición de la fuente de iluminación, su intensidad, así como las componentes de color de la esfera y del piso. El objetivo es la esfera amarilla, figura 6.7.

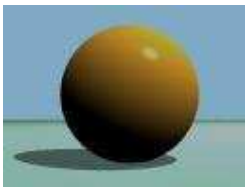


Figura 6.8: Imagen obtenida por aproximación manual

La mejor imagen generada por la persona con el método semiautomático, figura 6.8, obtiene un color semejante al color del suelo y de la esfera, sin embargo no es mejor que el obtenido por la metodología, figura 6.9.

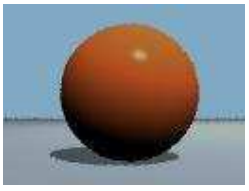


Figura 6.9: Imagen obtenida por uso de la metodología

La tabla de resultados de cada uno de los experimentos muestra que el sujeto se aproxima gradualmente a un buen resultado, y le toma un tiempo de 1 hora

con 57 minutos, ver tabla 6.3.

Tabla 6.3: Tabla parcial de experimentos para obtención de parámetros por método semiautomático

Diferencia de imagen objetivo vs imagen aproximada por método semiautomático				
Experimento	% R	% G	% B	Promedio
1	29.11	17.08	16.9	21.03
5	29.95	15.57	13.03	19.51
10	28.30	15.35	12.79	18.81
15	27.53	15.38	11.62	18.17
20	26.91	15.38	11.68	17.99
25	26.31	15.65	12.07	18.0
30	26.21	15.68	11.74	17.87
35	22.4	13.76	11.75	15.96
40	21.17	13.96	12.14	15.75
45	21.5	13.94	12.32	15.92
50	20.5	13.87	12.06	15.47

La curva de porcentaje de diferencia por componente RGB, figura 6.10, muestra una aproximación gradual para cada una de las componentes, mientras que la figura 6.11 muestra una mejora gradual total hasta un porcentaje de 15.47 de aproximación de la imagen generada por método semiautomático. Sin embargo la aproximación obtenida por la metodología propuesta, en este conjunto de experimentos, es de un porcentaje con diferencia de 13.3 %, la metodología lo obtiene en 32 minutos usando una programación secuencial, si se usa la implantación en la arquitectura del GPU se tiene una velocidad de 4.2 minutos.

La tabla 6.4, muestra que aunque la aproximación por método semiautomático se acerca a un buen resultado, la metodología obtiene mejores resultados en un tiempo menor. Asimismo muestra que el modelo de iluminación es una limitante para una mejor aproximación, tanto para una persona como para la metodología.

Tabla 6.4: Comparación resultados

Desviación					
Método	% R	% G	% B	Promedio	Tiempo (mins)
Manual	20.5	13.87	12.06	15.47	117.0
Metodología secuencial	15.77	12.60	11.80	13.34	32.0
Metodología GPU	15.77	12.60	11.80	13.34	4.2

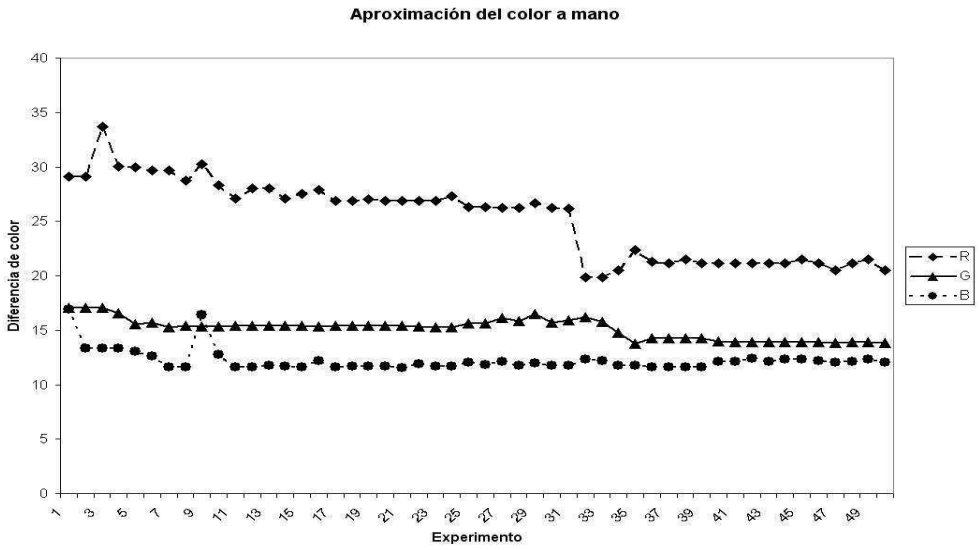


Figura 6.10: Imagen de referencia para los ajustes a mano

6.3.1. Evaluación de resultados

Es probable que una persona pueda, gradualmente, tener mejores tiempos para la obtención de los parámetros, sin embargo se debe considerar que el GPU usado tiene 32 núcleos y actualmente se venden en el mercado tarjetas con 240 y 480 núcleos a un precio accesible, esto significa una mejora significativa en el tiempo que la metodología puede obtener los parámetros. Asimismo, significa que una persona no tiene que estar en una tarea repetitiva y tediosa como lo es la obtención de dichos parámetros para el modelado de sus objetos y escenarios, y puede usar este tiempo para generar mejores modelos o mejores animaciones.

6.4. Experimento 3

El desempeño de nuestro método es altamente dependiente del modelo de iluminación usado, se programaron tres modelos de iluminación básicos y el desempeño del tercero es superior al primero y segundo, debido a que considera un mayor número de factores, como son la brillantez y la chispa del material, asimismo el cálculo del color del pixel resultante es de mayor exactitud a los dos anteriores. Es de suponer que al usar un modelo de iluminación más complejo, el desempeño también mejore. Para demostrar lo anterior se propone otro conjunto de experimentos en el cual la imagen objetivo no es adquirida por una cámara digital, en su lugar la imagen objetivo se genera usando los mismos escenarios con el modelo de iluminación más completo. Y verificar el nivel de aproximación

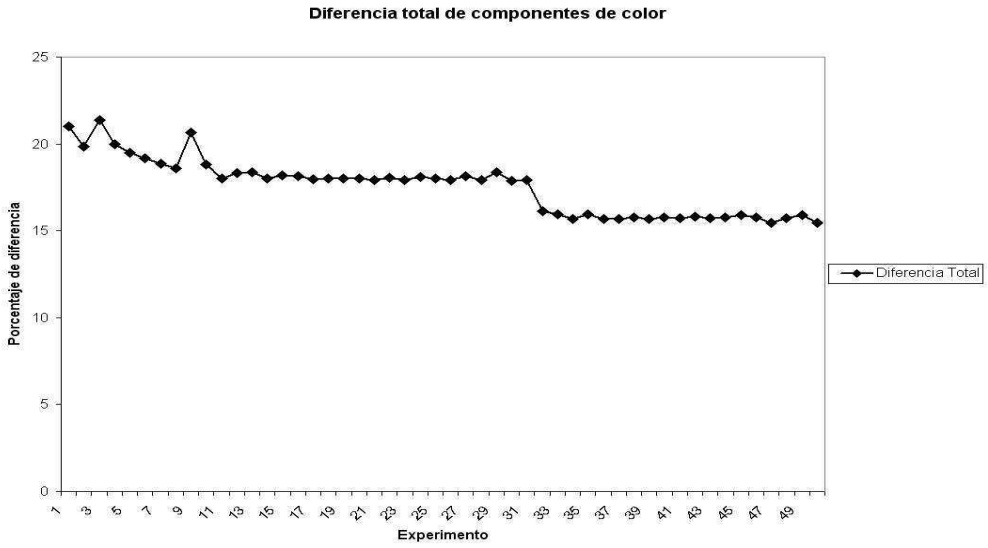


Figura 6.11: Gráfica de diferencia de los componentes de color de la imagen generada por el usuario, respecto a la imagen de referencia.

que logra la metodología.

Se diseñó un escenario virtual con un plano que representa el piso, una esfera de diferente material y una fuente de iluminación, se generaron 6 imágenes objetivo de diferentes colores y materiales, figura 6.12.

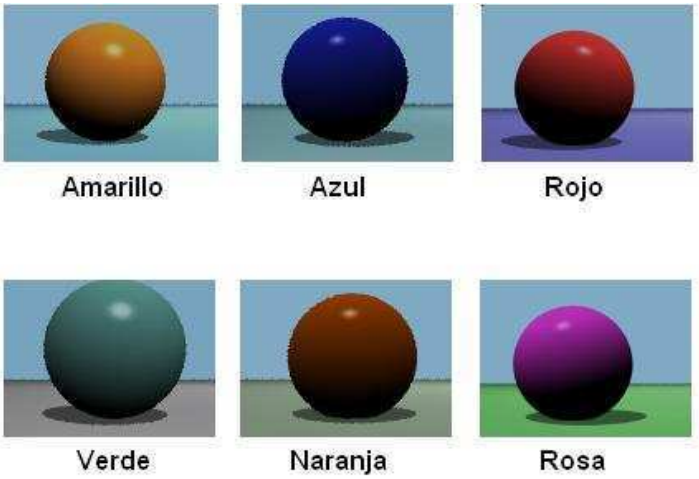


Figura 6.12: Imágenes de referencia para los experimentos

Al igual que en el experimento anterior, en el vector de variables de cada indi-

viduo las 4 primeras tienen una correspondencia con la fuente de iluminación del escenario, las 3 variables siguientes corresponden a los valores HSV de la esfera (esf), el siguiente valor corresponde al parámetro brillo (dureza) y el último a la chispa (rugosidad) y posteriormente los mismos valores del plano (pla), de forma tal que el vector X de cada individuo de evolución diferencial está conformado de la siguiente forma:

$$[Xpos_L, Ypos_L, Zpos_L, Poten_L, H_{esf}, S_{esf}, V_{esf}, BR_{esf}, C_{esf}, H_{pla}, S_{pla}, V_{pla}, BR_{pla}, C_{esf}] \tag{6.3}$$

A diferencia del experimento anterior, en este experimento si se conocen con exactitud los parámetros que se están buscando, se tiene la posición exacta de la fuente de iluminación, así como su intensidad para cada escenario, lo mismo para cada uno de los parámetros de los objetos que en el hay, en este caso la esfera y el plano.

Se han realizado 10 corridas del algoritmo, por cada objeto, con un criterio de parada de 50 iteraciones para encontrar la posición de la fuente de iluminación y su intensidad, a continuación se realizan 200 generaciones para el cálculo de los componentes del objeto (color HSV, brillo y chispa). El algoritmo de Evolución Diferencial se ejecuta con 60 individuos. Con una variable CR de 0.8 y F de 0.6. A continuación se muestran los resultados de una corrida para una esfera de color Amarillo, figura 6.13.

