Computación Gráfica - Ray & Path Tracing

Gonzalo Castiglione (49138) - Braulio Sepede (51074)

Grupo 2

Contents

1 Definiciones

A fin de evitar ambiguedades en las explicaciones realizadas en el informe, se listan a continuación todos los términos que van a ser usados:

Transform Representa una rotación, translación y escala. Implementado utilizando una matriz de $4x4$ doubles.

Bounding Volume Volumen en coordenada de mundo utilizado para encerrar formas complejas.

Tipos: AABB o Sphere

Spatial Objeto que posee un Transform y Bounding Volume.

Light Objeto con un color specular y difuse [1].

Material Representa las cualidades "físicas" de un objeto[2].

Mesh Representa la forma de un objeto.

Tipos: Triangle, Box, Sphere, Disc, Plane, FinitePlane

Geometry Spatial con un Material y un Mesh.

Scene Conjunto de Lights y Spatials a ser tenidos en cuenta para el cálculo de un frame.

2 Ray Tracer

2.1 Luces

Se implementaron los siguientes tipos de luces:

- Direccional
- Ambiental
- Puntual
- Spotlight
  - Este tipo de luz es identica a una luz puntual, con la diferencia que se multiplica el resultado por un decay cuando el ángulo entre la dirección del spotlight y el rayo incidente es mayor al especificado. Este decay es calculado utilizando la siguiente formula:

$$decay = max(ray.dir \bullet light.dir, 0)^{spot\_exp}$$

    - En donde $spot\_exp$ es una constante que define cuan rápudo va a set el deacimiento de la luz a partir que se pasa de la "pantalla". Cuanto maror es el valor para $spot\_exp$, mas brusco va a ser el cambio de intensidad de la luz una vez superado el ángulo máximo.

[1] Especificaciones en: 2.1
[2] Especificaciones en: 2.2

2.2 Materiales

2.2.1 Shader

Los métodos de shading implementados son: Lambert y Phong.

2.2.2 Tipos

En la tabla 1 se presentan todos los tipos de materiales disponibles que pueden ser aplicados a cada Geometry en la actual implementación:

	Reflejos	Refraccion	Texturizable	Shader (default)
Matte	No	No	Si	Lambert
Glass	Si	Si	Si	N/A
Metal (shiny)	Si	No	Si	Phong
Mirror	Si	No	Si	N/A

Table 1: Tipos de materiales

Para el caso del material glass, se implementó Fresnell.

2.2.3 Mix

Es posible crear un nuevo material mezcla de dos materiales existentes. Para la construccion del material mezcla, se realiza un *lerp* [3] entre cada una de las constantes de ambos materiales entregados y se asigna el valor al material mix.

La mezcla de materiales tiene como restricción que ambas texturas deben tener las mismas dimensiones. De lo contrario no es posible interpolarlas.

Debido a que no es posible la aplicacion de *lerp* sobre los campos "shader" y "tipo" de un material, simplemente se dejan los valores especificados en el primer material.

Ejemplo:

- Material 1: {*shader*: Lambert, *texture*: t1, *type*: Glass, *ka* : ka1, ...}
- Material 2: {*shader*: Phong, *texture*: t2, *type* : Metal, *ka* : ka2, ...}
- $\Rightarrow$ Material 3: {*shader* : **Lambert**, *texture* : lerp(t1, t2), *type* : **Glass**, *ka* : lerp(ka1, ka2), ...}

2.3 Antialiasing

Existen implementados dos métodos para anti-aliasing: Jittered y Uniform. Cada uno de estos parametrizable en el tamaño de la ventana a usar.

2.4 Problemas encontrados

- Dificultad de debugging

  Siempre que el resultado de un ray trace da valores anormales, puede resultar sumamente dificil encontrar la razón si se tiene en cuenta la cantidad de operaciones aritmeticas que se realizan hasta resolver el color.

[3] http://en.wikipedia.org/wiki/Linear_interpolation

La forma mas eficaz para solucionar estos casos por lo general consiste poner un único Geometry en escena y crear una imagen de un solo pixel (el rayo sale siempre con dirección $(0, 0, -1)$) facilitando mucho las cuentas.

  - El caso que mas remarcamos para esta categoría (costo mucho diagnosticar) sucedía al crear un Octree de triángulos. Lo que sucedía era que no se tenía bien implementado la verificación para detectar la intersección triángulo-AABB (OctreeNode). Produciendo imagenes como las siguientes:

- Representacion de los objetos (Eficiente vs mantenible)

  Durante toda la etapa de desarollo surgió la decisión entre realizar implementaciones que sean ineficientes pero muy fáciles de entender / mantener o bien eficientes pero complicadas de entender (Por ejemplo: Repetir codigo para realizar una operación compleja o evitar crear demasiados objetos temporales para cuentas sencillas).

