

# Computación Gráfica

*Trabajo Práctico 2 - Path Tracing*

## Yet Another Tracer

Because there are not too many



## Grupo 3

Del Giudice Gustavo

Menzella Facundo

Noriega Jose

# Índice

1. Estructuras de aceleración
2. Area Lights
3. Materiales implementados
4. Sampling
5. Iluminación indirecta vs Iluminación directa
6. Parámetros de consola
7. Parseo de escenas
8. Funcionalidades adicionales
9. Escenas de referencia
10. Bibliografía

# 1. Estructuras de aceleración

En cuanto a estructuras de aceleración, se perfeccionó la implementación del *SAH-KDTree*. La implementación del trabajo anterior, presentaba falencias a la hora de posicionar los planos de corte debido a:

- Cálculos fallidos de surface area.
- Cálculos fallidos a la hora de determinar el costo del plano de corte. Se le daba mucha prioridad a planos que podaban una rama del árbol (dejaban un lado del nodo vacío). Esto nos incrementaba considerablemente el tiempo para el *Ray Tracer* en escenas donde la densidad de triángulos estaba concentrada.

A continuación se encuentra la comparación entre los tiempos anteriores y actuales. Los renders se generaron bajo los mismos parámetros y con la misma PC.

| Escena | Tiempo promedio (Antes) | Tiempo promedio (Ahora) |
|--------|-------------------------|-------------------------|
| 1      | 1m 58s                  | 4508ms                  |
| 2      | 1m 59s                  | 4373ms                  |
| 3      | 2m 1s                   | 4487ms                  |
| 4      | 2m 24s                  | 5.69s                   |
| 5      | 2m 29s                  | 5.577s                  |
| 6      | 2m 26s                  | 5.507s                  |
| 7      | 2m 15s                  | 6.143s                  |
| 8      | 2m 12s                  | 6.243s                  |
| 9      | 2m 15s                  | 6.124s                  |
| 10     | 2m 25s                  | 5.368s                  |
| 11     | 2m 25s                  | 5.43s                   |
| 12     | 2m 25s                  | 5.693s                  |
| 13     | 852ms                   | 903ms                   |
| 14     | 2202ms                  | 2362ms                  |
| 15     | 8s                      | 7.654s                  |
| 16     | 12s                     | 11.835s                 |
| 18     | 3m 40s                  | 1316ms                  |
| 19     | 6m 52s                  | 2735ms                  |
| 20     | 17m 8s                  | 5.657s                  |