Se pueden observar claramente las etapas de la metodología, figura 6.13:

- De la imagen PAS00001 a la imagen PAS00051 es la etapa donde se optimizan los parámetros de la fuente de iluminación
- De la imagen PAS0052 a la imagen PAS00251 es la optimización de los parámetros de los objetos, color HSV, Brillo y Chispa

El promedio de las 10 corridas para la esfera Verde entrega la información de la posición de la fuente de iluminación. Sabiendo que la posición de la fuente de iluminación en el escenario generado manualmente es de:

- X = 0.430
- Y = 0.173
- Z = 0.731
- Intensidad = 300

Los valores entregados por el algoritmo son:

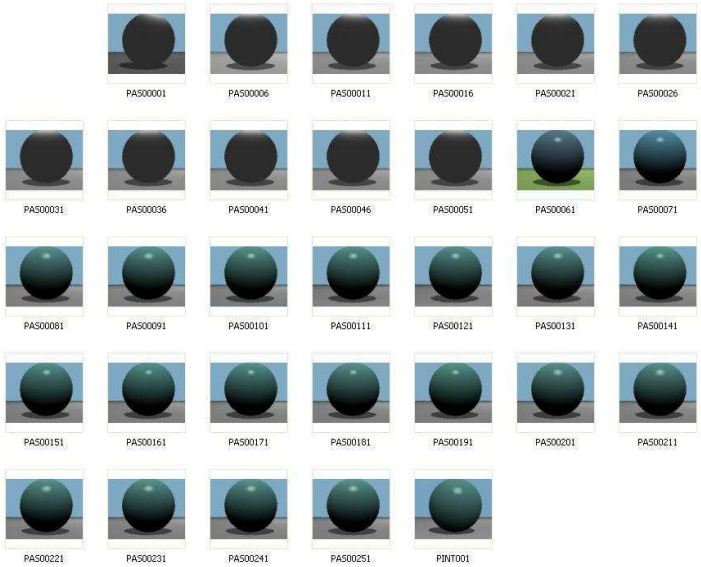


Figura 6.13: Evolución de las imágenes generadas, se presenta la mejor imagen de cada 10 generaciones, se puede apreciar el número de cada generación, después de las letras PAS

- X = 0.489
- Y = 0.966
- Z = 0.936
- Intensidad = 600

Los parámetros estimados de la fuente de iluminación no son iguales a los del escenario objetivo, sin embargo si requieren una revisión de cada uno de ellos ya que generan una iluminación, sombra y reflexión especular similar al objetivo.

- Posición X. Tiene una diferencia de 59 unidades respecto al objetivo, sin embargo esto no es notable en la figura renderizada.
- Posición Y. Aquí la diferencia es de 793, es una altura muy diferente a la original, sin embargo el nivel de iluminación es el mismo en la imagen debido a que el proceso de optimización calculó una fuente de iluminación de casi el doble de potencia que la original.
- Posición Z. Tiene una diferencia de 205 unidades con respecto al objetivo y esto se puede apreciar en la sombra de la esfera, así como en la reflexión especular, sin embargo la diferencia en el nivel de iluminación de la imagen es mínimo.

- Intensidad de la fuente. Es el doble de la fuente objetivo, esto es debido a la distancia que el algoritmo propuso como origen de la fuente de iluminación, por lo cual lo compensa con una intensidad de iluminación del doble.

La búsqueda de la posición de la fuente de iluminación se muestra en la figura 6.14

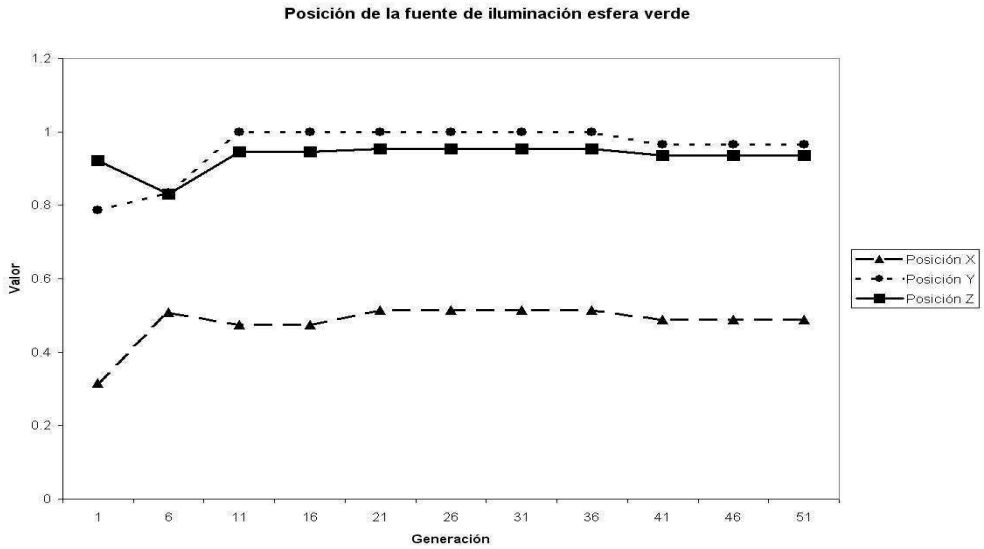


Figura 6.14: Búsqueda de los parámetros de la posición de la fuente de iluminación

Debido a que los parámetros de posición, en los cuales el algoritmo encontró un buen resultado, son de una distancia mayor a la del escenario real, el algoritmo lo compensó con una intensidad de iluminación mayor, figura 6.15.

Los parámetros de color del objeto se estabilizan en la generación 220, figura 6.16, se puede ver claramente que el parámetro que prueba los valores extremos es el del tono. Asimismo se está probando con diferentes valores de brillo y chispa en el material de la esfera, figura 6.17.

En cada uno de los experimentos se tuvo un desempeño similar al de la esfera verde, siendo el objeto más difícil de optimizar la esfera rosa, y la esfera naranja y la azul las que tuvieron una convergencia mayor, es decir, las que en más ocasiones de las 10 corridas llegaron a propuestas muy similares entre cada corrida.

6.4.1. Evaluación de resultados

Se evalúa la similitud de las imágenes usando la misma función de aptitud que se usó en el experimento anterior:

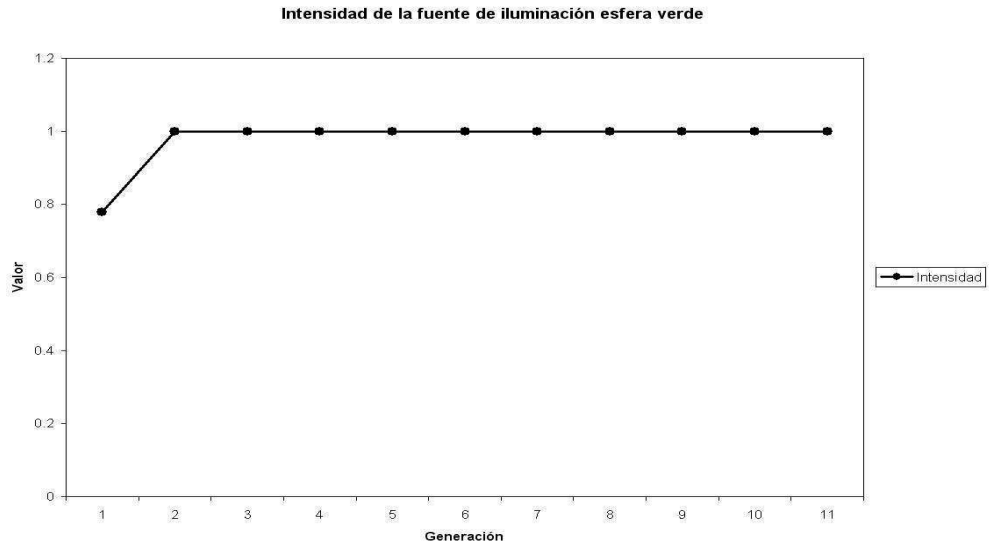


Figura 6.15: Búsqueda del parámetro de intensidad de la fuente de iluminación para la esfera verde

F01- Función de evaluación para asignar aptitud, considerando todos los píxeles de la imagen:

$$Dif = \sum_{i=0}^{i=n} |RGBObj_i - RBGTray_i| \tag{6.4}$$

Donde:

- n = Xmax * Y max
- RGBObji = Valor de cada componente del píxel i de la imagen objetivo
- RGBTrazai = Valor de cada componente del píxel i de la imagen generada por el trazador de rayos

Al colocar las dos imágenes juntas se puede apreciar el resultado de la estimación de parámetros obtenidos, ya que la imagen generada es muy similar a la imagen Objetivo, figura 6.18.

Esta función abre la imagen de referencia (imagen objetivo) y la imagen generada por el trazador de rayos con los parámetros finales del mejor individuo, en la tabla 6.5 se presentan los resultados de los mejores individuos para cada esfera.

El resultado de las imágenes generadas muestra una ligera variación, sin embargo en algunos casos la diferencia es mínima, y el color de los objetos es muy

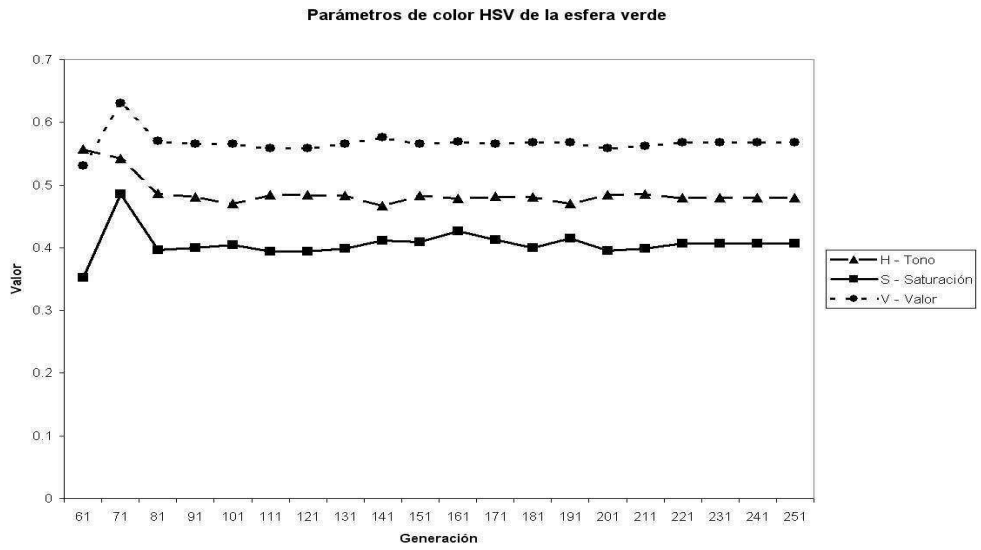


Figura 6.16: Evolución de los parámetros de color, para la esfera verde

Tabla 6.5: Desviación de cada uno de los mejores individuos con respecto a su imagen objetivo

Objetivo		Original			Estimado			% diferencia
		H	S	V	H	S	V	Promedio
Roja	esfera	0.0	0.764	0.850	0.001	0.558	0.999	6.21
	plano	0.590	0.428	0.723	0.664	0.569	0.912	
Verde	esfera	0.482	0.374	1.0	0.480	0.407	0.568	5.3
	plano	0.634	0.009	1.0	0.204	0.004	0.533	
Azul	esfera	0.655	0.825	1.0	0.654	0.863	0.720	2.63
	plano	0.522	0.301	1.0	0.513	0.275	0.680	
Amarilla	esfera	0.081	0.835	0.882	0.079	0.998	0.998	5.77
	plano	0.548	0.406	0.825	0.616	0.321	0.769	
Naranja	esfera	0.069	1.0	1.0	0.066	0.998	1.0	3.54
	plano	0.317	0.145	1.0	0.219	0.208	0.769	
Rosa	esfera	0.991	0.898	0.990	1.0	0.824	0.938	6.79
	plano	0.0	0.5	0.60	0.333	0.445	0.518	
Promedio general								5.04

similar al objetivo, específicamente en el caso de Verde, Azul, Amarilla y Naranja, figura 6.19.

