Práctica 2

Trazador de rayos

Julio Martín Saez y Raquel Peces Muñoz

Contenido

Práctica 2 – Trazador de rayos

Julio Martín Saez

Raquel Peces Muñoz

Funcionalidades implementadas

Características obligatorias

Algoritmo de ray casting e iluminación directa.

Hemos implementado un algoritmo básico de traza de rayos, en el cual se lanzan rayos por cada pixel de la pantalla a renderizar, y se calcula el aporte de cada una de las luces de la escena. El cálculo del color se ha realizado mediante la fórmula de iluminación de Phong.

En la imagen de la izquierda observamos el resultado el resultado de un render normal con OpenGL y a la derecha la imagen resultante del trazador de rayos realizado:

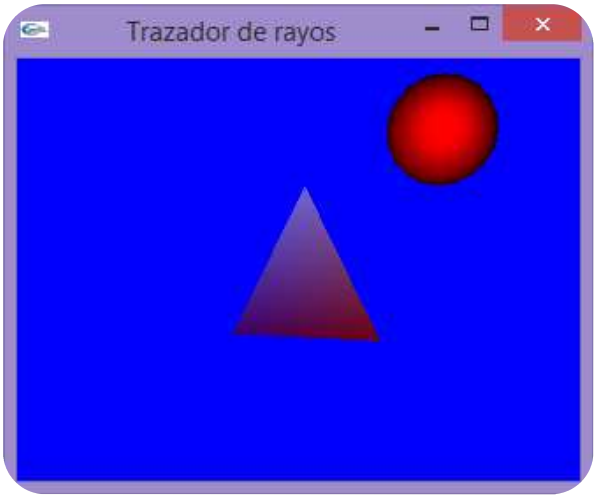

Imagen 1 : Render OpenGL

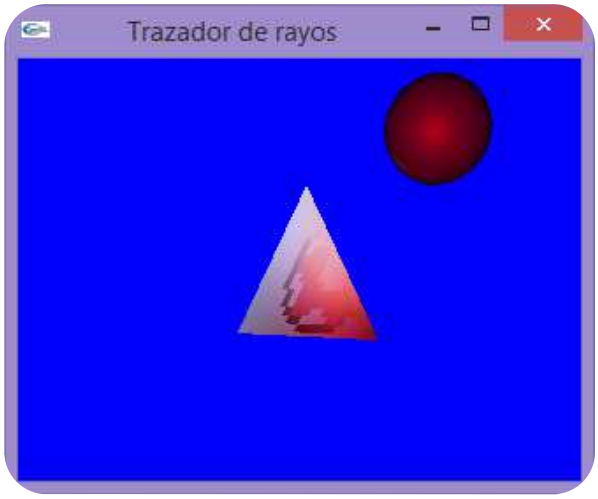

Imagen 2 : Traza de Rayos implementada

* En este caso hemos deshabilitado los objetos cubo del archivo *"test.xml"*, porque no eran imprescindibles en la parte obligatoria y así poder comparar mejor las dos imágenes obtenidas.

Incorporación de sombras.

En el momento del cálculo del color final del pixel, en el método de iluminación de Phong, se lanza un nuevo rayo desde el punto a cada una de las luces, de este modo podemos comprobar si el punto está oculto por otro objeto y si está en sombra o no.

El método "*collisionShadow*" es el encargado de esta comprobación, y devuelve un Vector, el cual puede tomar valores desde (1, 1, 1) en cuyo caso no hay sombra, hasta (0, 0, 0) en cuyo caso existirá una sombra completa. Si dicho Vector toma valores intermedios, es porque la sombra puede ser parcial al tratarse de un objeto "transparente". Podemos observar un resultado en la imagen 3 mostrada más abajo y definida en el archivo "test2.xml".

Imagen 3 : Sombras

Algoritmo de traza de rayos recursivo con 4 rebotes.

El algoritmo envía rayos reflejados con respecto a la normal. Para ello, una vez calculado el punto de intersección se calcula la normal en dicho punto y se envía un rayo en la dirección reflejada. Cada rayo tienen una variable "*numRebounds*" la cual almacena el número de rebotes restantes que le queda a cada rayo, por tanto, al enviar un rayo en la dirección reflejada dicha variable decrementa, y de este modo queda un valor menor de rebotes restantes.

Si el rayo no tiene más rebotes pendientes, devolverá el color correspondiente, mientras que en las iteraciones anteriores devolverán su color sumado, de forma ponderada por su índice de refracción.

El número de rebotes máximos por rayo se pueden variar, cambiando la constante definida *REBOUNDS* en la clase "*RayTrace*". En la imagen 4 se ve el reflejo del triángulo sobre la esfera, la escena está definida en "test3.xml":

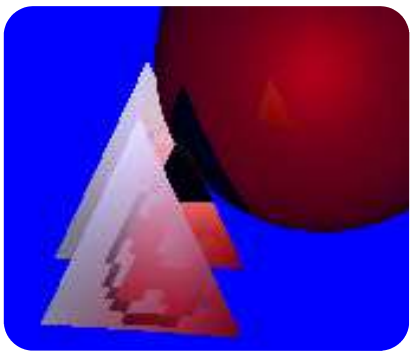

Imagen 4 : Reflexión

Geometría: esferas y triángulos individuales

Se ha implementado la colisión con esferas y triángulos individuales, utilizando las fórmulas de intersección de rayo con ambas primitivas.

