Computación Gráfica

Informe de Trabajo práctico Nº 2

Path Tracing

Federico Tedin - 53048
Javier Fraire - 53023

Junio 2016 - C1

Decisiones de Diseño:

Transición de Ray Tracing a Path Tracing:

El path tracer realizado para este trabajo práctico está basado en el código del trabajo anterior (raytracer), ya que la mayoría de las componentes son compartidas entre los dos tipos de sistemas. Esto nos permitió tener un path tracer que también puede generar imágenes utilizando ray tracing, cumpliendo asi uno de los requisitos adicionales, y también permitiéndonos ahorrar tiempo al no tener que re-implementar varios algoritmos y estructuras de datos.

Los cambios realizados al código, en breve, fueron los siguientes:

- Se agrego el material de tipo Cook-Torrance, el cual es uno de los requisitos requeridos.
- Se agrego el material de tipo Emissive, el cual tiene una propiedad de intensidad, lo cual lo distingue con el material ya existente de tipo Color. El propósito del material Emissive es ser utilizado en las luces con área.
- Se agregaron nuevos tipos de luces, con área: Sphere, Rectangle y Dome.
- Se modificó **SceneParser** para poder interpretar los nuevos tipos de materiales y luces.
- Se modificó la clase **MultiJitteredSampler**, y se agregó un sistema de sub-sampling. También, se creó un sistema para administrar reservas de muestras globales, las cuales son utilizadas en distintos puntos del proyecto.
- Se modificó la jerarquía de clases para poder lograr que **Light** pueda tener una representación visual, utilizando la clase **Primitive**.
- Se agregó un nuevo método a la clase **Material**, el cual calcula el color resultante de una superficie utilizando la técnica de path tracing, teniendo en cuenta todos los tipos de luces.
- Se agregó una implementación de la función BRDF en los materiales necesarios.
- Se mejoró la implementación del material refractivo.
- Se mejoró la implementación de la detección de colisiones para primitivas Box.

Cambios a la clase **Material**:

Para poder implementar path tracing, se decidió agregar un nuevo método abstracto a la clase **Material**, *traceSurfaceColor*, el cual calcula el color final de una superficie, generando otros rayos de ser necesario. Por esta razón, todos los materiales del proyecto tuvieron que ser modificados para implementar el nuevo método. El método *getSurfaceColor* se mantuvo, el cual es utilizado cuando se desea utilizar raytracing.

La mayor diferencia entre los métodos *traceSurfaceColor* y *getSurfaceColor* es que en *traceSurfaceColor* se tiene en cuenta (para ciertos materiales) la iluminación directa e indirecta. Para la iluminación directa, se utilizó la misma lógica que en *getSurfaceColor*. Por el otro lado, para la iluminación indirecta, simplemente se instancia un nuevo rayo, con una direccion calculada a partir de una muestra al azar y la función BRDF del material, y se traza el rayo contra la escena. Si el rayo colisiona con otra primitiva, se repite el proceso recursivamente hasta que el rayo alcanza su máxima cantidad de saltos posibles. Una vez terminado de colisionar el rayo contra la escena, se obtiene un color como resultado, el cual es combinado con el color del material que lanzó el rayo, para obtener un color final.

La siguiente tabla describe cuáles materiales utilizan iluminación directa e indirecta:

Sólo Iluminación Directa	Sólo Iluminación Indirecta	Iluminación Directa e Indirecta
Color Emissive	Refractive Reflective	Cook-Torrance Diffuse Phong

A su vez, en algunos de los materiales se implementaron las funciones *BRDF* y *sample*. *BRDF* retorna el valor por el cual se debe multiplicar al color del material y el método *sample* retorna un vector con la dirección en la que se debe tirar el siguiente rayo. De esta forma, se logró evitar repetir código ya que en todos los cálculos de luz se utilizó la función *BRDF*. En los materiales Cook-Torrance, Diffuse y Phong se puede observar que la estructura del método *traceSurfaceColor* es la misma. Más aún para Cook-Torrance y Phong en los que se calcula por un lado la componente difusa y por otro la especular. Esto se podría haber abstraído utilizando lambdas para evitar repetir código, pero se decidió no realizar dicha abstracción por performance ya que se agrega un *overhead* y más cambios en el pipeline al utilizar lambdas. También se podría haber hecho que Phong y Cook-Torrance hereden de la misma clase y

solamente sobrescriban los métodos necesarios pero no se optó por esta opción ya que se quiso evitar el acoplamiento.

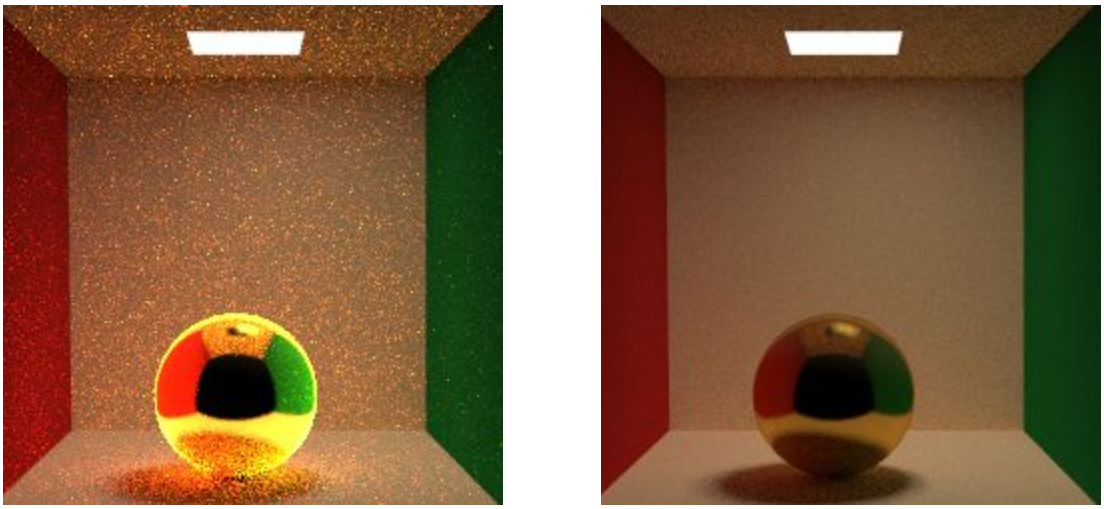
Implementación de Cook-Torrance:

Para Cook-Torrance se decidió utilizar la aproximación de Schlick en lugar de Fresnel ya que el costo computacional es menor y los resultados son muy similares. A su vez, se puede observar que se cuenta con dos implementaciones de Schlick. Este se debe a que una recibe los dos índices de refracción y utiliza un coseno distinto según cuál sea mayor, y la otra recibe un solo índice de refracción y utiliza el aire como el otro índice de refracción. Esto se realizó para evitar la pregunta de qué índice de refracción es mayor.

En el caso de la función de distribución se utilizó Beckman por dos motivos. Primero, es la función recomendada por Cook y Torrance en el paper "*A Reflectance Graphics*". Segundo, es físicamente más correcta que Phong y Gauss.

Para obtener un *sample* de la dirección en la que saldría el rayo, se utilizó el mismo método que en Phong. Para ello, se convirtió el *roughness* al exponente de phong utilizando la siguiente fórmula $\frac{2}{roughness^2} - 2$. La muestra luego es proyectada sobre el vector reflexión.

Otra decisión que se tomó fue utilizar como índice de refracción el valor del Cromo (3.0504) ya que su resultado es visualmente agradable. A su vez, al observar que para valores de *roughness* muy chicos la BRDF daba valores muy altos, se decidió realizar un *clamp* sobre la misma. Esto se debe a que la distribución de Beckman utiliza $roughness^2$ en el denominador causando que la variable D sea muy elevada. En los siguientes *renders* se puede observar utilizar *clamp (derecha)* contra no utilizarlo (*izquierda).*



Comparación de utilizar clamp en Cook Torrance. La imagen izquierda no utiliza clamp. La derecha si.

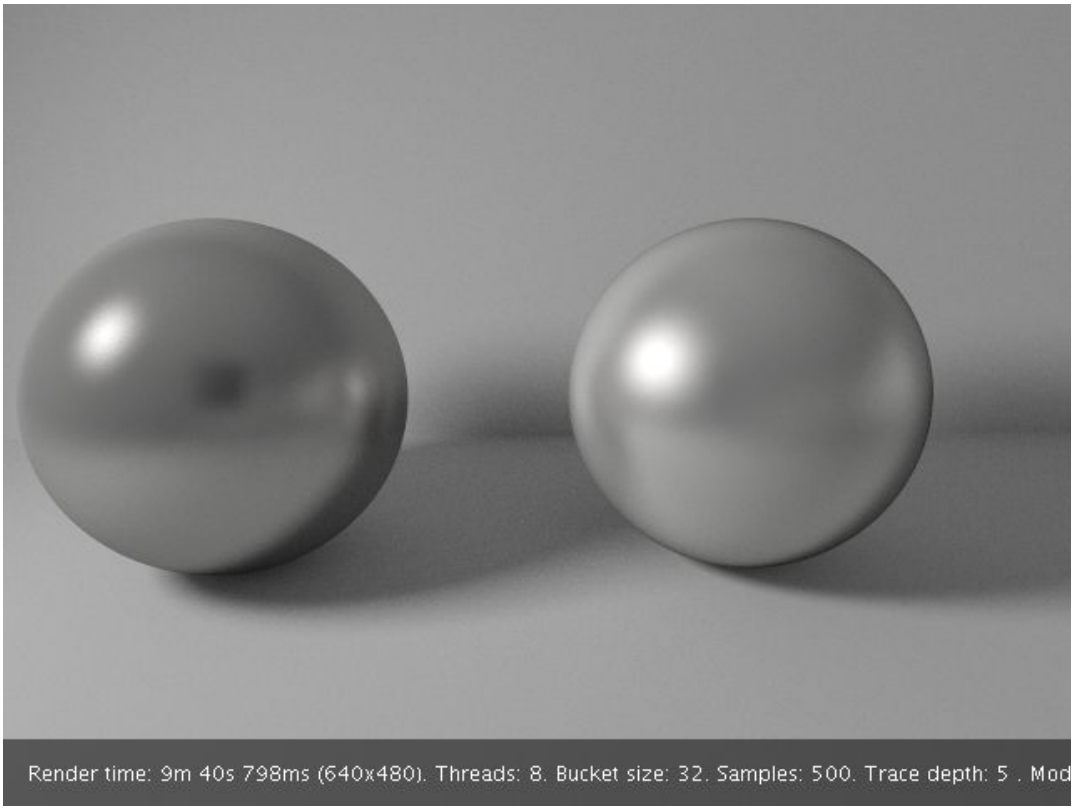
Es importante mencionar que en algunos casos la suma de la componente difusa más la componente Cook-Torrance pueden sumar un número mayor a uno. Para evitar esto, conviene elegir adecuadamente los colores en la escena para no llegar a dicha situación. En varios artículos se utilizaba una constante para definir que porcentaje era luz difusa y que porcentaje luz especular pero no se realizó dicha implementación porque no nos pareció correcto elegir un número de forma arbitraria.