Los resultados del algoritmo se compararon contra otros métodos que obtienen parámetros de renderizado y los implantan en objetos virtuales, la comparación se realizó obteniendo las imágenes de los artículos de cada método. Los resultados

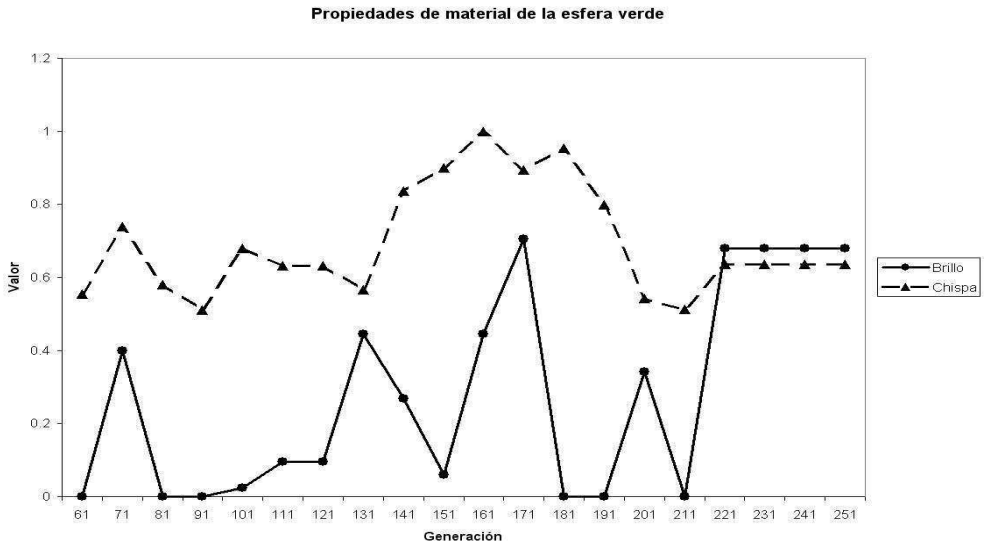


Figura 6.17: Evolución de los parámetros de brillo y chispa, para el material de la esfera verde

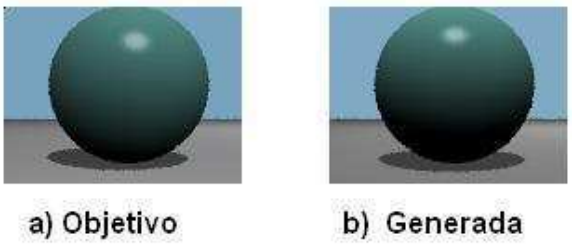


Figura 6.18: Imágenes de la esfera verde donde se pueden ver: a) imagen objetivo e b) imagen generada por la metodología.

se presentan en la tabla 6.6.

Se puede ver que el mejor desempeño de nuestro método obtiene resultados con una menor diferencia que la mayoría de los métodos comprobados, y resultados muy competitivos con el mejor método, Ward-Durn.

Al usar un conjunto de imágenes objetivo, generadas por computadora con el mismo modelo de iluminación que se usó para su extracción de parámetros, se obtuvieron imágenes con un promedio de diferencia de 5.04, este resultado brinda imágenes que al ser comparadas con las originales son difíciles de diferenciar a simple vista. En el caso de las imágenes con una diferencia de 2.63. El ojo humano

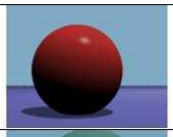
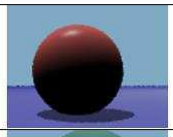
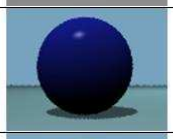
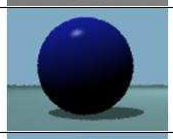
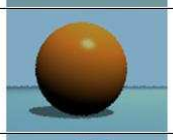
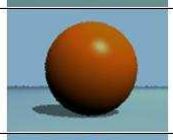
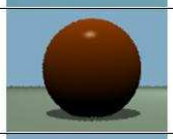
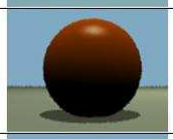
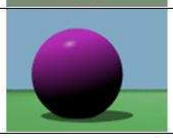
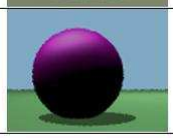
Objetivo	Original	Mejor valor estimado	% diferencia total de imagen
Roja			6.21
Verde			5.3
Azul			2.63
Amarilla			3.75
Naranja			3.54
Rosa			6.79

Figura 6.19: Comparación de las imágenes objetivo y las imágenes generadas por la metodología

tiene dificultades para encontrar diferencia en color y tono de una respecto a la otra.

La metodología aqui probada y los resultados observado muestran que mientras que la propuesta de Kawai, Painter y Cohen [18], resuelve las "mejores" posibles configuraciones para: emisividad de la fuente de luz, reflexividad de los elementos y parámetros de direccionalidad de la fuente de luz, esta propuesta ya tiene los parámetros de los objetos tales como el color, la dureza, el índice de reflexión especular y la difracción propuestos por el usuario de antemano, a diferencia de nuestra propuesta que los propone durante la extracción de los parámetros.

Por otra parte Yu, Debeverec, Malik y Hawkins [96] obtienen un conjunto de parámetros como son: el índice de reflexión difusa y el índice de reflexión es-

Tabla 6.6: Comparación de la desviación de los diferentes métodos

	Desviación			
Metodo	% R	% G	% B	Promedio
Wai Kit Addy 1	8.92	9.55	8.54	8.97
Wai Kit Addy 2	12.41	9.67	8.58	10.22
Ward	3.30	3.48	3.14	3.30
Ward-Durn	2.20	2.32	2.25	2.25
Nuestro Mejor	1.83	2.10	3.96	2.63
Nuestro Peor	7.43	5.41	7.54	6.79

pecular para los colores rojo, verde y azul. Sin embargo este método recibe las condiciones de iluminación del usuario, a diferencia de nuestra metodología que lo calcula por su cuenta.

Finalmente en el problema de la iluminación inversa abordado por Schonenman et al. así como por Elorza y Rudomin [21], buscan obtener los parámetros de: la cantidad, la posición y la intensidad de las fuentes de iluminación usadas en la escena. El objetivo es encontrar los parámetros que generen una imagen minimizando la energía usada y maximizando la iluminación.

Nuestra propuesta, obtiene un conjunto más amplio de parámetros como son: los índices de reflexión difusa, de difracción, de rugosidad, y de brillantez, así como los parámetros de la fuente de iluminación tales como posición e intensidad de la fuente. Asimismo utiliza una heurística, ED, que hasta el momento nadie ha usado para resolver este tipo de problemas. De la misma forma nuestra metodología genera un conjunto de puntos de referencia de la imagen objetivo que son dependientes de cada imagen, es capaz de generar una "huella" descriptora de la imagen de referencia, que es usada por ED para extraer un mejor conjunto de parámetros.

Capítulo 7

Conclusiones

Se ha demostrado que es factible la extracción de un conjunto de parámetros para la renderización de objetos sencillos. Entre los parámetros que es posible extraer se encuentran la posición de una fuente de iluminación así como la intensidad de la misma. El color de dos objetos contenidos en el escenario en este caso una esfera y un plano.

Se ha demostrado una metodología de extracción de parámetros de iluminación para renderizado, en donde se ha comprobado que al ser utilizada en pruebas de laboratorio ha funcionado. Asimismo la metodología ha mostrado mejores resultados que los que obtiene una persona para obtener los parámetros de una imagen de referencia, y lo hace en un factor de tiempo mucho menor.

Se propuso y probó una función que evalúa la diferencia entre dos imágenes de la misma resolución, siempre y cuando sean imágenes de las mismas dimensiones y de la misma cantidad de colores. De igual forma se tiene un algoritmo heurístico bioinspirado para generar una propuesta de parámetros de iluminación para algoritmos de renderizado. Las pruebas con el algoritmo desarrollado demostraron que es factible la extracción de los parámetros correspondientes a la posición e intensidad de la fuente de iluminación así como el color de objetos en el escenario.

Se debe destacar que la metodología obtiene mejores resultados, tabla 6.3, que una persona usando el método semiautomático, donde se puede apreciar que el mejor valor obtenido por la persona es un porcentaje de diferencia de 15.47 respecto a la imagen original. La metodología obtiene un 13.34 de diferencia en ambas implementaciones, la secuencial y la paralela. En este mismo sentido a una persona le toma una mayor cantidad de tiempo generar su mejor propuesta, 117 minutos, mientras que a la metodología en su ejecución secuencial le toma

en promedio 32 minutos y a su implementación paralela le toma únicamente 4.2 minutos en promedio.

Es importante mencionar que, a diferencia de trabajos anteriores, la metodología extrae un conjunto de parámetros más amplio, tanto los parámetros referidos a la fuente de iluminación , asi como los referidos a las características ópticas de los objetos contenidos en el escenario.

La imagen final es totalmente dependiente del modelo de iluminación usado, por lo tanto el resultado de la búsqueda está limitado por el modelo. Por lo anterior se puede demostrar que la metodología de extracción de parámetros es funcional, si el modelo de iluminación es capaz de representar todos los fenómenos físico-ópticos que se generaron en la imagen objetivo. Esto es cierto para modelos de iluminación limitados, sería conveniente para investigaciones futuras, probar con un modelo de iluminación de los más complejos.

Se demostró la viabilidad de usar el GPU para disminuir el tiempo de cálculo de un algoritmo bioinspirado de optimización. Se desarrolló un método de estimación de las propiedades básicas de un objeto con referencia a una imagen real. Se tiene una función objetiva del grado de aproximación de una imagen real con una imagen generada.

7.1. Aplicación

El presente desarrollo puede ser usado principalmente en el área de diseño de escenarios virtuales, para la obtención de una apariencia realista de los objetos a usar, así como para la colocación de las fuentes de luz y su intensidad en la simulación de escenarios reales. Esto ya se realiza pero de una forma semiautomática, usando extracción de parámetros, probando dichos parámetros y ajustando la diferencia, en el caso de esta investigación todo esto se lleva a cabo de forma automática. Una mejora en tiempo que es demostrada en el capítulo 6, en la tabla 6.4 y donde se aprecia una mejora significativa en la que una tarea que toma 117 minutos, se puede llevar a cabo en sólo cuatro, es decir en un 3 % del tiempo. Y cualquier mejora en tiempo en la industria de los gráficos 3D se traduce en ganancias millonarias.