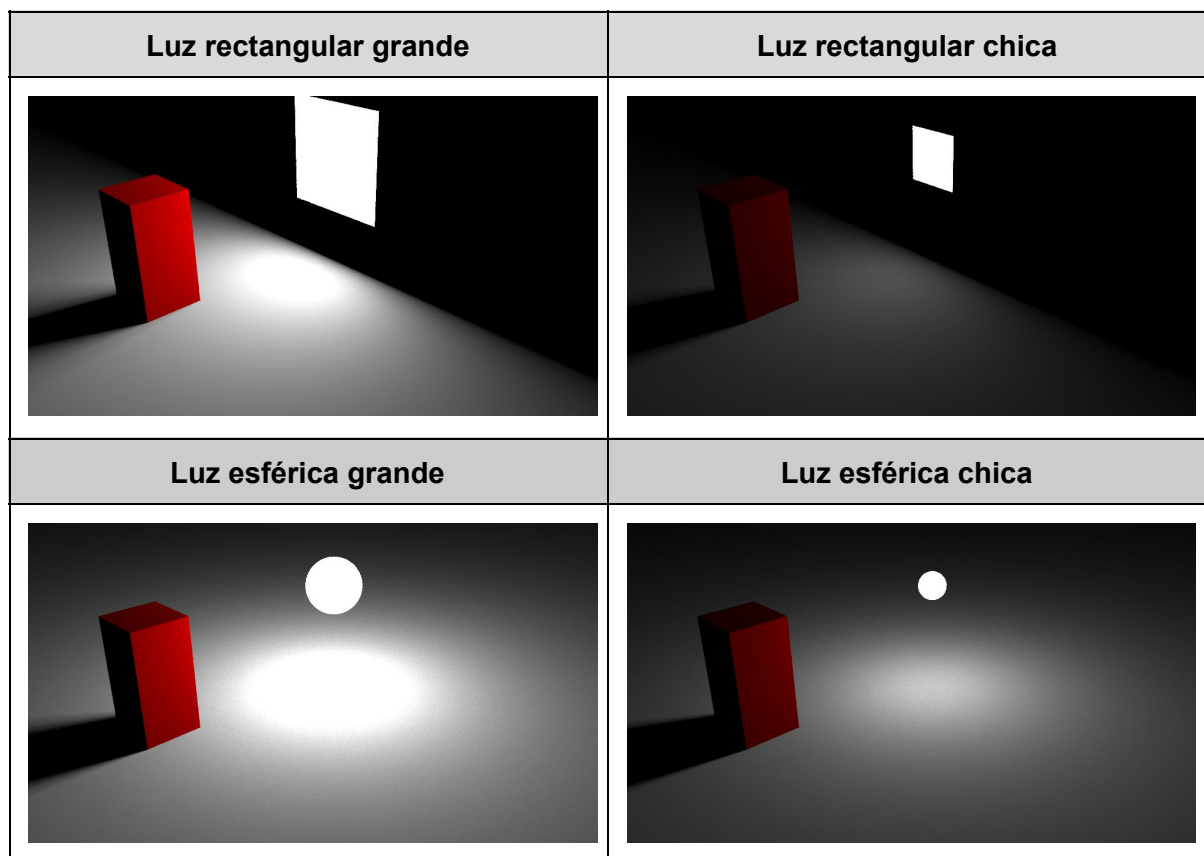
## 2. Area Lights

En esta implementación, las Area Lights están compuestas por tres partes:

1. Una figura, usada para representar al Area Light
2. Un material emisor, usado para los rayos primarios que impacten sobre la figura
3. Una luz, usada para computar la iluminación directa sobre los otros objetos en la escena

Estos componentes extienden la lógica implementada en el Trabajo Práctico anterior, por lo que su integración resultó sencilla.

Al impactar un rayo contra una superficie, se comienza a computar la iluminación directa producida por todas las Area Lights de la escena. Los cálculos son similares a las luces puntuales y direccionales, con la diferencia de que la función de visibilidad no se aplica una vez por luz, sino que se toman muestras de cada Area Light, y se aplica a cada muestra. Además de considerar la luminosidad de la luz, se tiene en cuenta el vector normal de la muestra tomada, y el tamaño de la figura usada por el Area Light. Esto permite que luces de mayor tamaño produzcan mayor iluminación.



Para optimizar el muestreo de las luces, se genera una gran cantidad de muestras al cargar la escena, incluyendo el cómputo de las normales y la transformación de los puntos. A la hora de calcular la iluminación directa, se toma aleatoriamente una de estas muestras.

Para simplificar el muestreo de Area Lights esféricas, se muestrea sobre toda la esfera, ya que muestrear el hemisferio visible desde el punto de impacto requiere recalculas las muestras.

Debido a que el archivo de escena de LuxRender define a las Area Lights rectangulares como un Mesh, (a pesar de no ser parte del enunciado) se decidió dar soporte primitivo para Mesh Lights, lo cual incluye soporte para Area Lights rectangulares. Al muestrear un Mesh Light, se elige aleatoriamente un triángulo, y a su vez una muestra sobre dicho triángulo.

### 3. Materiales

#### Metal2

En la entrega anterior, el material Metal2 se implementó usando la clase Reflective que a su vez extiende de Phong. En esta nueva iteración, se implementó la clase Metal2, que extiende de MaterialAbstract. El objetivo es aproximar metales con “microfacetas”, es decir con imperfecciones en la superficie y para esto se utilizó el modelo de Cook-Torrance, implementado en la clase del mismo nombre, que extiende de BRDF. Metal2 toma como parámetros un número “roughness” y un color de fresnel (“float roughness” y “color fresnel” en Lux)

El modelo de Cook-Torrance contiene una componente de Fresnel (F), una atenuación geométrica (G) y la distribución de microfacetas (D).

$$r_s = \frac{F \times D \times G}{\pi (\mathbf{n} \cdot \mathbf{l})(\mathbf{n} \cdot \mathbf{v})}$$

Para D, se utilizó la función de distribución de Beckmans, donde el valor de  $m$  representa la rugosidad (roughness) de la superficie.