Tanto en Cook-Torrance como en Phong, no se implementó importance sampling ya que no se logró una implementación que funcione adecuadamente.

La siguiente imagen muestra una comparación entre distintos valores de *roughness*. Comenzando de la izquierda los valores son: 0.05, 0.15, 0.3, 0.5, 0.7 y 0.9. Como se puede observar los valores más bajos son más metálicos.

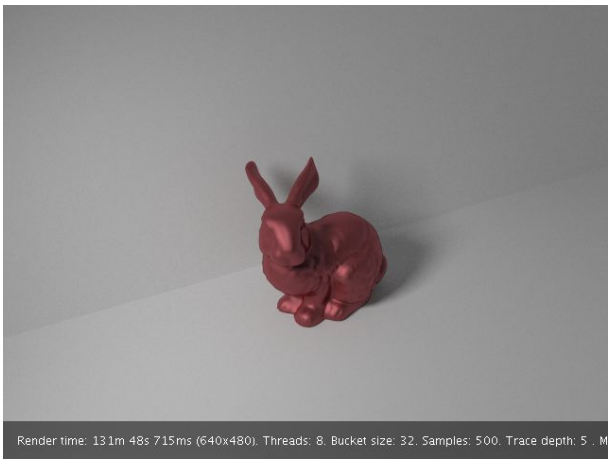
Cook Torrance Spheres (1). Roughness de izquierda a derecha: 0.05, 0.15, 0.3, 0.5, 0.7 y 0.9

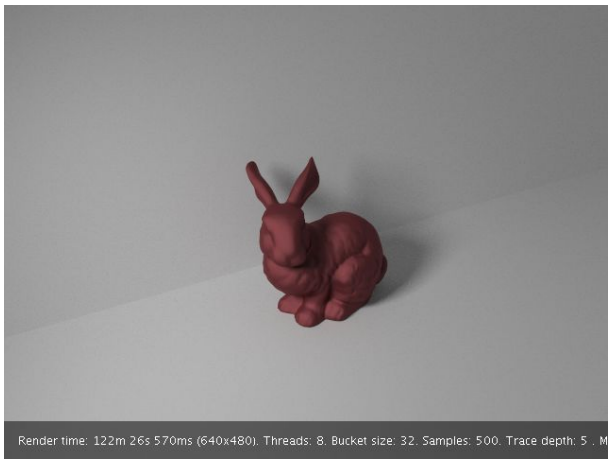
La siguiente imagen muestra una comparación de Phong y Cook-Torrance. Ambos tienen el mismo color difuso y especular y los exponentes son los mismos (0.2 *roughness* y 52 exponente phong). Como se puede observar, Cook-Torrance tiene una apariencia más metálica.



Cook Vs Phong (2). La imagen izquierda utiliza Phong y la derecha Cook-Torrance

Las siguientes imágenes muestran la diferencia entre usar *roughness* de 0.3 contra 0.5. La imagen de la izquierda utiliza un *roughness* de 0.3 y la de la derecha de 0.5. Se puede observar que la de la izquierda es más reflectiva y más metálica que la de la derecha y por ese motivo, genera un mayor color bleeding sobre el piso y la pared.





Izq: Cook Torrance Bunny (3). Der: Cook Torrance Bunny 2 (4).

A continuación se muestra una lata que utiliza Cook-Torrance con *roughness* de 0.2.

Pepsi (5)

Cambios a la Generación de Muestras:

Al comenzar con el trabajo práctico, se encontraron dos problemas con el sistema de muestras implementado anteriormente: no era performante para números grandes de muestras, y no resultaba fácil ni práctico de usar cuando se necesitaba generar muestras desde distintos contextos de código dentro del proyecto. Por un lado, la cantidad de muestras necesarias para poder obtener resultados visualmente satisfactorios en este trabajo práctico resultó ser mayor a la del trabajo anterior: como ejemplo, normalmente se especifica un número de muestras por píxel mayor en path tracing que en raytracing (por ejemplo, 1500 y 64 respectivamente). Por el otro lado, se requirió el uso de muestras en distintos lugares en vez de sólo en la generación de rayos iniciales: las luces con área requieren de muestras para poder contribuir a la iluminación directa, y la iluminación indirecta requiere de muestras aleatorias para poder generar nuevos rayos.

Por éstas dos razones, se tuvo re-pensar cómo se generan las muestras en el momento de hacer path tracing. Utilizer un *sampler* por pixel para 1000 muestras significaba más de 30 minutos de generación de muestras para una imagen de 1280x720. Finalmente, se decidió crear un sistema de caches de muestras globales. Al iniciarse el programa, se generan varios **MultiJitteredSampler**s de gran tamaño, que son administrados por la clase **SamplerCacheQueue**. En cualquier punto del código, si se requieren muestras, se toma un sampler de la cola de samplers. Una vez reservado el sampler, se utiliza la nueva clase **SubSampler**, la cual permite copiar cualquier cantidad de muestras del sampler original a otra ubicacion temporal. El sampler base luego se devuelve a la cola de samplers, para poder ser utilizado en otro contexto y momento.

De esta forma, se logró mantener un sistema performante, ya que la generación de muestras se realiza una sola vez al comienzo, pero a la vez manteniendo aleatoriedad, ya que un gran número de muestras es generado, y las muestras en si son tomadas aleatoriamente.