7.2. Trabajo futuro

Futuras investigaciones en los algoritmos poblacionales pueden enfocarse para alguno de los siguientes tres puntos:

- Aplicar la totalidad de las capacidades multihilo del GPU, incluyendo las operaciones paralelas en múltiples datos, adicionalmente a la característica multihilos, para lograr un algoritmo PSO o DE totalmente paralelizado.

- La necesidad de probar mas funciones objetivo

- El uso de GPU y el nuevo modelo de programación CUDA para lidiar con la paralelización de otros algoritmos poblacionales, p.e. Estrategias Evolutivas, Programación Genética o Sistema Inmune Artificial.

La metodología es susceptible de ser mejorada en algunos aspectos, se mencionan a continuación los mas relevantes.

- Se puede realizar un preprocesamiento a la imagen con el fin de detectar las reflexiones especulares para determinar de manera exacta la posición de las fuentes de iluminación. El preprocesamiento puede incluir, una detección de bordes, así como un análisis de las partes más brillantes de la imagen objetivo. De este preprocesamiento se puede obtener el tamaño de las reflexiones especulares así como su posición en el objeto y sería posible extrapolar una región del espacio para la ubicación de la fuente, tomando como referencia el centro del objeto con la reflexión especular.

- Es recomendable programar un modelo de iluminación que considere la mayor cantidad posible de propiedades físico-ópticas para el cálculo del color de un punto, tal como los modelos de iluminación de realismo físico.

- Realizar una programación en CUDA tomando el cuenta el paso de mensajes en la arquitectura, para una mejora en el desempeño, específicamente en una reducción en el tiempo de generación de la imagen.

7.3. Productos de la investigación

Las publicaciones derivadas de este trabajo de tesis, 7.2 y 7.1 que se pueden destacar hasta el momento:

- Conacyt - Se acepto para su publicación en la revista Polibits el articulo titulado “Optical Parameter Extraction Using Diferential Evolution Rendering in the Loop”.

- JCR- Se publicó el articulo titulado “Comparative study of parallel particle swarm optimization algorithm implemented on a multitreading GPU”, en la revista Journal of Applied Research and Technology de la U.N.A.M., en diciembre de 2009

- JCR - Se envío, en marzo de 2011, a la revista Journal of Applied Research and Technology de la U.N.A.M., el articulo"Parallel Implementation of Particle Swarm Optimization and Differential Evolution Algorithms on a Multithreading GPU", actualmente se encuentra en la fase de revisión con los respectivos revisores.

- Revista Internacional - Se publicó en el Journal of Theoretical and Applied Information Technology, el articulo titulado, "Fitness function for rendering in the loop Diferential Evolution algorithm", en el volumen 21 número 2, en las páginas 128-136, en noviembre 30 de 2010.

- Congreso Internacional - Se presentó la conferencia "Real scenario parameter extraction by using PSO rendering in the loop", en el CERMA 2010, los procedings aparecen en el IEEEXplore. Se presentó el día 29 de septiembre de 2010.

- Congreso Internacional - Se presentó la conferencia "Revisión de algoritmos de renderizado óptimo" en el congreso internacional Tendencias tecnológicas del CIDETEC, octubre de 2008.

- Congreso Nacional - Se presentó la conferencia "Algoritmos de renderizado óptimo" en el congreso internacional del colegio de Ingenieros, noviembre de 2008

Tabla 7.1: Publicaciones internacionales

Tipo	Cantidad total	Relacionados
Internacionales JCR	1 en revisión	1 en revisión
Internacionales ISI	1	1
Internacionales NO ISI con Arbitraje	3	1

Tabla 7.2: Ponencias en congresos internacionales

Año de ponencia	Cantidad total	Relacionados
2007	4	0
2008	2	2
2009	2	0
2010	1	1

Referencias

[1] O. Lafontaine A. Mzoughi and D. Litaize. Performance of the vectorial processor vecsm2* using serial multiport memory. 1996. [citado en p. 68]

[2] John Amanatides. Ray tracing with cones. Volume 18, Number 3, 1984. [citado en p. 9]

[3] A. Appel. *Some techniques for shading machine renderings of solids*, volume 32. Proceedings of the Spring Joint Computer Conference, 1968. [citado en p. 7]

[4] James Arvo. Ray tracing with meta-hierarchies. 1990. [citado en p. 9]

[5] S. Baskar and P Suganthan. A novel concurrent particle swarm optimization. 2004. [citado en p. 62]

[6] M. Belal and P. El-Ghazawi. Parallel models for particle swarm optimizers. 2004. [citado en p. 61]

[7] M. Beyer and B. Lange. Rayvolution: An evolutionary ray tracing algorithm in photorealistic rendering techniques. 1995. [citado en p. 17]

[8] James F. Blinn. Models of light relfection for computer synthetized pictures. 11, 1977. Issue 2, Summer 1977. [citado en p. 53]

[9] L. Bo and et al. Parallelizing particle swarm optimization, ieee pacific rim conference on communications. In *Computers and signal Processing, PACRIM*, pages 288–291, Aug 2005. [citado en p. 62]

[10] Srinivvasa G. Narasimhan Shree K Nayar Bo Sun, Ravi Ramamoorthi. A practical analytic single scattering model for real time rendering. 24, 3, 1040-1049, 2005. [citado en p. 10, 12]

[11] W. J. Bouknight. *A procedure for generation of three-dimensional half-tone computer graphics presentations*, volume 32. Communications of the ACM, 1970. [citado en p. 7]

[12] E. Catmull. *A subdivision algorithm for computer display of curved surfaces*, volume PhD thesis. University of Utah, 1974. 623-629. [citado en p. 7, 10]

[13] J. Chang and et al. A parallel particle swarm optimization algorithm with communication strategies. *Journal of information science and engineering*, 21(4):809–818, 2005. [citado en p. 61]

[14] John P. Collomosse. Supervised genetic search for parameter selection in painterly rendering. 2007. [citado en p. 16]

[15] T. Carpenter L. Cook, R.L. Porter. Distributed ray tracing. 18 (3), 137145, 1984. [citado en p. 7, 9]

[16] NVIDIA Corporation. Cuda: Computer unified device architecture programming guide, version 2.0. 2009. [citado en p. 34, 77, 81]

[17] S. Cui and D. Weile. Application of a parallel particle swarm optimization scheme to the design of electromagnetic absorbers. *IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION*, 53(11):3616–3624, November 2005. [citado en p. 61]

[18] Ran Gal Tommer Leyvand Daniel Cohen-or, Olga Sorkine and Ying-Qing Xu. Color harmonization. 2006. [citado en p. 110]

[19] Günter Wyszecki Deane B. Judd. Color in business, science and industry. 1975. [citado en p. 134]

[20] J. Velázquez-Reyes E. Mezura-Montes and C. A. Coello-Coello. A comparative study of differential evolution variants for global optimization. 2006. [citado en p. 80, 81, 85]

[21] Rudomin Goldberg Isaac Elorza Tena Joaquin. An interactive system for solving inverse illumination problems using genetic algorithms. 1997. [citado en p. 19, 111]

[22] J. Li et al. An efficient fine-grained parallel particle swarm optimization method based on gpu-acceleration. 2007. [citado en p. 62]

[23] J. Owens et al. A survey of general-purpose computation on graphics hardware. Volume 26, Number 1, 2007. [citado en p. 60]

[24] J.F. Schutte et al. Parallel global optimization with the particle swarm algorithm. 2003. [citado en p. 61]

[25] C. Puech F. Durand, G. Drettakis. Fast and accurate hierarchical radiosity using global visibility. Vol. 18, No. 2, 1999. [citado en p. 9, 10]

[26] E. G. Finsy and J. Joosten. Maximum entropy inversion of static light scattering data for the particle size distribution by number and volume. in advances in measurements and control of colloidal processes. 1991. [citado en p. 11]

[27] Adrian Ford and Alan Roberts. Colour space conversions. 1998. [citado en p. 132]

[28] E. Fuchs and J. S. Jaffe. Thin laser light sheet microscope for microbial oceanography. 10 (2), 145-154, 2002. [citado en p. 12]

[29] Z.M. Naylor B.F. Fuchs, H. Kedem. On visible surface generation by a priori tree structures. 14, 124133, 1980. [citado en p. 7, 8]

[30] Ikekuchi K. Sakauchi M. Furukawa R., Kawasaki H. Appearance based object modeling using texture database: acquisition, compression and rendering. 3 pp. 257-266, 2002. [citado en p. 13]

[31] M. Sattler R. Sarlette G. Müller, J. Meseth and R. Klein. Acquisition, synthesis and rendering of bidirectional texture functions. 2004. [citado en p. 14]

[32] A. M. Day G. Simiakakis, Th. Theoharis. Parallel ray tracing with 5d adaptive subdivision. 1998. [citado en p. 9]

[33] Mark Gatter. Getting it right in print: Digital prepress for graphic designers. 2004. [citado en p. 132]

[34] David Geisler-Moroder and Arne Dür. Color-rendering indices in global illumination methods. Oct-Dec 2009. [citado en p. 27, 40]

[35] Reinhard Klein Gero Müller, Gerhard H. Bendels. Rapid synchronous acquisition of geometry and appearance of cultural heritage artefacts. Archaeology and Cultural Heritage VAST, 2005. [citado en p. 13]

[36] A.S. Glassner. Space subdivision for fast ray tracing. 4 (10), 1522., 1984. [citado en p. 8]

[37] H. Gouraud. *Computer display of curved surfaces*, volume 20. IEEE Transactions on Computers, 1971. 623-629. [citado en p. 7]

[38] C. Piatko P. Sanders y B. Rust H. Rushmeier, G. Ward. Comparing real and synthetic images: Some ideas about metrics. June 12-14, 1995. [citado en p. 43]

[39] T. Halfhill. Parallel processing with cuda, microprocessor report. www.MPRonline.com, on line at: www.nvidia.com/docs/IO/55972/220401_Reprint.pdf, 2008. Consulted on Nov 4th 2008. [citado en p. 62]

[40] Kwan-Liu Ma Han-Wei Shen, Ling-jen Chiang. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. 1999. [citado en p. 9]

[41] Pat Hanrahan). Ray-triangle and ray-quadrilateral intersections in homogeneous coordinates. 19. [citado en p. 7, 8]

[42] S. Harding and W. Banzhaf. Fast genetic programming and artificial developmental systems on gpus. In *Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications, HPCS*, pages 2–2, May 2007. [citado en p. 62]

[43] S. Harding and W Banzhaf. Fast genetic programming on gpus. In M. Ebner and et al., editors, *Proceedings of the 10th European Conference on Genetic Programming, ser. Lecture Notes in Computer Science*, volume 4445, pages 90–101. Springer, April 2007. [citado en p. 62]

[44] Juan Buhler Henrik Wann Jensen. A rapid hierarchical rendering technique for translucent materials. 2000. [citado en p. 10]

[45] Juan Buhler Henrik Wann Jensen. A rapid hierarchical rendering technique for translucent materials. 2002. [citado en p. 8]

