Paralelización de transformaciones y pruebas de intersección para Raytracing

Karen Jaqueline Reyes Flores
Lic.Tecnologías para la Información en Ciencias
UNAM, ENES CAMPUS MORELIA
karenreyes346@gmail.com

Alejandra Guadalupe Esquivel Guillén
Lic.Tecnologías para la Información en Ciencias
UNAM, ENES CAMPUS MORELIA
alejandraeg9899@gmail.com

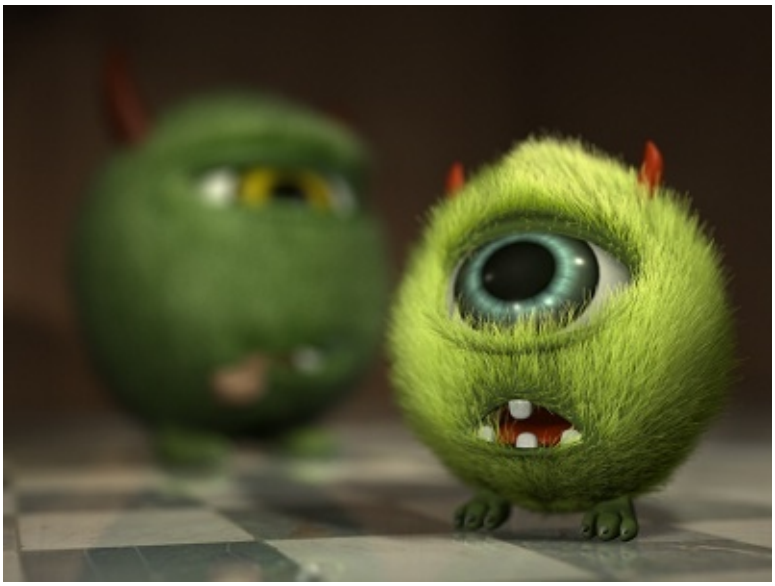

Figura 1: Representación de Raytracing

ABSTRACT

Para el presente proyecto se eligió de entre una gama de opciones, el objetivo era encontrar un código que ya hiciera la implementación de raytracing para posteriormente implementar la parte paralela utilizando el lenguaje de programación C o C++.El Raytracing utilizado fue el localizado en el siguiente repositorio:

- https://github.com/bnaveenkr/raytracer

Nuestro objetivo principal sobre la implementación se baso en las transformaciones contenidas en el mismo raytracer:

- Traslación
- Escalación
- Rotación

Anexando la paralelizacíón de las pruebas de intersección entre rayos y la geometría poligonal para la producción de una imagen de salida.

UNAM, CAMPUS MORELIA

DOI: 10.1145/nnnnnnn.nnnnnnn

KEYWORDS

Paralelización, Raytracing, OpenMP

ACM Reference format:
Karen Jaqueline Reyes Flores and Alejandra Guadalupe Esquivel Guillén. 2018. Paralelización de transformaciones y pruebas de intersección para Raytracing. In *Proceedings of UNAM, CAMPUS MORELIA, ENES*, 4 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1. INTRODUCCIÓN

- Raytracing

  Raytracing es una técnica para la generación de imágenes o animaciones 3D, que aplica técnicas de iluminación, reflejo, texturizado; utilizada en proyectos como:
  - Animación
  - Simulación

  El Raytracing modela el comportamiento de la interacción entre los objetos y la luz.Originalmente fue llamado algoritmo de Ray Casting y fue propuesto en 1980 por Turner Whitted.

  El algoritmo de Ray Casting determina las superficies que son visibles en la escena por medio del trazado de rayos desde el observador (cámara) hasta la escena a través del plano de la imagen, calculando así las intersecciones

más cercanas al observador determinando con ello el objeto visible.

Una de las principales ventajas de esta técnica es que realmente te enfocas en los rayos de importancia ya que no exploras todos los rayos que se generarían; sino en aquellos que son de relevancia para el observador.