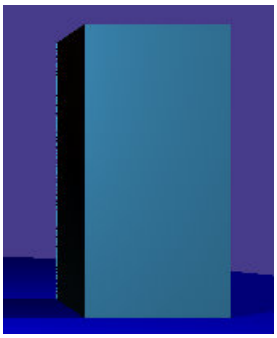
  Muchas veces un código cortito puede parecer muy inocente, pero al ejecutarse tantas miles de veces, pequeños cambios hacen la diferencia.

- Opraciones con colores (sumas vs multiplicacion)

  Para el caso de los reflejos y refracciones, se tuvieron muchas dudas sobre si el color resultante de un reflejo o refraccion se debe sumar o multiplicar al color del material. Cada uno de los casos daba mejores resultados dependiendo de los materiales y cantidad de objetos.

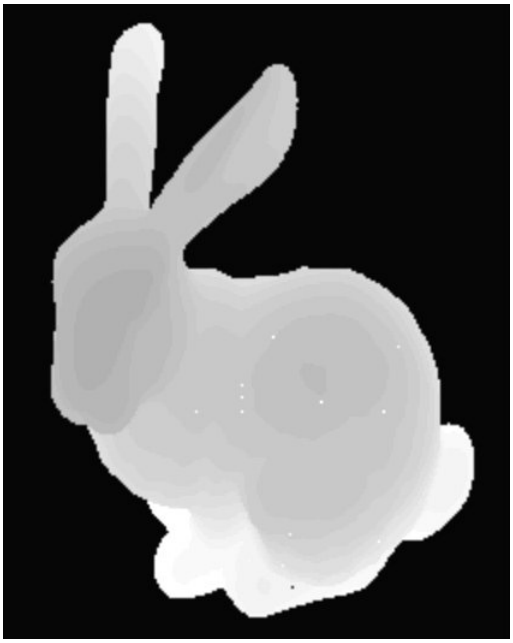
- Condiciones de bordes / limites

  - Para el caso del Box, hubieron problemas con los bordes de cada cara. El resultado de la colision pegaba muchas veces en la que se esperaba, y otras en la del otro lado.



– En el caso del octree, sucedía que cuando el mesh era muy chico (pasaba para el caso de bunny.lux), el tamaño de cada nodo se volvía demasiado pequeño y aprecían algunos pequeños puntitos sobe el mesh debido a que el rayo traspasaba al nodo de octree y pegaba contra el de atrás directamente. El problema se solucionó poniendo una cota mínima para el tamaño de cada nodo de octree.

En la imagen a continuación se muestra un mapa de profundidad de las colisiones removiendo la cota inferor del octree con un mesh de $75K$ polígonos y un ancho aproximado de 0.03 unidades para el mesh entero.



2.5 Tiempos

En la tabla a continuación se muestran los resultados obtenidos con la aplicación para el cálculo de una imagen en con un procesador $i5$ y 4 Gb de RAM.

- Escenas:

  Escena 1

  – Stanford Bunny. Matte. ($69k$ polígonos)

- 2 Luces puntuales

Escena 2

- Dos espejos enfrentados
- 3 Esferas: Metal, Glass y Matte entre los espejos.
- 1 disco matte (piso)
- 1 luz direccional
- 1 luz puntual

Escena 3

- Stanford Bunny. Matte. ($69k$ polígonos)
- 1 Spot light
- 1 Disco Vidrio

Escena 4

- Escena complea con$0.5M$ de triángulos
- 1 Spot light
- Tamaño de archivo fisico: 37 $[Mb]$ (sin considerar texturas).
- Vasos de vidrio, cubiertos metalicos, platos y nombre del grupo (meshes).

- Configuraciones:

0 Multi Thread. Cant reflejos: 0. Cant. de Refracciones: 0. AA: Uniform,1. Resolución: $640x480\,[px]$

1 Single Thread. Cant reflejos: 3. Cant. de Refracciones: 5. AA: Uniform: 1. Resolución: Full HD [4]

2 Multi Thread. Cant reflejos: 3. Cant. de Refracciones: 5. AA: Uniform: 1. Resolución: Full HD

3 Multi Thread. Cant reflejos: 3. Cant. de Refracciones: 5. AA: Jittered: 4. Resolución: Full HD

	Conf. 0 [s]	Conf. 1[s]	Conf. 2 [s]	Conf. 3 [s]
Escena 1	9.4	142.7	90.0	323.6
Escena 2	1.3	9.9	6.4	15.19
Escena 3	3.2	37.5	16.1	69.81
Escena 4	2.1	39.4	26.5	102.3

3 Path tracer

3.1 Luces

Las luces implementadas incluyen todas las ya mencionadas en el ray tracer2.1. Además, se agregaron area lights.