[46] Marc Levoy Henrik Wann Jensen, Stephen R. Marschner and Pat Hanrahan. A practical model for subsurface light transport. 2001. [citado en p. 12]

[47] S. Feis I. Poupyrev, S. Weghorst. Non-isomorphic 3d rotational techniques. Vol. 14, num. 3, 2000. [citado en p. 8, 9]

[48] O. Lafontaine J. Jorda, A. Mzoughi and D. Litaize. Performance of the vectorial processor vec-sm2* using serial multiport memory. 1996. [citado en p. 32]

[49] David Kirk James Arvo. Fast ray tracing by ray classification. Volume 21, Number 4, 1987. [citado en p. 9]

[50] N. Jin and Y. Rahmat-Samii. Parallel particle swarm optimization and finite- difference time-domain (pso/fdtd) algorithm for multiband and wide-band patch antenna designs. *IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION*, 53(11):3459–3468, November 2005. [citado en p. 61]

[51] Andrew Woo) John Amanatides. A fast voxel traversal algorithm for ray tracing. 1988. [citado en p. 7, 8]

[52] James S. Painter John K. Kawai and Michael F. Cohen. Radioptimization - goal based rendering. 1993. [citado en p. 18]

[53] P.M. Hall John P. Collomosse. Genetic paint: A search for salient paintings. 2005. [citado en p. 17]

[54] Jeff Goldsmith John Salmon. A hypercube ray-tracer. 1989. [citado en p. 7, 9]

[55] Yun-Kung Chung Jui-yu Wu. Artificial immune system for solving generalized geometric problems: A preliminary results. 2005. [citado en p. 9]

[56] James T. Kajiya. The rendering equation. Volume 20, 1986. [citado en p. 10, 23]

[57] J. Kennedy and R. Eberhart. Swarm intelligence. 2001. [citado en p. 30, 65]

[58] W. B. Langdon and W Banzhaf. A simd interpreter for genetic programming on gpu graphics cards. In *in EuroGP, ser. LNCS*, volume 4971, pages 73–85. Springer, March 2008. [citado en p. 62]

[59] Stephen Mann Leo Dorts, Daniel Fontijne. *Geometric Algebra for Computer Sciences*. Morgan Kaufman, 2007. [citado en p. 9]

[60] Craig A. Lindleyr. *Practical Raytracing in C*. Wiley, 1992. ISBN 0471573019. [citado en p. 22]

[61] H. Ma and et al. Research on parallel particle swarm optimization algorithm based on cultural evolution for the multi-level capacitated lot-sizing problem. 2008. [citado en p. 61]

[62] Werner Hartmann Michael Haller, Stephan Drab. A real-time shadow approach for an augmented reality application using shadow volumes. 2003. [citado en p. 10]

[63] S. Mostaghim and et al. Multi-objective particle swarm optimization on computer grids. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation, GECCO'07*, pages 869 – 875. ACM, July 2007. [citado en p. 62]

[64] Wai Kit Addy Ngan. *Acquisition and Modeling of Material Appearance*. PhD thesis, Massachusetts Institute of Technology, 2003. [citado en p. 14]

[65] J. Nickolls and et al. Scalable parallel programming with cuda. *ACM Queue*, pages 40–53, March/April 2008. On line at: http://mags.acm.org/queue/20080304/?u1=texterity. [citado en p. 61, 62, 78, 88]

[66] Ricardo Cancho Niemietz. Schematic representation on how tv signals are gamma-compensated when broadcasted. 2008. [citado en p. 137]

[67] NVIDIA. Nvidia cuda instalation and verification. 2008. [citado en p. 63]

[68] U.S. Departament of Energy. Color rendering index and leds. January 2008. [citado en p. 27, 40]

[69] Y. Willems P. Dutre, F. Suykens. Optimized monte carlo path generation using genetic algorithms. Volume 19, Number 3, 1998. [citado en p. 7, 8, 9, 10, 11, 17]

[70] K. Parsopoulos and et al. Multiobjective optimization using parallel vector evaluated particle swarm optimization. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications (AIA)*, pages 869 – 875, July 2007. [citado en p. 62]

[71] Danny Pascale. A review of rgb color spaces... from xyy to r'g'b'. 2003. [citado en p. 133]

[72] B-T. Phong. Illumination for computer generated pictures. 311-316, 1975. 623-629. [citado en p. 52]

[73] D. Robilliard and et al. Population parallel gp on the g80 gpu. In *Lecture Notes in Computer Science*, volume 4971, pages 98–109. Springer Berlin / Heidelberg, 2008. [citado en p. 62]

[74] A. Rowbottom. Evolutionary art and form. 1999. [citado en p. 17]

[75] Jonathan Sachs. Color magnament. 2008. [citado en p. 137]

[76] Norbert Schanda, János; Sándor. Visual colour-rendering experiments. 2005. [citado en p. 26]

[77] I. Schoeman and A. Engelbrecht. A parallel vector-based particle swarm optimizer, adaptive and natural computing algorithms. In *Proceedings of the International Conference in Coimbra*, pages 268–271. Springer, 2005. [citado en p. 62]

[78] Smits Brian Arvo James Schoeneman Chris, Dorsey Julie and Greenberg Donald. Painting with light. In *SIGGRAPH '93 Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, July 1993. [citado en p. 19]

[79] Y. Shi and R. Eberhart. A modified particle swarm optimizer. 1998. [citado en p. 31]

[80] Ken Shoemaker. Animating rotation with quaternion curves. Volume 19, Number 3, 1984. [citado en p. 7, 8, 9]

[81] Daniel Thalmann Srikanth Bandi. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. Vol. 14, num. 3, 1995. [citado en p. 7, 8]

[82] Craig Donner Ravi Ramamoorthi Shree K. Nayar HenrikWann Jensen Srinivasa G. Narasimhan, Mohit Gupta. Acquiring scattering properties of participating media by dilution. 2006. [citado en p. 12, 14]

[83] R. Storn and K.V. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. 11(4), 1997. 341-359. [citado en p. 28]

[84] Unemi T. [citado en p. 17]

[85] Barbara Fritchman Thompson. Astronomy hacks: Tips and tools for observing the night sky. 2005. [citado en p. 26]

[86] Marko Tkalcic. Colour spaces - perceptual, historical and applicational background. 2001. [citado en p. 135]

[87] Latham W. Todd, S. Evolutionary art and computers. 1992. [citado en p. 17]

[88] Latham W. Todd, S. The mutation and growth of art by computers. 1999. [citado en p. 17]

[89] T. Unemi. Sbart2.4: Breeding 2d cg images and movies, and creating a type of collage. The Third International Conference on Knowledgebased Intelligent Information Engineering Systems, 1999. [citado en p. 17]

[90] L. Veach, E. Guibas. Metropolis light transport. *Computer Graphics (Proceedings of SIGGRAPH 1997)*, 16 6576, 1997. [citado en p. 7, 10, 17]

[91] Nicholas J. Wade and Josef Broek. Purkinje's vision. 2001. [citado en p. 26]

[92] T. Whitted. An improved illumination model for shaded display. 343-349, 1980. 623-629. [citado en p. 7]

[93] Remo Ziegler Addy Ngan Wojciech Matusik, Hanspeter Pfister and Leonard McMillan. Acquisition and rendering of transparent and refractive objects. 2002. [citado en p. 14]

[94] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. 1997. [citado en p. 77]

[95] Donald Hearn y M. Pauline Baker. *Gráficas por computadora*. Prentice Hall, segunda edition, 1995. [citado en p. 21, 137]

[96] Jitendra Malik Yizhou Yu, Paul Debevec and Tim Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photograps. 1999. [citado en p. 18, 110]

[97] Yizhou Yu. Modeling and editing real scenes with image-based techniques. 2000. [citado en p. 18]

Apéndice A

Mejora en el tiempo

En una búsqueda de imágenes cada vez más realistas, la tecnología de las gráficas por computadora se ha transformado y ha evolucionado desde las primeras gráficas de líneas hasta imágenes que son idénticas a una fotografía o en su caso de videos reales grabados por cámaras de TV. El costo de estas imágenes, cada vez más realistas, se ha reflejado en el poder de procesamiento necesario para generar dichas imágenes, donde una sola computadora del siglo 21 puede tardarse días para una sola imagen. Y un grupo de computadoras (Granja), puede tomarse horas para una sola imagen. Lo cual trae a la mente un sencillo cálculo, una imagen toma varias horas de cálculo, se necesitan 24 imágenes por segundo para presentar una sensación de movimiento continuo a los espectadores de una película de cine. Las películas duran en promedio 2 horas, entonces:
24 imágenes x 60 segundos x 120 minutos (2 horas) = 172,872 imágenes (im).

Ahora interviene la resolución de la imagen, para una resolución de 1024 X 768 (es decir 1024 pixeles horizontales por 768 pixeles verticales) = 786,432 pixeles por imagen (ppi). Además se debe considerar que para una imagen fotorrealista es necesario usar color verdadero que está definido como color con 24 bits (3 bytes).

Multiplicando 172,872 imágenes x 786,432 ppi x 3 bytes = 407,856,218,112 bytes de información generada. O sea más de 407 Gigabytes de información.

Como se puede apreciar, cualquier reducción en el tiempo de renderización de una imagen impacta en el tiempo total de renderización de una animación o película que use gráficas generadas por computadora, los métodos usados para generar imágenes cada vez más realistas van desde el mapeo de texturas realistas hasta la medición de características de las texturas reales y su modelado en tres

dimensiones.

Desde el surgimiento de las computadoras el problema de la graficación ha sido un reto que gradualmente ha ido aumentando desde las primeras imágenes presentadas en un monitor.

El caso que nos ocupa es el del software de renderzado, ya que al generar la imagen, el modelo de iluminación debe considerar la corrección gamma que el sistema en conjunto va a necesitar. Adicionalmente cada elemento que participe en la adquisición, almacenamiento, comparación y generación de una imagen necesita considerar si alguna de sus partes está llevando a cabo la corrección gama y en que forma, así como sus respectivos índices de corrección.

Apéndice B

Renderizado

El renderizado se define como el proceso de la generación de una imagen a partir de los datos de un escenario virtual y tiene los siguientes pasos:

1. Lectura de datos de la memoria de la computadora
2. Renderizado
3. Despliegue de la imagen

El proceso de renderizado se puede dividir en tres grandes etapas conceptuales:

1. Etapa de aplicación
2. Etapa Geométrica
3. Etapa de Rasterizado

Cada una de las etapas antes mencionadas tiene su tarea específica a realizar, y cada una de dichas etapas, tiene una gran cantidad de cálculos asociados para obtener los resultados que de ella se esperan, a continuación se explica más a detalle cada una de las tareas.

B.0.1. Etapa de aplicación

Su responsabilidad principal es enviar primitivas al hardware de gráfico. Aunque evidentemente realiza otras tareas tales como: la interpretación de los datos leídos del archivo del escenario virtual, la conformación de la escena, la detección de colisiones, las técnicas de aceleramiento, la interacción de los personajes u objetos en escenario, la animación, la comunicación con otros procesos de sistema,

entre los más importantes. De todas las tareas antes mencionadas, se pueden destacar 3.