- OpenMP

  OpenMP es un framework para programación que gestiona automáticamente los recursos para el flujo de los multiproceso de memoria compartida.
- Trabajo a realizar

  La intención de este trabajo es llevar a cabo una paralelización, esto con la intención de que el código sea más eficiente. Para ello se va a utilizar OpenMP.

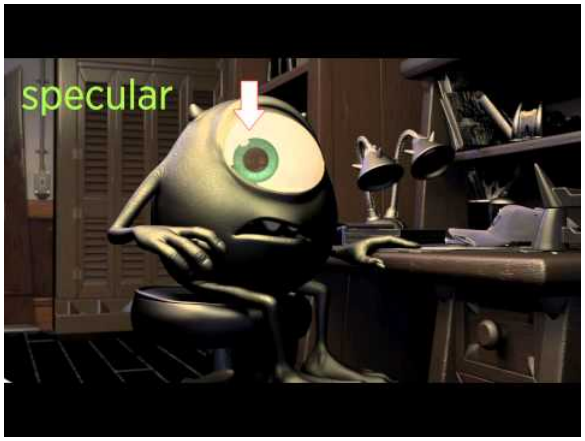

Figura 2: Render de la película Monsters University

2. TRABAJOS RELACIONADOS

La primera implementación de un Raytracer .en tiempo real"fue acreditada en la conferencia de gráficos por computadora SIGGRAPH 2005 como las herramientas *REMRT/RT* desarrolladas en 1986 por Mike Muuss para un sistema de modelado de sólidos *BRL − CAD*. Inicialmente publicado en 1987 en USENIX, el Raytracer *BRL − CAD* es la primera implementación conocida de un sistema de Raytracer distribuido en paralelo que logró varios cuadros por segundo en rendimiento de renderizado. Este rendimiento se logró mediante el motor de rastreo de rayos *LIBRT* altamente optimizado pero independiente de la plataforma en *BRL − CAD* y mediante el uso de una sólida geometría CSG implícita en varias máquinas paralelas de memoria compartida en una red de productos. El Raytracer de *BRL−CAD*, incluidas las herramientas *MAAKA/RT*, sigue estando disponible y desarrollado hoy como software de código abierto.

Desde entonces, se han realizado considerables esfuerzos e investigaciones para implementar el Raytracer en velocidades de tiempo real para una variedad de propósitos en configuraciones de escritorio independientes. Estos propósitos incluyen aplicaciones interactivas de gráficos en 3D, como producciones de demoscene, computadora y videojuegos, y representación de imágenes. Algunos motores 3D de software en tiempo real basados en el Raytracer han sido desarrollados por programadores de demostraciones de aficionados desde finales de la década de 1990.

El proyecto *OpenRT* incluye un núcleo de software altamente optimizado para el Raytracer junto con una API tipo *OpenGL* para ofrecer una alternativa al enfoque actual basado en rastreo para gráficos 3D interactivos. El hardware de Raytracer, como la unidad de procesamiento de rayo experimental desarrollada en la Universidad de Saarland, se ha diseñado para acelerar algunas de las operaciones intensivas en computación del trazado de rayos. El 16 de marzo de 2007, la Universidad de Saarland reveló la implementación de un motor de rastreo de rayos de alto rendimiento que permitía renderizar los juegos de computadora mediante el Raytracer sin un uso intensivo de recursos.

El 12 de junio de 2008 Intel demostró una versión especial de Enemy Territory: Quake Wars, titulada $QuakeWars : RayTraced$, utilizando el Raytracer para renderizar, que se ejecuta en resolución HD básica (720p). *ETQW* operaba a 14-29 fotogramas por segundo. La demostración se desarrolló en un sistema Xeon Tigerton de 16 núcleos (4 zócalos, 4 núcleos) que funcionaba a 2,93 GHz.

En SIGGRAPH 2009, Nvidia anunció *OptiX*, una API gratuita para el seguimiento de rayos en tiempo real en las GPU de Nvidia. La API expone siete puntos de entrada programables dentro del pipe de Raytracer, lo que permite cámaras personalizadas, intersecciones primitivas de rayos, sombreadores, sombras, etc. Esta flexibilidad permite el trazado bidireccional, el transporte ligero de Metropolis y muchos otros algoritmos de renderización que no se pueden implementar con recursividad de cola. Nvidia ha enviado más de 350,000,000 GPU compatibles con *OptiX* a partir de abril de 2013. Los procesadores basados en *OptiX* se utilizan en Adobe AfterEffects, Bunkspeed Shot, Autodesk Maya, 3ds max y muchos otros renderizadores.