$$D = \frac{1}{\pi m^2 \cos^4 \alpha} e^{-\left(\frac{\tan \alpha}{m}\right)^2} = \frac{1}{\pi m^2 (\mathbf{n} \cdot \mathbf{h})^4} e^{\left(\frac{(\mathbf{n} \cdot \mathbf{h})^2 - 1}{m^2 (\mathbf{n} \cdot \mathbf{h})^2}\right)}$$

G se implementó usando el modelo de Blinn.

$$G = \min\left(1, \frac{2(\mathbf{n} \cdot \mathbf{h})(\mathbf{n} \cdot \mathbf{v})}{\mathbf{v} \cdot \mathbf{h}}, \frac{2(\mathbf{n} \cdot \mathbf{h})(\mathbf{n} \cdot \mathbf{l})}{\mathbf{l} \cdot \mathbf{h}}\right)$$

Y por último, como las ecuaciones de Fresnel pueden ser muy complejas, se utilizó la aproximación de Schlick, donde  $f$  es el color de Fresnel mencionado anteriormente.

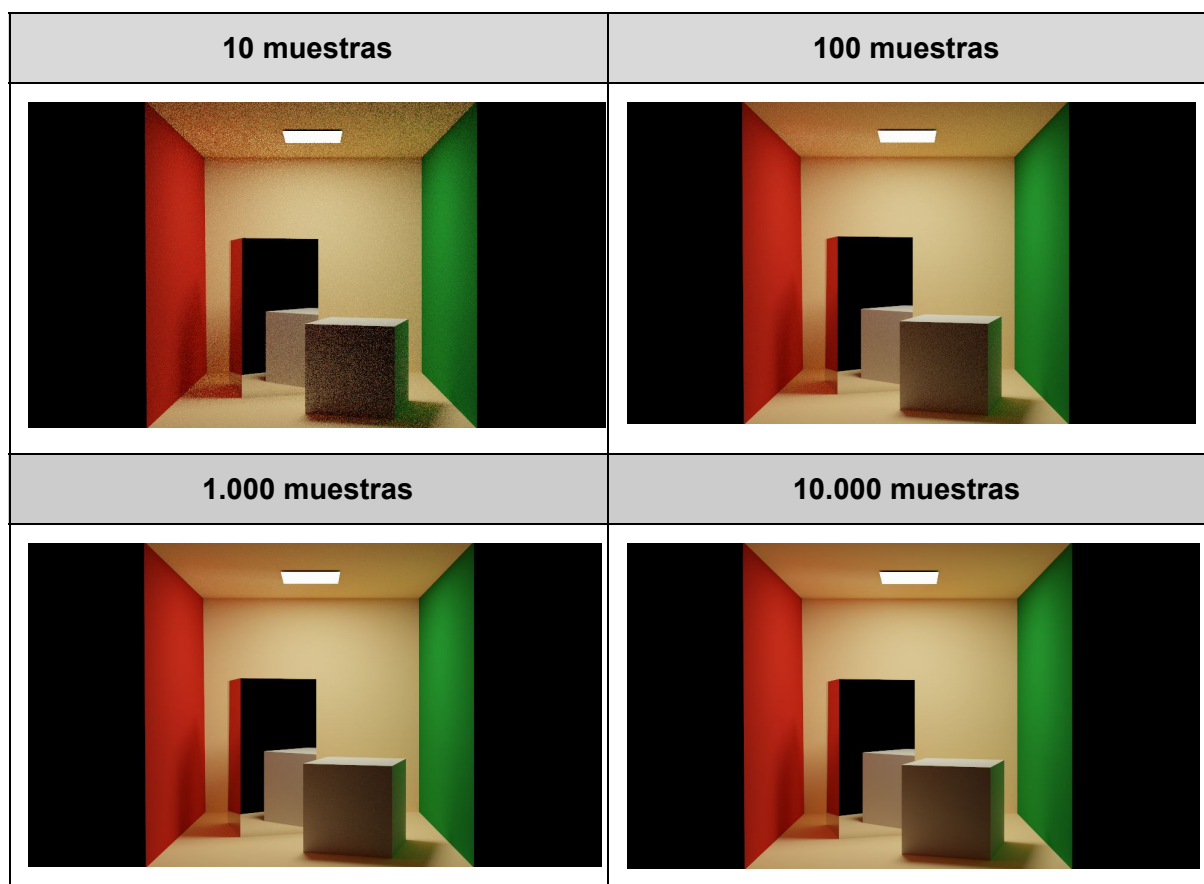
$$F_\lambda(u) = f_\lambda + (1 - f_\lambda)(1 - (\mathbf{h} \cdot \mathbf{v}))^5$$