1. Definición del objeto
2. Conformar escena
3. Definir vista de referencia

B.0.2. Etapa geométrica

La etapa geométrica tiene la responsabilidad de hacer un procesamiento intenso en operaciones algebraicas asociadas a espacios afines y como se relacionan los objetos, el observador y mundo virtual, estas operaciones generalmente se realizan con vectores y matrices. Los vectores representando los elementos básicos que forman a los objetos virtuales, y las matrices representando las operaciones a realizar sobre dichos objetos.

La etapa geométrica recibe elementos que describen a los puntos básicos y como se relacionan para dar forma a los objetos así como su interrelación para conformar al escenario, pero dicho escenario no existe en la computadora hasta que la etapa geométrica realiza su análisis y su procesamiento. La etapa geométrica se encarga entonces de manejar los diferentes sistemas de coordenadas que serán necesarios para obtener la representación geométrica del escenario. Estos sistemas de coordenadas manejan diferentes aspectos de un objeto.

Por ejemplo el sistema de coordenadas locales del objeto tiene que ser relacionado con el sistema de coordenadas globales del escenario virtual, para hacer esto la etapa geométrica tiene que relacionar dos espacios afines a través de operaciones geométricas, muchas veces llevados a cabo, por medio de multiplicaciones de matrices. Pero esto no esto todo, ya que se tiene que pasar en diferentes sistemas de coordenadas, figura B.1, las cuales son:

1. Locales
2. Mundiales
3. Vista
4. Pantalla
5. Dispositivo

Lo anterior es para lograr que todos los elementos del escenario virtual se manejen en el mismo sistema de coordenadas, pero adicionalmente se tienen que

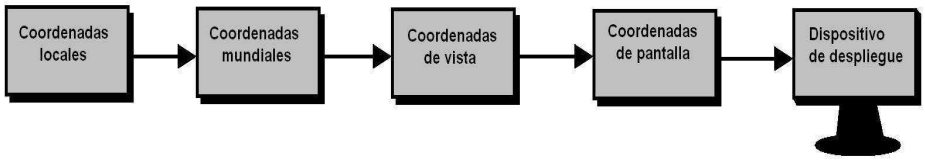


Figura B.1: Transformaciones de coordenadas mundiales a coordenadas de dispositivo

realizar otros cálculos; los cuales también son llevados a cabo por la parte geométrica, tales como: Mover objetos (multiplicación de matrices), mover cámara (multiplicación de matrices), calcular la iluminación en los vértices del triángulo (multiplicación de vectores), proyectar en la pantalla y recorte, entre otros.
La etapa geométrica se puede separar básicamente en tres etapas:

- Remoción de polígonos ocultos.
- Recorte a vista de volumen 3D.
- Retirar superficies ocultas.

B.0.3. Etapa de rasterizado

Tarea principal: tomar la salida de la etapa de geometría y transformarlo en píxeles en la pantalla. Tiene que hacerse una adecuación entre formas geométricas perfectas y las formas que es capaz de desplegar un sistema limitado como es el dispositivo de despliegue, ya que dicho dispositivo no es capaz de presentar una línea diagonal completamente recta, ni círculos perfectos ya que la resolución del dispositivo tiene una limitante dada por la forma y organización de sus elementos básicos. Esta limitante se debe a la forma en la que están organizados los elementos básicos (pixels - picture elements) que, eventualmente, darán forma a la imagen final, dichos elementos están organizados en triadas RGB(Red, Green, Blue - rojo, verde y azul), las cuales a su vez están en arreglos matriciales equivalentes a la resolución máxima del dispositivo de despliegue (p.e. 1024 x 768). Presentar un círculo con triadas de elementos en una matriz no es sencillo, de hecho al desplegar los círculos (entre otras formas geométricas) en el dispositivo genera una imagen con bordes imperfectos de una apariencia discontinua, lados segmentados (forma aserrada). La tarea de la etapa de rasterizado es presentar imágenes lo más cercanas a estas formas perfectas entregadas por la etapa geométrica. También se encarga de agregar texturas y muchas otras operaciones por píxel. La visibilidad se resuelve aquí, ordenando las primitivas en la dirección z.

El proceso de rasterizado se puede subdividir en etapas de la siguiente forma:

- Rasterización
- Sombreado
- Despliegue de la imagen obtenida

A continuación se muestran las nueve etapas que forman parte del proceso de renderización:

1. Definición del objeto
2. Conformar escena
3. Definir vista de referencia
4. Remoción de polígonos ocultos
5. Recorte a vista de volumen 3D
6. Retirar superficies ocultas
7. Rasterización
8. Sombreado
9. Despliegue de la imagen obtenida

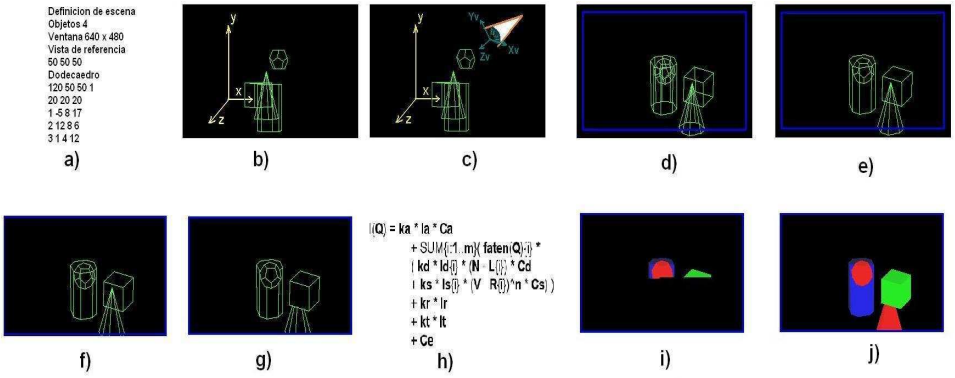


Figura B.2: Etapas del proceso de renderizado: a) Definición de escena, b) Conformación de la escena, c) Definición del punto de vista, d) Antes de eliminar los polígonos ocultos, e) Polígonos ocultos eliminados, f) Recorte a volumen 3D, g) Eliminación de superficies ocultas, h) Modelo de iluminación, i) Rasterización, j) Despliegue de la imagen.

Dependiendo del algoritmo de renderizado que se use, las etapas mencionadas se pueden ejecutar en la secuencia descrita o en algunos casos algunas etapas

se pueden ejecutar al mismo tiempo. El algoritmo de buffer de profundidad (ZBuffer), ejecuta en secuencia cada una de las etapas. En el caso del algoritmo de traza de rayos, que es el que se va a usar para esta tesis, las etapas de remoción de polígonos ocultos, recorte a vista de volumen 3D, retirar superficies ocultas, rasterización, y sombreado, se calculan todas ellas en la misma etapa, que es la del cálculo de la traza del rayo.

Cada una de las etapas de renderizacion es de gran importancia para la adecuada generación de la imagen, sin embargo la etapa donde el presente trabajo va a tener impacto es en la aplicación del modelo de iluminación en la etapa de rasterizado.

En el proceso de rasterizado la aplicación del modelo de iluminación (sombreado) es lo que indica los parámetros físico ópticos a usar para generar la imagen final. El modelo de iluminación se define como: el conjunto de cálculos necesarios para obtener el color concreto que le corresponde a un punto.

Apéndice C

Modelos de color

C.1. Modelo de color

El modelo de color define la forma en la que se van a manejar los colores y los tonos de la imagen a generar. Existen varios pero es posible destacar 3: CMY, RGB, y HSV. Cada uno de ellos se desarrolló con diferentes objetivos en mente.

Síntesis aditiva y sustractiva

En la teoría del color se tienen principalmente dos modelos de síntesis de color, estos modelos son el modelo aditivo y el sustractivo.

En la síntesis aditiva se empieza con un color negro, cero intensidad luminosa, y se agregan componentes de color para conseguir una mayor emisión de luz, en este modelo al agregar todas las componentes de color con la mayor intensidad se consigue la máxima emisión de luz con un color blanco, uno en intensidad luminosa. Por ejemplo, usando un modelo RGB, rojo, verde y azul, como componentes básicas del modelo, se puede tener un color blanco al agregar todos los componentes de luz a su máxima intensidad, fig 2.6. Normalmente este sistema aditivo se usa para generar imágenes con sistemas de emisión de luz, como son los monitores de computadoras, los cañones proyectores y los sistemas de televisión que usan tecnologías como el plasma, el cristal liquido y los basados en LEDs.

El sistema de síntesis sustractivo parte de un fondo blanco al que se le hace incidir una luz blanca. Cuando se colorea una parte o todo el fondo blanco con un color, el color absorbe una parte de la luz emitida por la fuente y reemite la energía restante, de esta forma las componentes emitidas forman el color percibido por el ojo, si aplicamos un tinte rojo al lienzo blanco, el color emitido será rojo. A medida que agregamos colores al lienzo, la cantidad de luz emitida disminuye hasta que no se emite luz y se tiene un color negro. Este sistema de síntesis

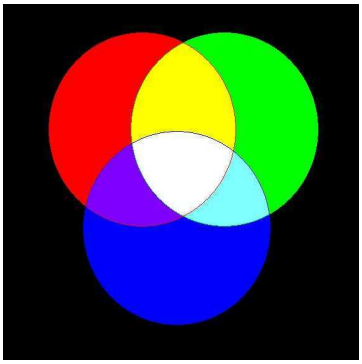


Figura C.1: Síntesis de color, Aditiva

sustractiva es muy usado en la pintura artística como el óleo, acuarela, carbón, etc. y en la industria digital es usado en las impresoras y los plotters.

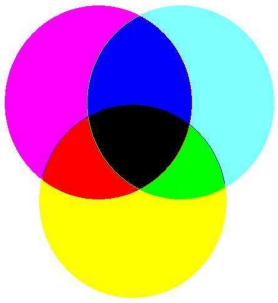


Figura C.2: Síntesis de color, sustractiva

Modelo CMY

El modelo CMY [33] debe su nombre a los colores Cyan, Magenta y Yellow (Acua, Rosa-violeta, amarillo) que son los colores básicos para crear color en un modelo sustractivo, esto significa que cada vez que se agrega una componente de color, la luz emitida por la página donde se encuentra el color es disminuida, este modelo de color es el usado en las copias impresas y en las impresoras.

CMY es muy fácil de implementar, pero la adecuada transferencia del RGB al CMY es muy difícil. CMY es dependiente del dispositivo, no lineal con respecto a la percepción visual y razonablemente intuitivo [27]. Con diferentes dispositivos tiene características distintas de transferencia entre valores CMY y la cantidad de tinta a colocar sobre el papel. Existen algunas formas simples de transformar

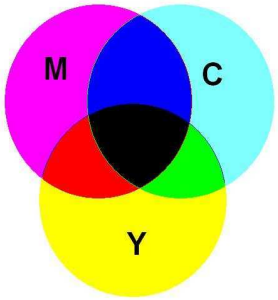


Figura C.3: Modelo de color CMY, Cyan, Magenta, Yellow

colores RGB a colores CMY para impresión, sin embargo estas transformaciones en raras ocasiones generan en impresión los colores observados en pantalla.