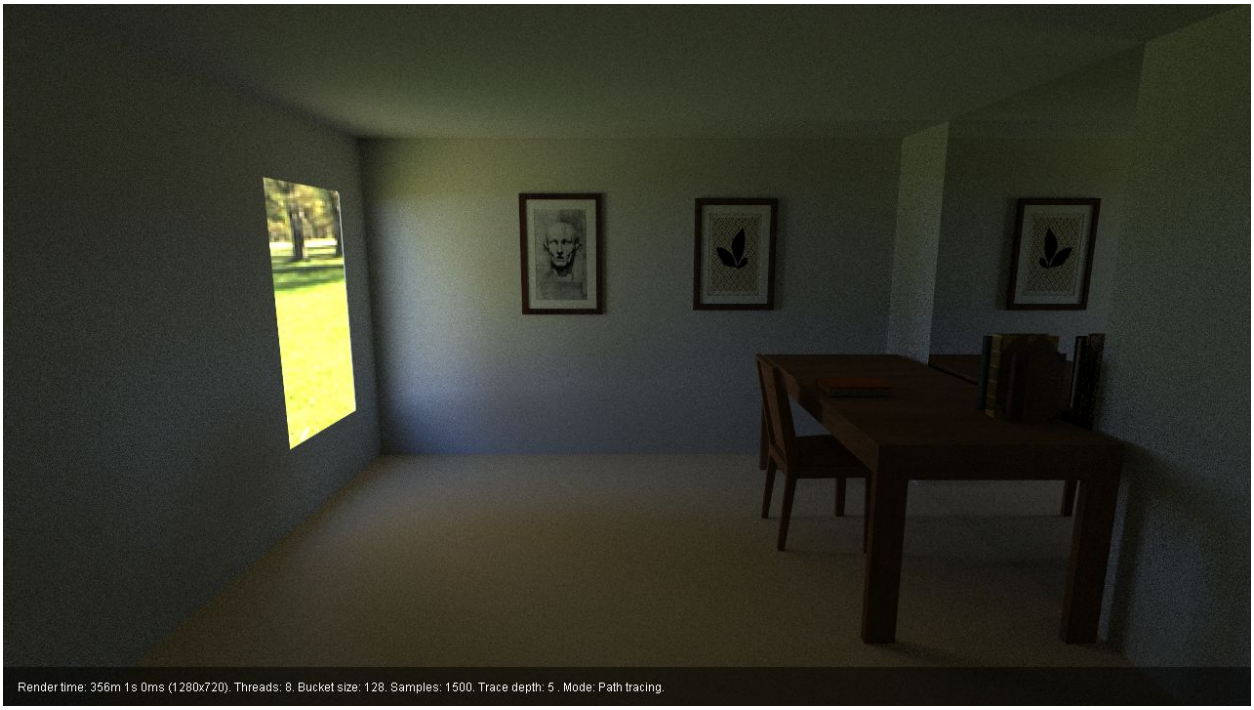
Luces:

En el caso de las *area lights* rectangulares, se multiplicó la intensidad por el producto escalar entre el vector superficie-luz y la normal de la luz. De esta forma, de acuerdo al ángulo se recibe más o menos intensidad, lo que genera una mayor oclusión. Otra decisión que se tomó fue descartar la luz indirecta en el caso en el que el rayo de la misma colisione con la misma luz que se utilizó en la luz directa. De esta forma, se evita utilizar dos veces a la misma luz. Esto ayuda a reducir la cantidad de pixeles muy brillantes (*speckles)*.

En el caso de las *dome lights* se decidió que las mismas abarquen una esfera y no únicamente un hemisferio de 180 grados. Esto se debe a que la mayoría de las texturas vienen preparadas para ser utilizadas en 360 grados. Además, sería muy extraño que en una escena se observe mitad de una textura y mitad de color negro.

En el caso de que se cuente con más de una luz, se elige una al azar para cada colisión. Además, se utilizó ruleta rusa con probabilidad de 0.05 para que exista la posibilidad de que un rayo no siga el camino completo.

La siguiente imagen utiliza únicamente una *dome* light como fuente de luz. La misma cuenta con una textura de un parque y una ventana por la que entra la luz. El índice de refracción de la misma es 1.53. También se puede observar que la luz toma un color verde debido al color de la textura y se observan *soft shadows.*



Window 2 (6)

A continuación se muestran dos imágenes Cornell Box. A la izquierda se observa con una luz rectangular y a la derecha con una luz esférica. Se puede observar que la luz esférica ilumina más. Esto se debe que al emitir luz en 360º, más objetos reciben luz directa. Es importante mencionar que la caja tiene un material ligeramente más oscuro al provisto por la cátedra. Esto ocurrió accidentalmente y recién se descubrió una vez realizados todos los *renders.*

Izq: Cornell Box Gamma (7). Der: Cornell Box Sphere Light(8).