El principal problema que se tuvo fue establecer una relación entre el reflejo y la rugosidad del material. El libro “Ray Tracing from ground up”, utiliza una serie de samplers que determinan para donde tiene que rebotar el rayo cuando choca contra un objeto. Estos samplers toman como parámetro el exponente de Phong, que toman valores sumamente altos. El workaround fue pasar por parámetro  $1 \div (\text{roughness} * \text{roughness})$  y así, con valores muy chicos de rugosidad, el exponente crece, y así nos aproximamos a un reflejo perfecto.

El único problema es que al ser exponencial, los valores se empiezan a percibir cuando nos acercamos al 0.

## 4. Sampling

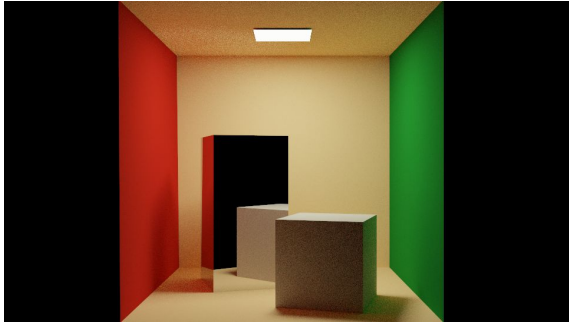
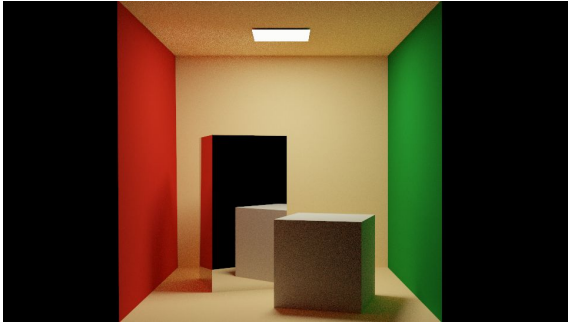
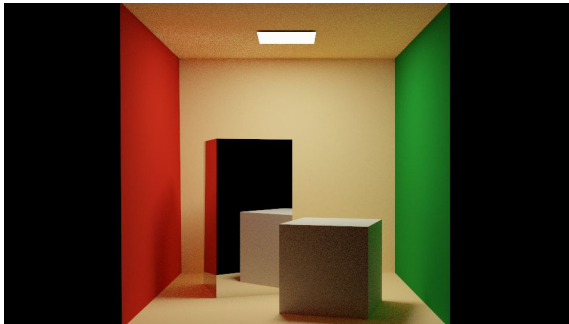
Al realizar las diversas pruebas se puede ver claramente que el ruido se reduce considerablemente al incrementar la cantidad de muestras por píxel. De la misma forma el tiempo de render incrementa exponencialmente. A partir de 1.000 muestras por píxel, el ruido percibido es mínimo, como se puede ver a continuación.



Para el muestreo se implementaron los siguientes tipos:

- NRooks
- Jittered
- MultiJittered

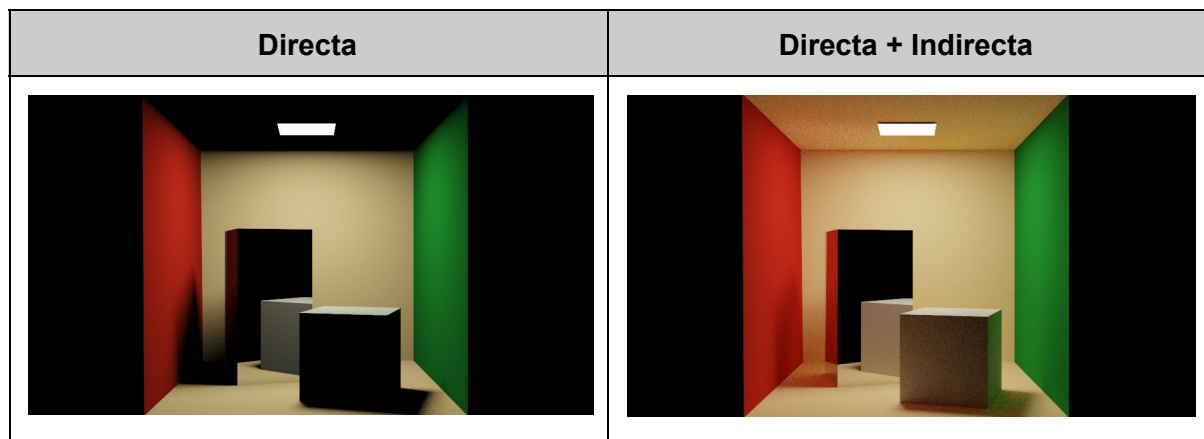
Se pueden ver en las imágenes a continuación (100 samples por píxel, 10k samples generar por sampler). Cabe aclarar que si se genera suficiente cantidad de muestras para utilizar (los samplers generan una cantidad de muestras en la etapa de preprocesamiento), la diferencia no es notoria.

| NRooks   | Jittered   |
|--|--|
|   |  |
| MultiJittered  |  |
|  |  |



## 5. Iluminación directa vs. Iluminación indirecta

A continuación se puede ver el render de la Cornell-Box realizado sólo bajo luz directa y bajo luz directa + indirecta. A simple vista se puede percibir que solo con luz directa no tenemos color bleeding, cáusticas e iluminación global. Debido a que la función de visibilidad es lo único que se computa a la hora de impactar un objeto, todos esos “efectos” no son posibles solo con luz directa. Combinando ambas, se pueden ver en la cornell-box presenta color bleeding incluso con pocas samples por pixel.



## 6. Parámetros de consola

Se siguen manteniendo los parámetros de consola anteriores. Sólo se agregó el parámetro

**-hb**

**(Opcional)** Imprime en salida estándar cada vez que se completa un bucket. Solo funciona fuera del modo Benchmark y modo UI.

## 7. Parseo de escenas

Debido a la flexibilidad que ofrecía el diseño del parser en el Trabajo Práctico anterior, no fue necesario realizar grandes cambios en el mismo para soportar las nuevas funcionalidades. Las mismas restricciones explicadas en el informe anterior siguen existiendo en esta versión.

Se agregó el identificador `AreaLightSource` para definir Area Lights. Dicho identificador debe estar seguido de un identificador `Shape` para definir la forma del Area Light. Actualmente, sólo se da soporte para esferas, cajas y meshes. Se le aplica automáticamente un material emisor de luz a la figura. En caso de tratarse de un Mesh, se aplica el último material definido a las caras traseras de los triángulos. Si se hallara una figura no soportada para Area Lights (por ejemplo, planos infinitos), el Area Light es ignorado y la figura funciona como cualquier objeto definido anteriormente.

La cantidad de muestras utilizadas para Area Lights se definen con la propiedad `nsamples`. La documentación de LuxRender indica que esta es una sugerencia para el motor. Como se desconoce el criterio que utiliza LuxRender para determinar la cantidad de muestras, se decidió que este valor determina la cantidad de muestras, sin ponderar. Se recomienda mantener este valor en 1 para Path Tracing, e incrementarlo para Ray Tracing.

Al ser necesarias Area Lights para obtener resultados satisfactorios en Path Tracing, se reemplazan las luces puntuales por luces esféricas de poco radio al cargar una escena para Path Tracing.