RGB a CMY

$$\begin{aligned} Cyan &= 1 - Red \\ Magenta &= 1 - Green \\ Yellow &= 1 - Blue \end{aligned} \tag{C.1}$$

CMY a RGB

$$\begin{aligned} Red &= 1 - Cyan \\ Green &= 1 - Magenta \\ Blue &= 1 - Yellow \end{aligned} \tag{C.2}$$

Existe un método más preciso el cual consiste en calcular los valores triestímulo CIE de los valores RGB de los pixeles de la imagen y usar estos valores como objetivo para una transformación similar hacia los valores de cromaticidad CMY del dispositivo y la adecuada función de transferencia de los valores correspondientes CMY.

Modelo RGB

El modelo RGB obtiene su nombre de los colores Red Green y Blue [71], del inglés para el Rojo el Verde y el Azul. El modelo RGB es un modelo de síntesis aditiva, de forma que cuando se desea generar un color primario se aumenta la componente del color primario a mostrar, por ejemplo rojo, y se aumenta la intensidad para obtener un color más intenso desde el color negro hasta el rojo intenso. Sin embargo si se desea obtener un color secundario como el amarillo se deben incrementar en forma conjunta dos colores primarios como son el rojo y el verde. En las gráficas por computadora el modelo RGB es muy popular ya que

son usados por los dispositivos de despliegue de imagen como son: los monitores de tubos de rayos catódicos (CRT - Cathodic Ray Tube)o los de formato digital como son los monitores de crital liquido (LCD), los de plasma y los cañones proyectores son controlados por un sistema digital que permite hasta 256 niveles de intensidad en cada componente de color, esto proporciona hasta 256 niveles diferentes de rojo verde y azul, y asimismo 256 niveles de colores secundarios. La cantidad de colores que es capaz de mostrar un modelo como éste es de 256r x 256g x 256b = 16,777,216 diferentes colores. De acuerdo con Judd, Deane B. y Wyszecki, Günter [19], el ojo humano sólo puede percibir entre 8 y hasta 10 millones de colores, así que el modelo RGB de 256 niveles por componente es más que suficiente. Esto quiere decir que de los 16 millones de colores que es capaz de desplegar el modelo RGB, el ojo humano no puede percibir casi la mitad de ellos. Sin embargo, tanto las modernas tarjetas de gráficos, así como los modernos monitores de computadora, si son capaces de desplegarlos. Adicionalmente a esto, el ojo humano es más sensible a unas longitudes de onda que a otras, y esto, como ya se mencionó, es dependiente de si es iluminación nocturna, o diurna.

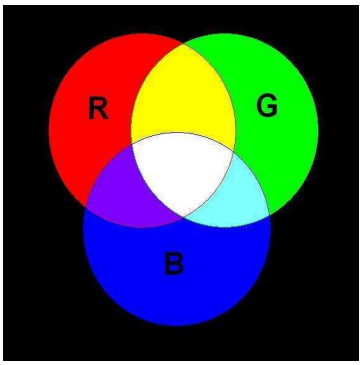


Figura C.4: Modelo de color RGB, Red, Green, Blue

El espacio de color del modelo RGB se puede visualizar como un cubo en el que cada uno de los ejes corresponde a la intensidad de los colores primarios RGB, figura C.5. En el extremo de mayor intensidad para todas las componentes se encuentra el color blanco, cuando rojo=verde=azul=1, y en el extremo opuesto se encuentra el color negro, cuando rojo=verde=azul=255. RGB se usa frecuentemente en la mayoría de las aplicaciones de computadora debido a que no necesita realizar una transformación de los valores almacenados en memoria, para su transferencia hacia la pantalla. Por esta razón es el modelo de color para la mayoría de las aplicaciones.

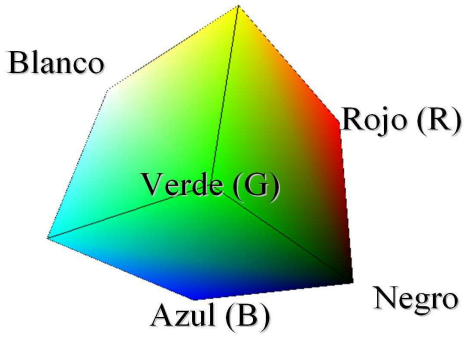


Figura C.5: Espacio de color RGB

Modelo HSV

Es una transformación no lineal del espacio de color RGB. En el espacio de color HSV (Hue Saturation Value, Matiz, saturación y valor) [86] el color es definido como una posición en un plano circular alrededor del eje de valor. El Matiz (Hue) es el ángulo de un punto nominal alrededor del círculo del color, con un valor desde 0 hasta 360, cada valor corresponde a un color por ejemplo rojo = 0, amarillo = 60 y verde = 120, figura C.6.

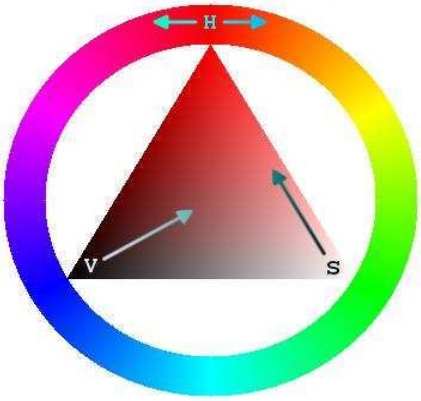


Figura C.6: Rueda de ajuste HSV

La saturación es el radio (distancia) desde el eje central de brillo negro-blanco, los valores posibles van del 0 a 100 %, a este parámetro también se le conoce como pureza del color. El valor del color representa la altura en el eje blanco-negro con valores de 0 a 100 % negro = 0, color saturado = 100. Existen modelos de color similares al HSV entre los que se encuentran el HSL (Hue Saturation Lightness),

HSI (Hue Saturation Intensity) y el HCI (Hue Chroma Intensity). Todos los modelos son transformaciones lineales del RGB y como tal comparten algunas de sus limitaciones, sin embargo, el espacio de búsqueda es diferente ya que tiene la forma de un cono, y esto puede ayudar en la búsqueda de soluciones diferentes al usar heurísticas para exploración del espacio de color, figura C.7.

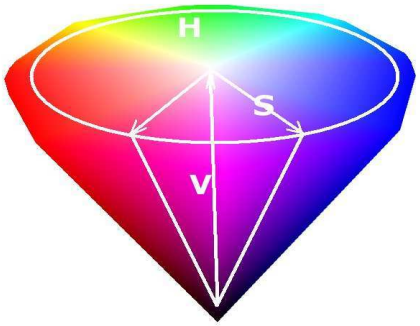


Figura C.7: Espacio de color HSV, se conceptualiza como una forma cónica

Apéndice D

Corrección gamma

D.1. Corrección gamma

Un problema que se asocia al despliegue de las intensidades calculadas es la característica no lineal de los dispositivos de despliegue [95]. Todos los modelos de iluminación para gráficas por computadora producen un rango lineal de intensidades. Por ejemplo, el color RGB (128, 128, 128) que se obtiene a partir de un modelo de iluminación representa la mitad del rango de 0 a 255 de cada una de las componentes. Estas intensidades se despliegan o se almacenan en un archivo de imágenes como valores enteros, normalmente con un byte para cada componente. El archivo también es lineal. Sin embargo, un monitor de video es un dispositivo no lineal. Al establecer los valores de voltaje para el tubo de rayos catódicos proporcionales a los valores de píxel lineales, entonces las intensidades desplegadas cambian de acuerdo a la curva de respuesta del dispositivo. De esta forma cuando la imagen con valores lineales se presenta en un dispositivo con respuesta no lineal se presente una atenuación en la iluminación general de la imagen, dicha atenuación es exponencial respecto a la curva de respuesta del dispositivo, figura D.1, [66].

Para corregir los factores no lineales del dispositivo de despliegue, los sistemas de gráficas por computadora utilizan una corrección gamma de la intensidad[75]. La corrección gamma es una operación no lineal que se usa para codificar y decodificar la luminancia en las imágenes y se puede definir como:

$$V_{Salida} = V_{Entrada} \tag{D.1}$$

Donde los valores de entrada y de salida deben ser valores reales positivos, con valores de 0 a 1. A un valor gamma < 1 se le denomina gamma de codificación, ya que al usarlo como una potencia del valor de color a transformar, el valor disminuye, es decir se realiza una compresión gamma. Por otra parte un valor

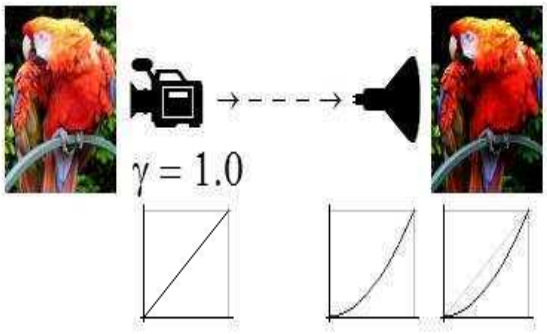


Figura D.1: Imagen lineal mostrada en un dispositivo con comportamiento no lineal y sin corrección gamma

.

gamma > 1, se le llama gamma de decodificación y al aplicarlo en la conversión de un color se tiene una expansión gamma.

Cuando se aplica la corrección gama a los valores entregados al dispositivo de despliegue la imagen corrige la luminancia y se ve fiel a la original, figura D.2. Los diferentes dispositivos de despliegue tienen distintos valores de corrección

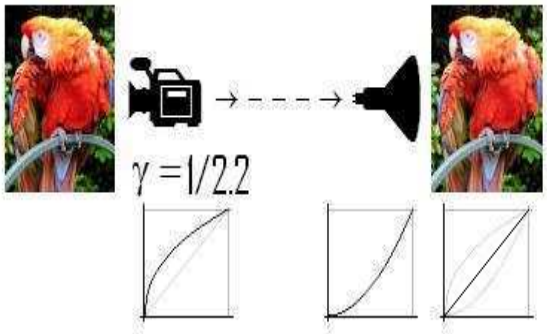


Figura D.2: Imagen lineal mostrada en un dispositivo con comportamiento no lineal y con corrección gamma

gamma. Incluso los sistemas operativos usan diferentes niveles de corrección. Por ejemplo una imagen generada en un sistema operativo Machintosh, al ser pasada a un sistema Windows debe ser corregida con un factor gama para que la imagen se vea igual que la original.

Corrección gamma en las gráficas por computadora

En las computadoras es posible manipular hasta cuatro elementos para lograr una adecuada codificación gamma, para corregir la imagen a ser mostrada:

- El software de renderizado puede entregar valores lineales, o puede entregar valores con corrección gamma adecuada para el dispositivo y el sistema operativo.
- Los valores de intensidad del pixel en una imagen de forma que representen los valores gamma corregidos en lugar de una codificación lineal. Esto es muy común en los archivos de video digital para evitar la etapa de decodificación gamma.
- Adaptadores modernos de despliegue cuentan con sistemas de calibración dedicados, los cuales pueden ser cargados con tablas de búsqueda que contengan valores apropiados de corrección gamma, antes de que lleguen al Convertidor Digital Analógico.
- Los dispositivos de despliegue actuales permiten al usuario modificar su comportamiento gamma, que al igual que la exterior, es un ajuste gamma por hardware.