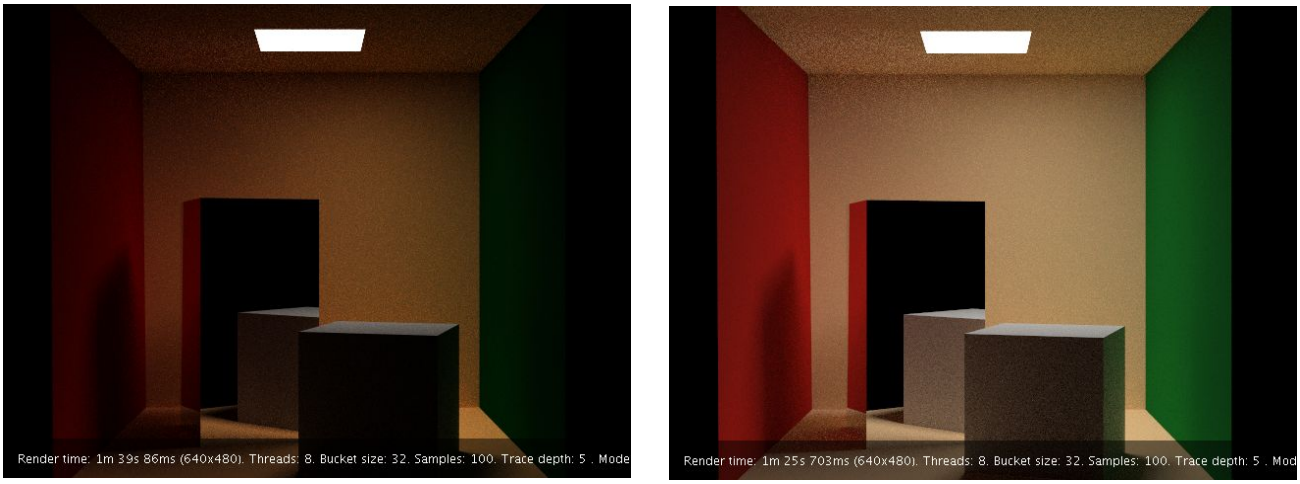
La siguiente imagen muestra una esfera reflectiva con una *dome light* con una textura.

Dome Light (9)

Requerimientos Opcionales:

Como opcional se decidió implementar *gamma correction* ya que la misma mejora mucho las imagenes. Al utilizar dicha mejora, las imágenes son más agradables visualmente ya que que se contrarrestan los cambios implementados por el monitor. En el caso de las texturas, se le aplica la corrección inversa si estas son de tipo sRGB ya que las mismas se encuentran gamma corregidas. A continuación se mostrarán algunos ejemplos de la aplicación de la corrección gamma.

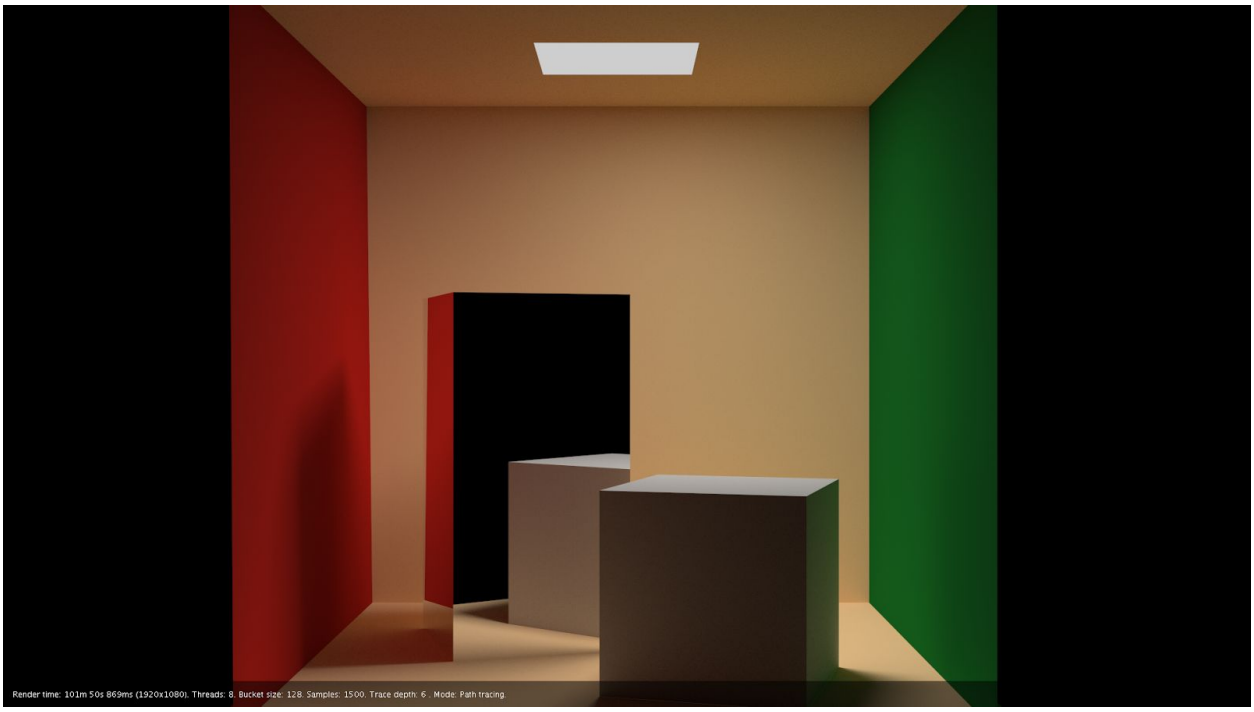
Las siguientes imágenes muestran la diferencia entre aplicar *gamma correction* y no aplicarla. La imagen izquierda no aplica *gamma correction*. Como se puede observar, la imagen que cuenta con *gamma correction* cuenta con colores más claros y es más agradable visualmente. Por este motivo se decidio que por *default* se aplique gamma correction (a menos que se especifique lo contrario). Además, se elimina el color bleeding excesivo, sobre todo en en las zonas más iluminadas. En la realidad el color bleeding se observa pero no de forma excesiva como en la imagen de la izquierda. También se puede observar que se obtuvo poco ruido con tan solo 100 muestras.



Izquierda: Cornell Box No Gamma (10), Derecha: Cornell Box Gamma (7)

Más *renders* y análisis:

En la siguiente imagen se pueden observar diversas funcionalidades del trabajo. Por un lado se puede observar el *color bleeding* sobre el cubo blanco, sobre el techo y sobre las paredes. A su vez, se puede observar la oclusión en las esquinas y se pueden observar *soft shadows*. También se puede destacar que la imagen no cuenta con ruido. Esto se debe a la técnica de sampling utilizada.