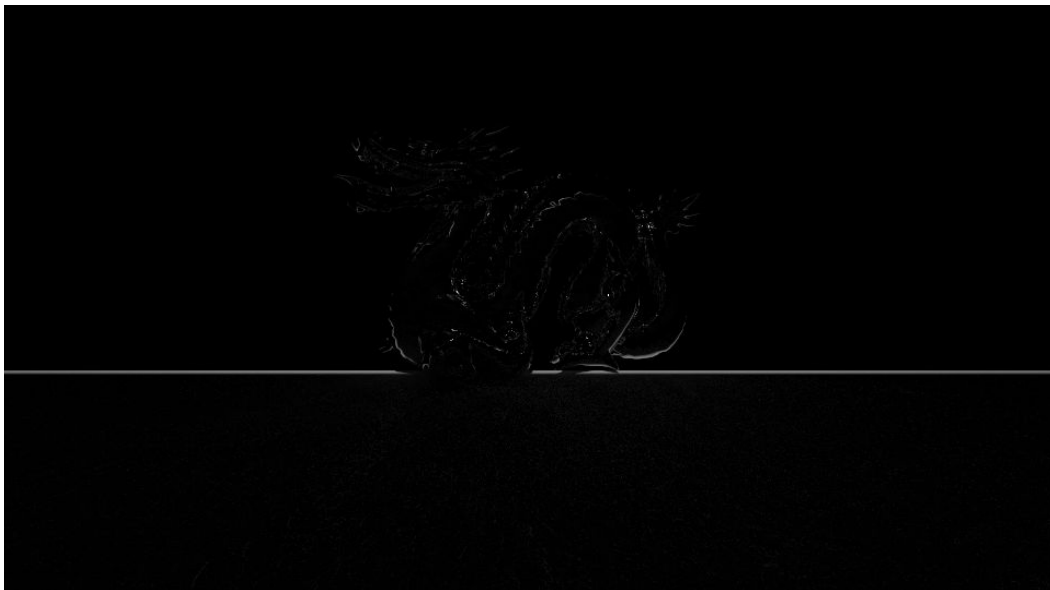
Se utiliza la propiedad `tonemapkernel` en el identificador `Film` para elegir el operador de Tone Mapping. Sólo se le da soporte a los operadores `linear` y `reinhard`. En caso de hallarse otro, se elige `linear` por default. Ninguno de los operadores de Tone Mapping acepta parámetros. Todos son ignorados por el motor.

## 8. Funcionalidades adicionales

Estas funcionalidades no fueron pedidas explícitamente en el enunciado, pero el motor las soporta ya sea por ser resultado indirecto de Path Tracing, o para obtener resultados similares a los del enunciado.

### Cáusticas

No fue necesario implementar cáusticas, ya que el mismo Path Tracing las produce. Sin embargo, las cáusticas son débiles y extremadamente ruidosas, incluso con una gran cantidad de muestras por píxel. Se debe emplear otro método de Path Tracing para obtener cáusticas mejor definidas.



### Gamma Correction

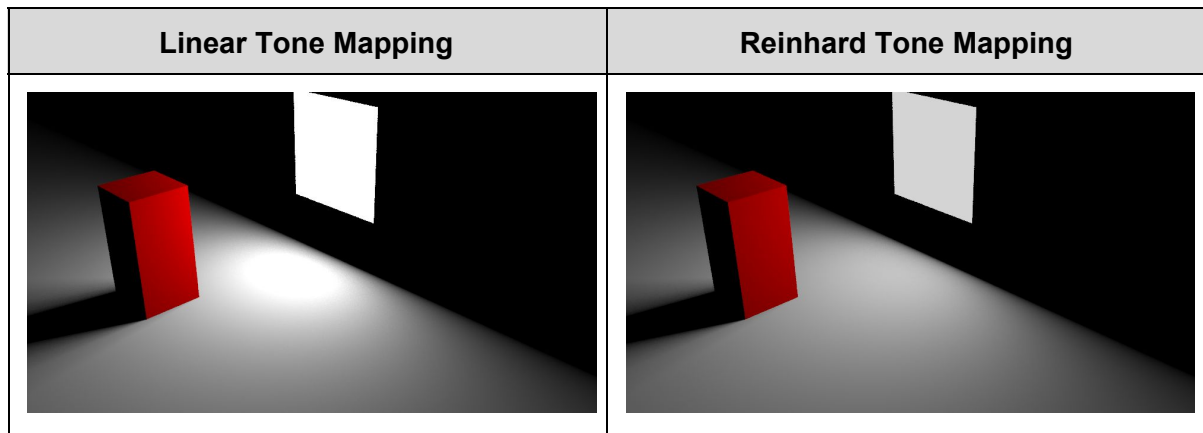
La escena de Cornell Box resultaba muy oscura y lejana a la referencia. Se implementó Gamma Correction, lo cual además de ser sencillo, incrementó el realismo de las escenas. Todas las escenas utilizadas para el Trabajo Práctico anterior se benefician de esto.