El caso que nos ocupa es el del software de renderizado, ya que al generar la imagen, el modelo de iluminación debe considerar la corrección gamma que el sistema en conjunto va a necesitar. Adicionalmente cada elemento que participe en la adquisición, almacenamiento, comparación y generación de una imagen necesita considerar si alguna de sus partes está llevando a cabo la corrección gamma y en que forma, así como sus respectivos índices de corrección.

Apéndice E

Otros experimentos

E.1. Problemas de comunicación

Cuando se trabaja con unidades de procesamiento gráfico (GPU), se presuponen muchas ideas, una de ellas es que el aumento de unidades de procesamiento supone una mejora lineal con respecto a las tareas a dividir. Como es bien sabido en el área de procesamiento paralelo esto no es necesariamente cierto. En el caso de nuestra investigación el GPU usado es un modelo con 32 núcleos de procesamiento, todos los experimentos se realizaron ahí y se demostró que se tuvo una aceleración equivalente 30 veces la ejecución secuencial, sin embargo al ejecutar los mismos algoritmos en un GPU con 240 núcleos se esperaba una aceleración de aproximadamente 200 veces respecto a la ejecución secuencial, sin embargo la aceleración fue solo ligeramente superior a la ejecución en la arquitectura de 32 núcleos, es decir, la aceleración en la arquitectura de 240 núcleos fue de solo 46 veces la de la ejecución secuencial. Al revisar el desempeño del GPU con el visual profiler de la arquitectura se observó que un gran tiempo consumido por el GPU se encontraba en la comunicación interna del GPU, en el área de memoria interna del dispositivo, no en los registros, sino en la memoria compartida de los multinucleos. Esto nos lleva a teorizar acerca de la forma de escribir los códigos para el GPU, en este caso el código debe ser escrito pensando en que se van a generar colisiones de comunicación debido al comportamiento del programa ejecutado en el GPU, en este caso una heurística bioinspirada de carácter poblacional. NVIDIA ya contempla estas dificultades en la comunicación y las expone en los manuales de las arquitecturas mas recientes, y asimismo las ataca definiendo un conjunto de funciones para la adecuada administración de la comunicación en caso de este tipo de problemas de desempeño.

Lista de abreviaciones

Abreviación	Descripción	Usado
3D	Tres dimensiones	2.3
4D	Cuatro dimensiones	2.3
ASCA	Analizador de control y estabilidad para aeroplanos de la marina	1.1
BFGS	Método de análisis Broyden Fletcher Goldfarb Shanno	2.3.2
BMP	Mapa de Bits	4.2.1
BRDF	Función bidireccional de distribución de la reflectancia	2.3.2
BTF	Función de transporte de luz bidireccional	2.3
CIC	Centro de Investigación en Computación	4
CIDETEC	Centro de Innovación y Desarrollo Tecnológico en Cómputo	7.3
CPU	Unidad de procesamiento central	3.6
CMY	Modelo de color aqua, margenta y amarillo	C.1
CR	Taza de cruce	3.5.1
CRI	Índice de color de renderizado	3.4.1
CRT	Tubo de rayos catódicos	C.1
CUDA	Arquitectura de dispositivo de computo unificada	3.6
ED	Algoritmo de Evolución Diferencial	3.5.1
G	Generación	3.5.1
GPU	Unidad de Procesamiento Gráfico	2
HCI	Modelo de color de matiz, cromaticidad e intensidad	C.1
HSI	Modelo de color de matiz, saturación e intensidad	C.1
HSL	Modelo de color de matiz, saturación y luminosidad	C.1
HSV	Modelo de color de matiz, saturación y valor	C.1
IEEE	Instituto de ingenieros en electricidad y electrónica	7.3
IPN	Instituto Politécnico Nacional	4

Abreviación	Descripción	Usado
ISI	Instituto para Información la Científica	7.3
JCR	Reporte de citas en revistas	7.3
JPG	Grupo de unión de fotógrafos	4.2.1
LCD	Despliegue de cristal liquido	C.1
LED	Diodo de emisión de luz	3.4.1
MIT	Instituto Tecnológico de Masachusets	1.1
PC	Computadora Personal	2.3
PSO	Algoritmo de optimización por cúmulo de partículas	3.5.2
RGB	Modelo de color rojo, verde y azul	C.1
SIMT	Instrucción única múltiples hilos	3.6
UNAM	Universidad Nacional Autonoma de México	7.3
VRML	Lenguaje de modelado de realidad virtual	3.3

Glosario

Término	Descripción
Algoritmo evolutivo	Es un método de optimización y búsqueda de soluciones basado en algunos postulados de la evolución biológica, estos algoritmos son parte de la rama de la Inteligencia Artificial, se usan, en su mayoría, en espacios de búsqueda extensos y no lineales.
Algoritmo poblacional	Es un método donde se mantiene un conjunto de agentes que representan posibles soluciones, las cuales se mezclan, y compiten entre sí, de tal manera que las mejores son capaces de prevalecer a lo largo del tiempo.
Color difuso	Es el color más significativo de un objeto, es el color que un objeto revela bajo la iluminación de una fuente de luz blanca, sin ningún otro aporte de luz.
ED	Evolución Diferencial, es un método de optimización que pertenece a la categoría de algoritmos de computación evolutiva y que se aplica a la resolución de problemas complejos.
Embebido	En este caso particular se refiere al nivel de implantación de un algoritmo heterogéneo en programación CUDA, en la cual la mayor parte del algoritmo se ejecuta dentro del GPU y prácticamente nada se ejecuta en el CPU, salvo entrada y despliegue de datos.
Estimación de parámetros	Consiste en el cálculo aproximado de un parámetro desconocido, a partir de una gran variedad de métodos, para este caso es a partir de heurísticas bioinspiradas.

Término	Descripción
Estimación Montecarlo	Es un método no determinístico usado para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud, actualmente es parte fundamental de los algoritmos de traza de rayos. En la renderizacion se usa para eliminar el efecto de diente de sierra en las imágenes sintetizadas, de esta forma la resolución aparente de la imagen es mayor que la real.
Extracción de parámetros	Consiste en la medición de parámetros de un sistema. En este caso se refiere a los parámetros ópticos de los objetos contenidos en los escenarios a analizar, dichos parámetros pueden ser desde el color del objeto hasta el valor de dispersión de la luz de un fluido o gas.
Función objetivo	Es la función que se desea optimizar, maximizar o minimizar, en este caso se trata de minimizar la diferencia entre dos imágenes, la imagen objetivo y la imagen generada por la metodología.
Goniómetro	Es un instrumento de medición de ángulos, en este caso se usa un fotogoniómetro el cual mide la intensidad de la luz a diferentes ángulos, se usa para medir la curva de distribución luminosa.
GPU	Unidad de procesamiento gráfico, es una tarjeta de aceleramiento gráfico usada para mejorar el desempeño en los cálculos asociados al proceso de renderizado de imágenes de dos o de tres dimensiones.
Heurística bioinspirada	Estrategia o método usado para facilitar la solución de problemas difíciles, basada en las observaciones del comportamiento de sistemas biológicos tales como la evolución o el comportamiento de seres vivos como pueden ser los peces o las hormigas.
Hilo	Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea. Un hilo de ejecución o subproceso es una característica que permite a una aplicación realizar varias tareas de forma concurrente. Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Término	Descripción
Iluminación inversa	Se refiere a un problema en el cual se busca le mejor configuración de las condiciones de iluminación, para iluminar la escena por medio de la minimización de la diferencia entre la escena renderizada y la imagen deseada.
Índice de reflexión	Es determinado por la luz que se refleja de cada superficie, y depende de las propiedades ópticas de dicha superficie, es decir es el diferencial de la cantidad de energía incidente que se refleja y la cantidad que se absorbe.
Índice de refracción	Es una medida que determina la reducción de la velocidad de la luz al propagarse por un medio homogéneo. Es decir, es el cambio de la fase por unidad de longitud, esto es, el número de onda en el medio k será n veces más grande que el número de onda en el vacío k_0.
Mapa de bits	Se refiere a un arreglo matricial de pixeles donde los renglones y las columnas describen la posición de cada píxel en una imagen. El valor almacenado en una celda correspondiente a un renglón n por una columna m, donde se almacena la intensidad de iluminación de un punto de la imagen correspondiente a la ubicación m x n.
Radiosidad	Es un conjunto de técnicas para el calculo de la iluminación que tratan de resolver el problema de síntesis de imagen de la forma más realista posible en el campo de los gráficos en tres dimensiones. El problema del transporte de la luz se modela considerando que cada fuente luminosa emite un número enorme de fotones, que rebotan al chocar contra una superficie describiendo una cantidad de trayectorias a simular en una computadora. Debido a que esto es muy difícil de simular se recurre a algunas técnicas, una de dichas técnicas empleadas en el cálculo de la radiosidad es por medio del Método Montecarlo.
Rasterizado	Es un proceso por el cual una imagen descrita en un formato gráfico vectorial se convierte en un conjunto de píxeles o puntos para ser desplegados en un medio de salida digital, como una pantalla de computadora o una imagen de mapa de bits.

Término	Descripción
Reflexión especular	Es el resultado de la casi total reflexión de la luz incidente en una región concentrada alrededor de un ángulo de reflexión especular, y se caracteriza porque al observar una superficie brillante iluminada, como una manzana, se aprecia un toque de luz o una mancha del color de la fuente de iluminación y no del color de la manzana.
Renderizar	Proceso de síntesis de una imagen a partir de los datos de un escenario virtual y que culmina con la generación de la imagen que representa a dicho escenario desde un punto de vista elegido con anterioridad.
PSO	Algoritmo de optimización por cúmulo de partículas. Algoritmo de heurística bioinspirada basada en un algoritmo poblacional que toma patrones de comportamiento de grupos de individuos como aves o enjambres de abejas, en el cual todos los individuos se guían por los resultados de una búsqueda propia y una búsqueda global, con el fin de maximizar los resultados.
Traza de rayos	Método de síntesis de imágenes por computadora el cual consiste en simular la emisión de un rayo de luz, desde una posición de píxel a fin de localizar las intersecciones de la superficie para determinar las superficies visibles en una escena, una vez encontrada una superficie, el rayo es rebotado alrededor de la escena para recopilar las contribuciones de la intensidad de otros objetos. Es un método sencillo y poderoso usado para presentar efectos de reflexión global, transmisión, sombreado, fuentes múltiples de iluminación, movimiento, desenfoque de imagen. Por desgracia el uso de este método es computacionalmente muy costoso, ya que para generar imágenes fotorrealistas se necesita generar una gran cantidad de rayos.
Virtual	Algo que no es real, pero que sin embargo puede presentar algunas características de lo real. En el caso de los sistemas de cómputo se refiere a programas que simulan el comportamiento de equipo o proceso. Sin embargo en este caso particular se refiere a simulaciones generadas por computadora de objetos reales para la construcción de escenarios.