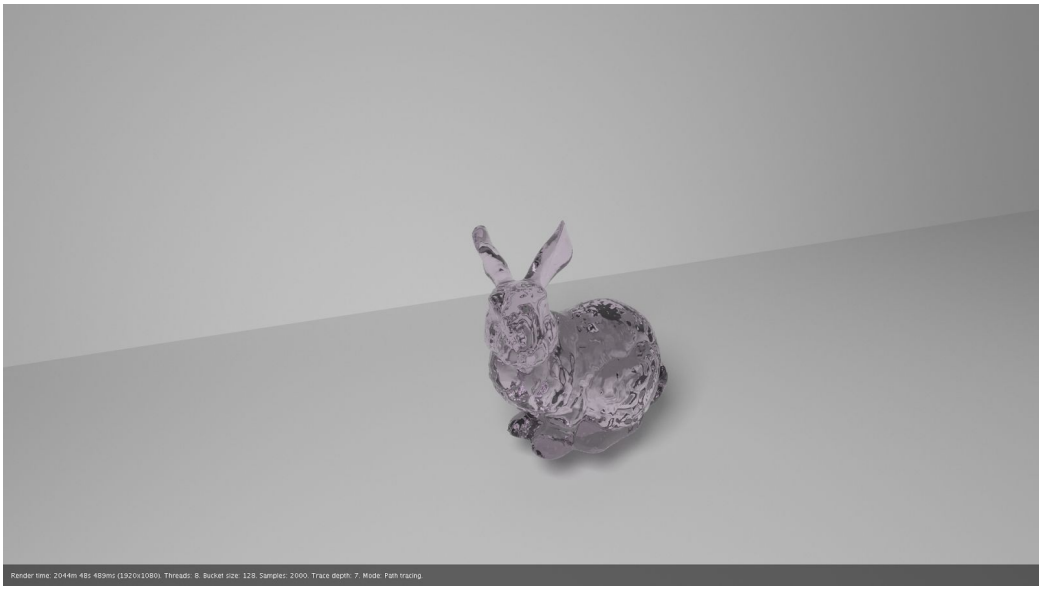


Cornell Box (11)

La siguiente imagen es similar a *Window 2*, pero en lugar de contar con una textura, cuenta con una *dome light* de color blanco.

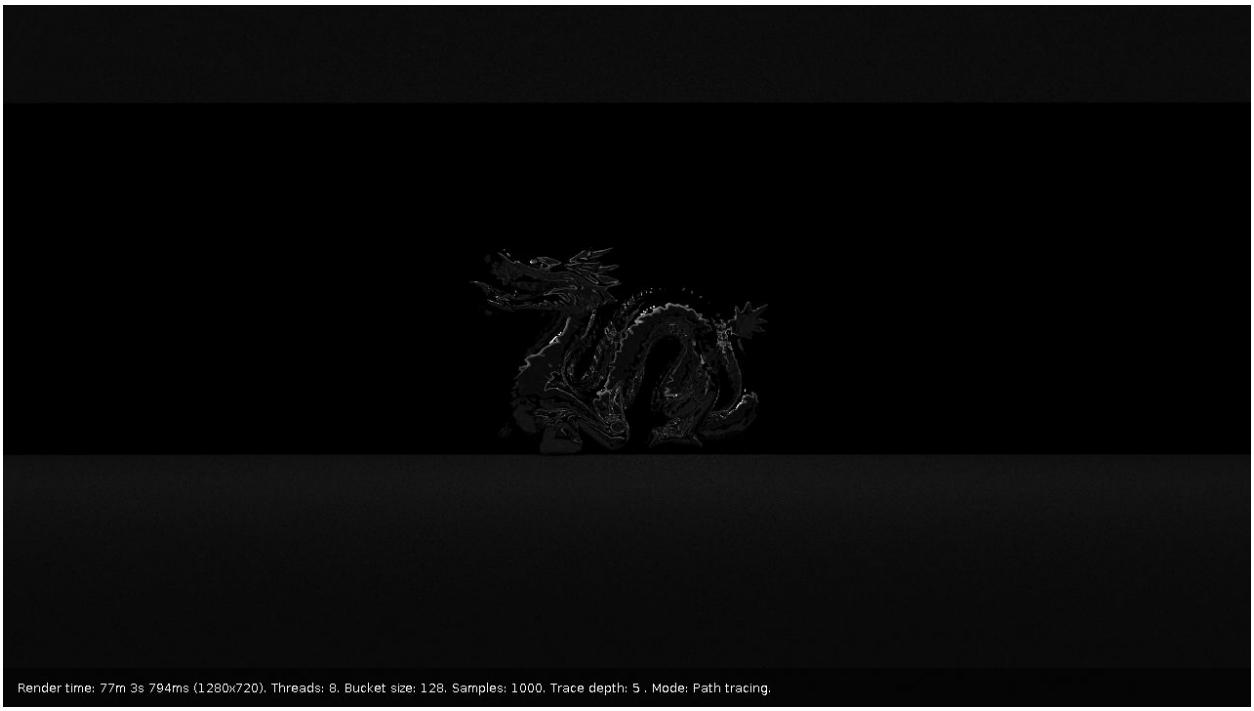
Window (12)

La siguiente imagen muestra un conejo refractivo de color rosa claro iluminado por 3 *area lights* rectangulares. Este nivel de detalle no hubiera sido posible lograrlo utilizando *ray tracing*. Se observa que la escena adquiere un ligero tono rosa. A su vez, al utilizar una gran cantidad de *samples*, no se observa ruido. Se utilizó un *max trace depth* de 7, lo que permitió que la escena obtenga más iluminación y más cáusticas. También se puede observar la *soft shadow* debajo del conejo.