### Tone Mapping

Se notó que el color de los píxeles cercanos a las fuentes de luz se salían del rango visible, resultando en imágenes muy saturadas. Además de utilizar el método de tone mapping actual, se implementó una forma aproximada del operador Reinhard. La misma aplica a los canales del color de cada píxel la siguiente fórmula

$$f(x) = \frac{x}{1+x}$$

Esto resultó en imágenes con colores más tenues pero dentro del rango visible.



## 9. Escenas de prueba

Todas las escenas que se muestran en este informe fueron generadas con una PC con las siguientes características:

- **Sistema operativo:** Debian 8 de 64 bits
- **CPU:** AMD FX-8150 de 8 núcleos @ 3.60Ghz
- **RAM:** 16GB
- **JVM:** OpenJDK 8

Debido a la gran cantidad de tiempo que tardan en generarse estas escenas y el poco tiempo con el que se cuenta, se decidió realizar una única corrida. Todas las escenas fueron generadas con un Trace Depth de 10 saltos. Se incluyen en este trabajo las escenas provistas por la cátedra, con leves modificaciones.

| N° | Archivo   | Comentario   |
|----|---|--|
| 1  | CornellBox/cornell-box.lxs  | Se definió el material de la caja menor, a pedido de la cátedra. Se cambió el método de Tone Mapping por Reinhard para evitar la saturación.   |
| 2  | Sponza/sponza.lxs   | Se invirtieron las normales del Area Light, ya que la misma apuntaba en sentido contrario a la escena y producía una pantalla en negro. Luego, se redujo considerablemente la intensidad de dicha luz, ya que al invertir las normales se obtenía una pantalla en blanco debido a la iluminación excesiva              |
| 3  | Dragon/dragon.lxs   | Se invirtieron las normales de los triángulos del piso, ya que estaban al revés. Como no se implementaron parámetros avanzados para la cámara, como <i>exposure</i> , la escena resultaba muy oscura y las cáusticas casi imperceptibles. Se incrementó la intensidad de la luz para obtener resultados satisfactorios |
| 4  | Goku/goku.lxs   | --   |
| 5  | Pillar/pillar-square-large.lxs<br>Pillar/pillar-square-small.lxs<br>Pillar/pillar-sphere-large.lxs<br>Pillar/pillar-sphere-small.lxs<br>Pillar/pillar-point.lxs<br>Pillar/pillar-reinhard.lxs | Se proveen escenas con distintos tipos de luces. Todas las escenas utilizan Tone Mapping Linear, a excepción de la última variante. Las luces utilizan 100 muestras, se recomienda utilizar una sola muestra a la hora de renderizar con Path Tracing.   |

## Resultados

A continuación se encuentran los tiempos de renderizado de las escenas de prueba, para distintas cantidades de muestras por píxel. Las celdas en rojo representan casos en los cuales no se renderizó la escena debido a que el render anterior presenta poco ruido y demoró una cantidad excesiva de tiempo.

| Escena         | 10<br>muestras | 100<br>muestras | 1.000<br>muestras | 10.000<br>muestras |
|----------------|----------------|-----------------|-------------------|--------------------|
| Cornell Box    | 12s            | 2m 1s           | 19m 19s           | 3h 19m 38s         |
| Palacio Sponza | 2m 32s         | 27m 48s         | 4h 23m 32s        |                    |
| Dragón         | 6s             | 56s             | 8m 46s            | 1h 29m 14s         |
| Goku           | 1m 11s         | 12m 43s         | 2h 2m 13s         |                    |

## 10. Bibliografía

Heuristic Ray Shooting Algorithms

<http://dcgi.felk.cvut.cz/home/havran/DISSVH/dissvh.pdf>

On building fast kd-Trees for Ray Tracing, and on doing that in  $O(N \log N)$

<http://dcgi.felk.cvut.cz/home/havran/ARTICLES/ingo06rtKdtree.pdf>

Ray Tracing with the BSP Tree

[http://www.cs.utah.edu/~thiago/papers/BSP\\_RT08.pdf](http://www.cs.utah.edu/~thiago/papers/BSP_RT08.pdf)

Ray-Triangle Intersection Algorithm for Modern CPU Architectures

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.189.5084&rep=rep1&type=pdf>

Ray Tracing from the ground up

<http://www.raytracegroundup.com/>

Physically Based Rendering, from theory to implementation. Second Edition. Matt Pharr & Greg Humphreys.