Un area light esta definido por un material que tiene asociado una luz. De esta forma, cualquier geometry de la escena podría ser una luz de cualquier tipo, en cualquier momento. Lo único que se requere ahora, es tener una forma para poder samplar el mesh asociado a este geometry y de esta forma poder calcular la iluminación directa producto de esta luz. Para esto, se asocia a interfaz Mesh un método sampler que se usa para crear samplings aleatorios sobre el mesh.

Las area lights soportadas por la aplicación son las siguientes:

[4] http://es.wikipedia.org/wiki/1080p

- Mesh light
- Rectange light
- Sphere light

Queda pendiente la implementación del cálculo de pérdida de energía de la luz debido a la distancia recorrida. No fue implementado por falta de tiempo en plazos de entrega.

3.2 Materiales

3.3 Tipos

Los materiales disponibles son los mismos que los presentados en la sección2.2 con la excepción de Phong. El uso de Phong en los metales fue reemplazado por Cook Torrance.

3.4 Sampling

Cada un vez que un rayo que salió de la camara choca contra un mesh, se obtiene su material. Según el tipo de material, la forma con la que el rayo vuelve a salir va a depender del tipo de sampler asociado.

3.4.1 Matte

En este caso, se usa un medio hemisferio cuyo centro es la normal al punto de colisión. En la siguiente imagen se muestra un sampling obtenido sobre un material matte.

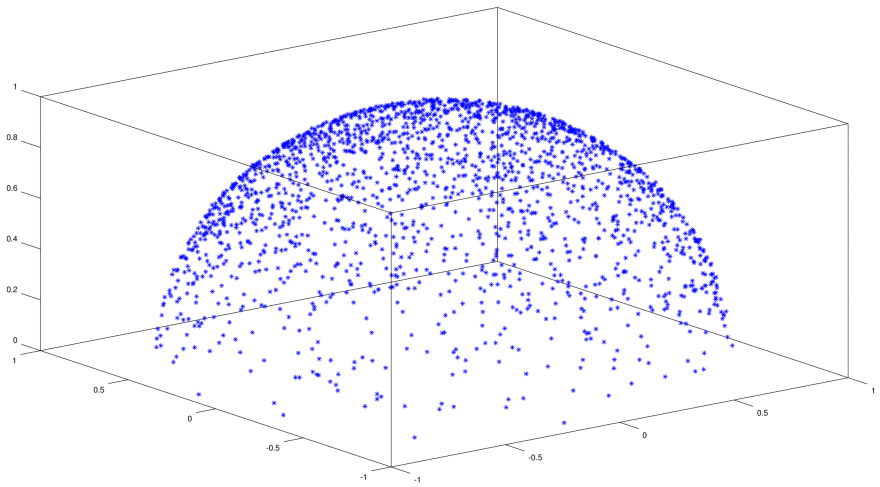


Figure 1: Sampling sobre un material tipo matte

Los puntos fueron obtenidos guardando la dirección de salida sobre un material matte en la aplicación y luego renderizandolos con octave.

3.4.2 Metal

Para el caso del metal. El sampling dependera de las caracteristicas del metal. Esta propiedad esta definida por el valor de $1/roughness$. A mayor exponente, mas concentrados van a ser los rayo reflejados.

En la imagen a continuación se muestran dos ejemplos de sampleos sobre un metal con una constante de $roughness = 0.1$ (rojo) y otro con $roughness = 0.04$ (azul). Se puede ver que a medida que se decrementa el roughness. Mas parecido al reflejo de un espejo va a parecer.