Figura 3: Ejemplo Raytracer con *OptriX*

3. DESCRIPCIÓN DE LA PARALELIZACIÓN

La parte de paralelización se realizo solamente la paralelización de un *FOR* que se encontraba en el método $transformObject(Object* object, MatrixM)$ lo que recibía era:

- **Object *object**: que era el objeto a transformar, creado con anterioridad.
- Matrix M: que son las transformaciones (rotación, traslación y escalación).

Esto mediante la directiva *pragma omp parallel for* que lo que hace es paralelizar (dividir las tareas entre el número de hilos) de las instrucciones involucradas dentro del *For*.

```
#pragma omp parallel for
  for(i = 0; i < (*object).ntris; i++)
  {
      //printf(" Threads: %d\n", i);
      temp.v0 = matVecMult(M, ((*object).tri[i].v0));
      temp.v1 = matVecMult(M, ((*object).tri[i].v1));
      temp.v2 = matVecMult(M, ((*object).tri[i].v2));
      (*object).tri[i].v0 = temp.v0;
      (*object).tri[i].v1 = temp.v1;
      (*object).tri[i].v2 = temp.v2;
  }
```

Figura 4: Paralelización de las transformaciones de translación, escalación y rotación.

Para la paralelización de las pruebas de intersección entre rayos y la geometría poligonal para la producción de una imagen de salida se implementó una región paralela utilizando:

- **shared(X)**: las variables se declaran como compartidas y existe una única copia a la que los threads pueden acceder y modificar la variable (Para este caso fueron scene, light y recursión).
- **private(Y)**: las variables se declaran como privadas en cada thread creando P copias por cada thread (Para este ejercicio fueron outcolor, ray, i, j, index).
- **firstprivate(Y)**: se queda con la primera variable privada que se genera (Para este caso fueron width, height, image).
- directiva **parallel** for: generando P threads.
- **collapse(n)**: indicándole al compilador que un ciclo será tratado como uno solo, indicando el número de ciclos a colapsar.

Ello con la finalidad de generar las imágenes de forma paralela.

```
#pragma omp parallel for collapse(2) private(outcolor, ray, i, j, index) firstprivate(width, height, image) shared(scene, light, recur)
  for(j = 0; j < height; j++){
      for(i = 0; i < width; i++)
      {
          //printf(" ray id: %d , %d\n",i,j);
          ray = generateRay(i, j, cam);
          outcolor = vectorZcolor(trace(ray, scene, light, recur));
          #pragma omp critical
            setPixel(&image, i, j, outcolor, index, width, height);
            //printf("Finish  critical section: %d \n", omp_get_thread_num());
      }
  }
```

Figura 5: Paralelización sobre la generación de la imagen.

4. RESULTADOS

Se genera una imagen de forma paralela sin aplicar aún las transformaciones y nos da como resultado:

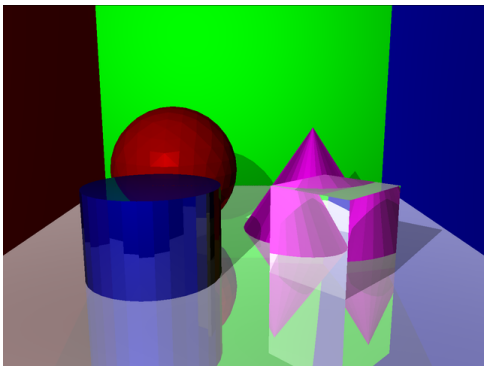

Figura 6: Raytracing generado de forma paralela.