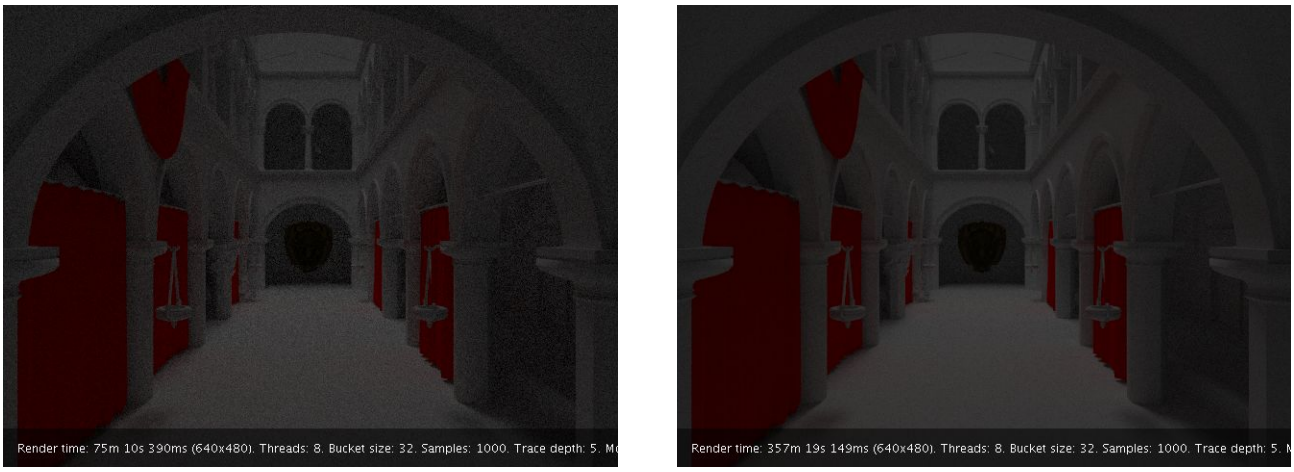
Path Refraction (13)

La siguiente imagen es un *render* del dragón chino provisto por la cátedra. Se puede observar ruido ya que es una escena muy oscura. A su vez se puede observar que es una escena iluminada puramente por luz indirecta ya que la luz se encuentra obstruida. Se decidió reemplazar la *point light* por una *sphere light*, ya que esta funciona mejor para *path tracing* debido a que los objetos pueden impactar contra la misma al calcular la luz indirecta.



Caustics (14)

Las siguiente imágenes es son *renders* de *sponza* utilizando 100, 500 y 1000 muestras. Es muy notable la diferencia de ruido en las distintas imágenes. Siendo mucho más ruidosa la de 100 muestras. También se puede observar el *color bleeding* rojo. A su vez, se observa que está un poco oscura la escena pues nos olvidamos de aumentar la intensidad.



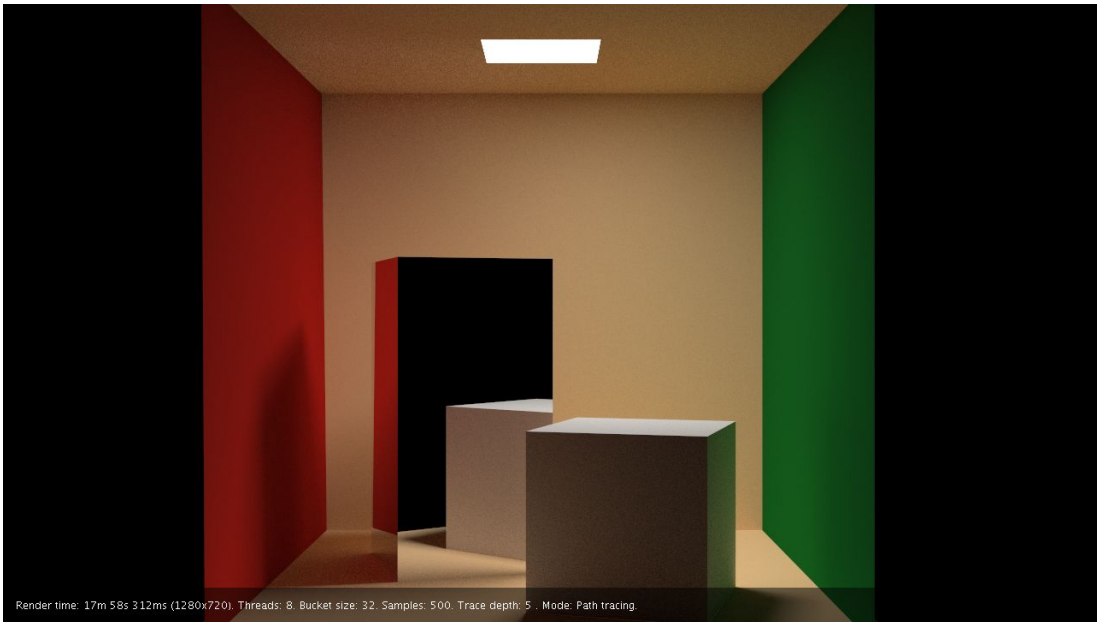
Izq: Sponza-100(15). Der: Sponza-500 (16)

Sponza-1000 (17)

La imagen debajo muestra una figura con material Phong, generada utilizando la técnica de *path tracing*. Se observa que tiene ruido ya que solamente tiene 100 muestras. También se puede observar el *bleeding* rojo.