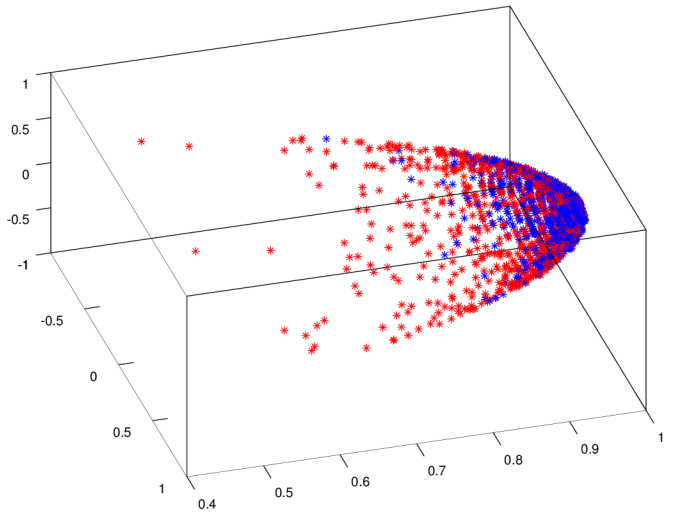


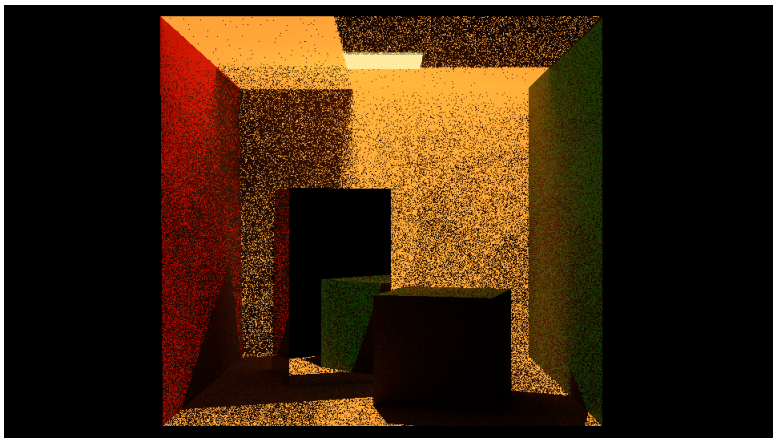
Figure 2: Sampling sobre dos materiales tipo metal

Los puntos fueron obtenidos guardando la dirección de salida sobre un material metal en la aplicación y luego renderizandolos con octave.

3.5 Problemas encontrados

Entre los problemas encontrados, se encuentran todos los mencionados en la seccion de ray tracer y algunos nuevos:

- Resulta de interés el tiempo de debugging adicional requerido. Ya que cada escena necesita de varias decenas de samplings por pixel para que se vea interesante, el tiempo de construcción de cada frame requiere de varias veces el tiempo que un ray tracer emplearía.
- Otro detalle (super) importante es el del sampleo correcto del hemisferio. El detectar estos errores resulta MUY dificil debido a la cantidad de muestras que se requeire para poder "visualizar" el error. A cantinuación se muestran algunas imagenes obtenidas con un sampling incorrecto:



- Otro detalle a agregar, es la dificulad que este algoritmo tiene para el cálculo de promedio de los n caminos por cada pixel y a su vez de los m samples de antialiasing para cada pixel ($m * n$ caminos en total) para el cálculo de un solo pixel de la imagen final. En la imagen a continuación puede verse un caso donde el promedio es incorrecto:

- Un último problema para la lista, tiene que ver con la elección de Octree como método de subdivisión espacial. La escena Sponza, tiene triángulos que están por afuera del rango máximo

que se configuró incialmente para el rango del octree[5]. Queda pendiente un cambio a una estructura que pueda sportar estos casos. No fue implementado por falta de tiempo.

3.6 Tiempos

En la tabla a continuación se muestran los resultados obtenidos con la aplicación para el cálculo de una imagen en con un procesador $i5$ y 4 Gb de RAM.

- Escenas
  - Cornell Box
  - Metal 2
  - Glass Bunny
- Configuraciones:

  0 Multi Thread. Samples: 10. Trace depth: 5. AA: Uniform,1.

  1 Multi Thread. Samples: 100. Trace depth: 5. AA: Uniform,1.

  2 Multi Thread. Samples: 1000. Trace depth: 5. AA: Uniform,1.

  4 Single Thread. Samples: 10. Trace depth: 5. AA: Uniform,1.

	Configuracion 1 $[s]$	Configuracion 2 $[s]$	Configuracion 3 $[s]$	Configuracion4 $[s]$
Cornell Box	18.8	130.5	1315.3	78.3
Metal 2	8.4	70.6	690.3	32.4
Glass bunny	132.2	1236.1	$10\,min+$	590.1

4 División espacial

El método de division espacial utilizado en la implementación actual hace uso de octrees. Existen dos usos para este tipo de árbol en la aplicación:

Scene Ordena los spatials que se encuentran en la escena principal. Cada Spatial se envuelve con un BoundingVolume y se inserta en el octree.

Triangles Para la resolución de colisiones contra triangle meshes, es necesario mantener una división espacial de los triángulos que lo conforman para una búsqueda eficiente. Por lo general, los meshes suelen tener una distribución bastante homogenea de los triángulos que lo en cuanto a distribución espacial, haciendo muy conveniente el uso de octrees en estos casos. Es por esto que se consideró a este como el mejor candidato a implementar en la apliacción.