Se genera una imagen de forma paralela aplicando la transformaciones de traslación:

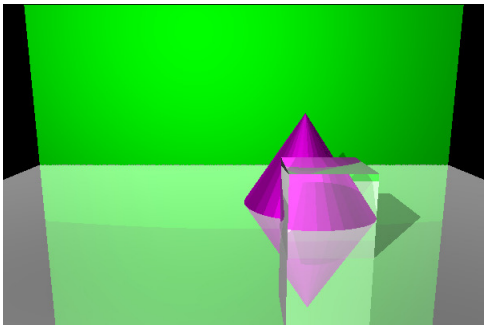

Figura 7: Raytracing aplicando traslación.

Se genera una imagen de forma paralela aplicando las transformaciones a solamente 2 objetos (cubo y cono) la traslación y rotación:

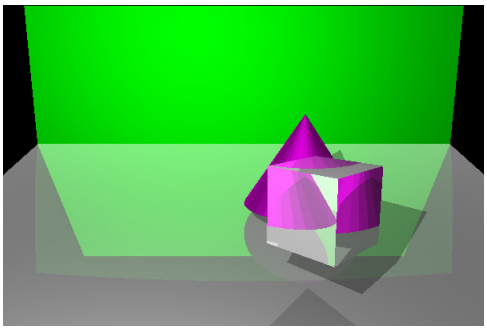

Figura 8: Raytracing aplicando traslación y rotación.

Se genera una imagen de forma paralela aplicando las transformaciones a solamente 2 objetos la traslación, rotación y escalación:

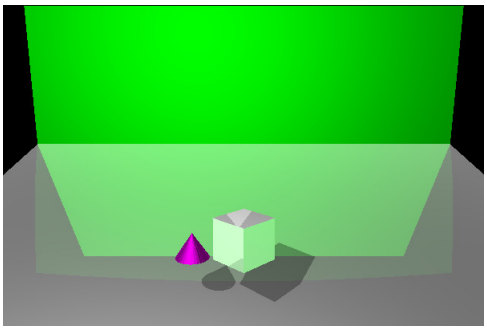

Figura 9: Raytracing aplicando traslación y rotación y escalación.

5. CONCLUSIONES

Durante la ejecución se pudo ver como si se utilizan todos los cores para la distribución de las tareas: La librería de **Openmp** fue

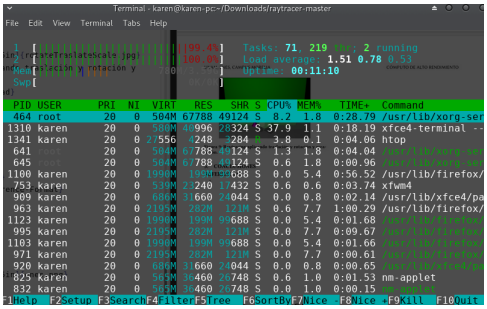

Figura 10: 4 cores utilizados al 100 **porciento.**

de gran ayuda para comprender como se distribuyen los procesos y tareas en los núcleos e hilos.

Al renderizar imágenes grandes y al generar varios objetos, la creación de triángulos por imagen y de rayos que relejan en la misma se convierten en una cantidad muy grande que para mejorar la calidad de la imagen se tiene que hacer uso de las herramientas de paralelización.

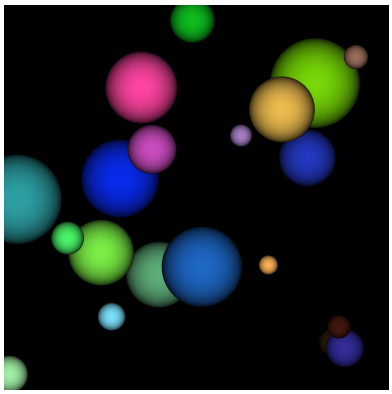

Figura 11: Ejemplo de Raytracing.

REFERENCIAS

Khan Academy. (2018). Introducción al renderizado. May 22, 2018, de Khan Academy Sitio web: https://es.khanacademy.org/partner-content/pixar/rendering/rendering1/v/rendering-1

Dr. Ulises Olivares Pinto. (2018). Introducción a OpenMP. May, 2018, de Dr. Ulises Olivares Pinto
Sitio web: https://app.schoology.com/course/1474429279/materials/gp/1580749513