Angel (18)

Como se mencionó anteriormente, accidentalmente se modificó el color de la caja en Cornell Box. A continuación se muestra una imagen del original:



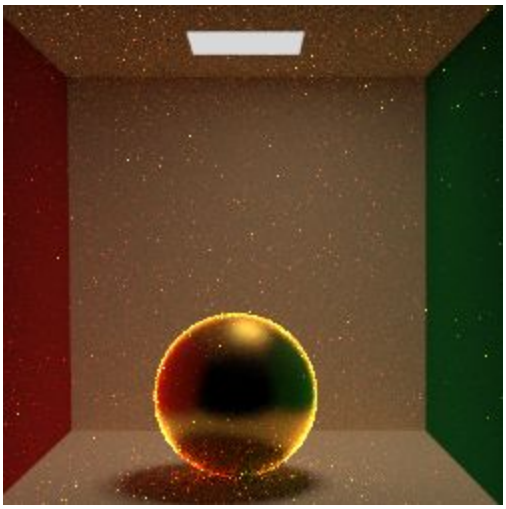
Cornell Box 2 (19)

Como se puede observar, no hay mucha diferencia con los demás renders de *Cornell Box*.

Dentro de la carpeta ‘renders’, en los archivos del trabajo, se pueden encontrar estas imágenes en su resolución completa y otras imágenes más que se omitieron del informe.

Problemas encontrados:

En Cook-Torrance se encontró otro además del mencionado anteriormente. Se observaba que el material era muy brilloso y generaba muchos *speckles*. En la siguiente imagen se observa lo mencionado:



Esto se debía a que no se estaba multiplicando por el coseno del ángulo entre la normal y la luz incidente.

Otro problema que se encontró fue con el material refractivo, en especial en las cajas. Este era un problema del primer trabajo. Un problema que había era que el algoritmo de detección de colisiones de cajas detectaba como colisión aún cuando el rayo estaba adentro de la misma. Esto ocasionó que se viera de color negro ya que se agotaban los saltos dentro de la caja. Se solucionó arreglando el algoritmo de colisión de cajas haciendo que la distancia mínima (t0) comience en menos infinito y si esta es menor que 0, que se retorne el t1.

A su vez, se tuvieron diversos problemas al implementar los materiales como errores de cuentas, de signo, etc. Se encontraron y solucionaron luego de muchas horas de *debugging*.

Benchmarks

A continuación se presentan los *benchmarks* de las imágenes presentes en el informe.

Nombre	Size	Samples	Threads	Bucket Size	Max Trace Depth	Hardware	Tiempo
CookTorranceSpheres (1)	640x480	500	8	32	5	Core i7 2.6ghz (3720QM)	8m 17s 894ms
CookVsPhong (2)	640x480	500	8	32	5	Core i7 2.6ghz (3720QM)	9m 40s 798ms
CookTorranceBunny (3)	640x480	500	8	32	5	Core i7 2.6ghz (3720QM)	131m 48s 715ms
CookTorranceBunny 2 (4)	640x480	500	8	32	5	Core i7 2.6ghz (3720QM)	122m 26s 570ms
Pepsi (5)	1280x720	1500	8	32	5	Core i7 2.6ghz (3720QM)	557m 32s 162ms
Window 2 (6)	1280x720	1500	8	128	5	Core i5 3.2ghz (4460K)	356m 1s 0ms
CornellBox Gamma (7)	640x480	100	8	32	5	Core i7 2.6ghz (3720QM)	1m 25s 703ms

CornellBoxSphereLight (8)	640x480	100	8	32	5	Core i7 2.6ghz (3720QM)	1m 36s 270ms
DomeLight (9)	1920x1080	500	8	32	5	Core i7 2.6ghz (3720QM)	4m 40s 828ms
CornellBoxNoGamma (10)	640x480	100	8	32	5	Core i7 2.6ghz (3720QM)	1m 39s 86ms
CornellBox (11)	1920x1080	1500	8	128	6	Core i7 2.6ghz (3720QM)	101m 50s 869ms
Window (12)	1280x720	1500	8	128	5	Core i7 2.6ghz (3720QM)	336m 122s 124ms
PathRefraction (13)	1920x1080	2000	8	128	7	Core i7 3.5ghz (4770K)	2044m 48s 489ms
Caustics (14)	1280x720	1000	8	128	5	Core i7 3.5ghz (4770K)	77m 3s 794ms
Sponza-100 (15)	640x480	100	8	32	5	Core i7 3.5ghz (4770K)	75m 10s 390ms
Sponza-500 (16)	640x480	500	8	32	5	Core i7 3.5ghz (4770K)	357m 19s 149ms

Sponza-1000 (17)	640x480	1000	8	32	5	Core i7 3.5ghz (4770K)	808m 16s 520ms
Angel (18)	1280x720	100	8	30	5	Core i5 3.2ghz (4460K)	164m 26s 939ms
CornellBox2 (19)	640x480	500	8	32	5	Core i7 2.6ghz (3720QM)	336m 52s 124ms

Bibliografía:

- Cook Torrance
  - http://ruh.li/GraphicsCookTorrance.html

- A Reflectance Model for Computer Graphics - Robert L. Cook y Kenneth E. Torrance - (1 de enero, 2006)
  - http://inst.cs.berkeley.edu/~cs294-13/fa09/lectures/cookpaper.pdf

- Physically Based Shading and Image Based Lighting - (4 de abril, 2015)
  - http://www.trentreed.net/blog/physically-based-shading-and-image-based-lighting/
- Physically Based Rendering: From Theory to Implementation - Second edition - Greg Humphreys y Matt Pharr (Julio 2010)
- Ray Tracing from the Ground Up - Kevin Suffern (2007)