Para la resolución de objetos que caen entre dos nodos del octree, se resolvió que se objeto se guarda repetido (una vez en cada nodo).

[5] Resulta muy dificil el calculo dinamico de rango para un mesh debido a posibilidad de transformaciones y anidaciones de los mismos

5 Rendering

Se implementaron tres métodos de rendering diferentes. Estos son los siguientes:

Normal Se dibuja un mapa de normales de cada colisión. No se calculan luces, refracciones ni reflejos.

Profundidad Se dibuja un mapa de prfundidad de cada colisión. No se calculan luces, refracciones ni reflejos.

Luces Se dibuja el color la geometría que produjo la colisión. Tiene en cuenta las luces, los reflejos y las refracciones.

6 Corrección de color

Dependiendo de la situación, puede darse que los valores de los colores resultantes tenga valores fuera de rango. Para poder representar estos valores en una imagen, es necesario aplicar algun método de corrección de color. Se tienen implementados dos métodos para realizar el ajuste de rango:

Corte Cuando un valor se encuentra por encima de 1. Se lo reemplaza por 1. De lo contrario, queda sin cambios.

Dynamic Range Compression El método de corte produce que cualquier valor de color que sea superior a 1 siempre se va a ver igual a cualquier otro valor superior a 1 (no importa de cuanto sea la diferecia). Esto hace parecer que dos objetos se vean parecidos (o iguales) mas alla de que en realidad no lo sean y sucede debdo a que se pierde la información real del color al pisar su valor. Este método se encarga de normalizar todos los valores según el máximo que exista en la imagen (en ese canal). Puede encontrarse una explicación mas detallada en http://en.wikipedia.org/wiki/Dynamic_range

7 Multi Threading

La división de tareas dentro de la aplicación se logra dividiendo la imagen en conjuntos de filas de tamaño fijo (50 por default). La cantidad de threads que van a crearse va a ser igual a la cantidad de cores disponibles [6] que tenga la *CPU* que se encuentre corriendo la aplicación.

Cada uno de los threads va a a ir tomando de a un conjunto de filas y creando el pedacito de imagen correspondiente. Cada vez que un thread termina con las filas que se le fueron asignadas, simplemente toma el próximo disponible y repite hasta que no queden más.

El frame queda construido completamente cuando todos los threads terminan su ejecución.

8 Configuración y ejecución

Parametros aceptados por linea de comandos:

help Opcional. Flag. Imprime la lista de parametros y lo que cada uno significa.

o Opcional. Nombre del archivo de salida.

i Nombre del archivo lux de entrada.

time Opcional. Flag. Mediciones de los tiempos empleados en el render.

[6] ver: https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html#availableProcessors()

aa Numerico. Cantidad de muestras de antialiasing.

benchmark Opcional. Numérico. Cantidad de frames consecutivos a crear.

La creacion del archivo fisico (PNG) se realiza únicamente para el primer frame.

d Opcional. Numérico. Define el ray depth de reflejos y refracciones.

La profundidad para las refracciones va a ser el valor de (d + 2)

r Opcional. Define el tipo de render a usar.

Valores: normal, distance, light.

correction Opcional. Tipo de correción de la imagen.

d Método de compresión de rango dinámico.

c Método de corte.

gui Opcional. Flag. Abrir interfaz gráfica. Consideraciones: Tiene como única intención el debugging de una escena. Esta aún en etapa alpha.

pathtracer Opcional. Utilizar metodo de path tracing

tr Define el la cantidad de hops que un camino puede tener (solo pathtracer)

s Define la cantidad de samples por pixel (solo pathtracer)

9 Ejemplos

Todas las imagenes de prueba (y videos) se encuentran en la carpeta *samples/pictures* y *samples/videos* del repositorio.

10 Fuentes consultadas

- Fundamentals of Computer Graphics. Third edition. Peter Shirley
- Ray Tracing From The Ground Up. Kevin Suffern.
- http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006–degreve–reflection_refraction.pdf
- http://www.cs.rpi.edu/∼cutler/classes/advancedgraphics/F05/lectures/13_ray_tracing.pdf
- http://www.cs.utah.edu/∼shirley/books/fcg2/rt.pdf
- http://courses.cs.washington.edu/courses/cse457/10au/lectures/markup/ray-tracing-markup.pdf
- https://en.wikipedia.org/wiki/Schlick's_approximation
- http://ruh.li/GraphicsCookTorrance.html
- Computación Gráfica - Clase 7, 8, 9 y 10