También se han variado las clases *"SceneTriangle"* y *"SceneSphere"* para tener en cuenta las posibles transformaciones que se le aplican a dichos objetos como pueden ser translaciones, rotaciones y escalados. En dichas clases se han añadido métodos para obtener los valores correctos directamente.

Podemos ver el resultado de hacer varios objetos esfera y triángulo en la imagen 5, con sus correspondientes tamaños y posiciones y manteniendo los algoritmos anteriormente desarrollados, escena definida en *"test4.xml"*:

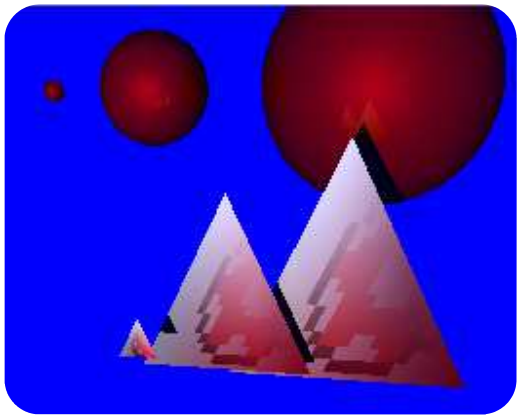

Imagen 5 : Objetos esfera y triángulo

Materiales: lambertianos, con reflexión tipo Phong y mapeado de texturas

Se ha implementado el mapeado de texturas, para ello se hace uso de las coordenadas u, v y el método del material que obtiene el color de la textura. Las texturas se aplican en la componente difusa.

Podemos ver los resultados en las imágenes 6 y 7, en la 6 podemos observar una textura propia llamada *"ejemplo2.jpg"* y en la 7 la textura por defecto *"ejemplo.jpg"*:

Imagen 6 : Textura ejemplo2.jpg

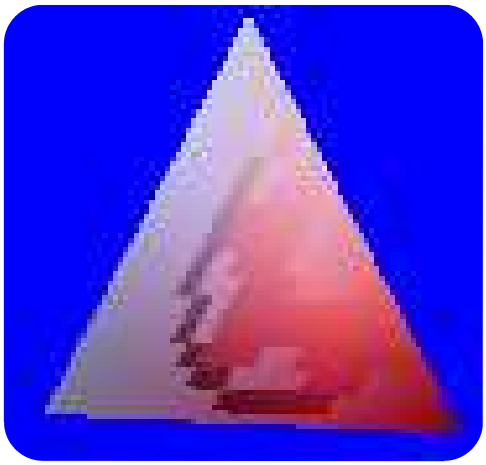

Imagen 7 : Textura ejemplo.jpg

Luces puntuales

A la hora de calcular la iluminación de Phong, se recorren todas las luces que hay en la escena, y se calcula el aporte de cada uno de ellas. En la imagen 8, podemos ver la inclusión de 2 luces puntuales, una verde y otra roja, la verde más cercana al triángulo de la izquierda y la roja al de la derecha, la escena está definida en el archivo *"test5.xml"*:

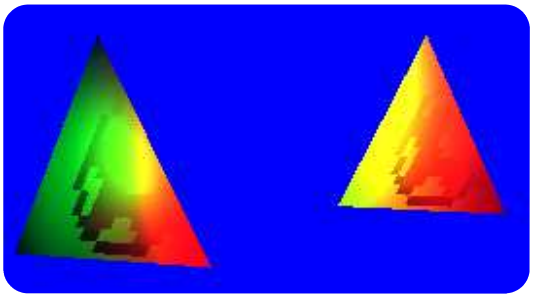

Imagen 8 : Luces verde y roja

Características opcionales

Modelos externos descritos como mallas de triángulos

Se ha implementado la funcionalidad de poder añadir modelos definidos como mallas de triángulos. Dichos modelos mantienen todas las características implementadas obligatorias. Pero nos da algún tipo de error de memoria y por lo tanto no hemos podido realizar un render final aceptable, pero existe un trabajo realizado al respecto que se puede ver en la función *"Ray::colExtModel"*.

Refracción

Se ha añadido la funcionalidad extra que permite simular la refracción, utilizando la función glm::refract, para ello se ha creado un material nuevo que se llama transparent, que define un valior elevado de esta componente. Podemos ver el resultado en la imagen 9, la escena está descrita en el archivo *"test6.xml"*:

Imagen 9 : Refracción

Material utilizado

Librerías

Para el desarrollo de la práctica hemos hecho uso de las siguientes librerías:

- GLM: Para el manejo y cálculo de la refracción.

Herramientas

Para el desarrollo de la práctica hemos hecho uso de las siguientes herramientas:

- Microsoft Visual Studio 2013.
- Git Project - https://github.com/JulioUrjc/AdvancedRendering2

Problemas encontrados

Los problemas encontrados en el desarrollo de la práctica, los cuales han limitado las características que buscábamos implementar han sido los siguientes.

- Colocación de la cámara. En primer lugar realizamos una cámara que nos realizaba una proyección errónea y tardamos un tiempo en descubrir el error que nos proporcionaba la proyección correcta.

- Colisión con objetos externos. Hemos realizado implementación con objetos externos pero nos salen algunos errores que nos han impedido terminar esta parte de la práctica, pero se ha realizado el trabajo y queríamos dejar constancia.

Bibliografía

- Physically-based Rendering (Pharr, Humphreys)
- Triangle intersection (Moller & Trumbore Method 2003)
- Sphere intersection - http://www.lighthouse3d.com/tutorials/maths/ray-sphere-intersection/
- GLM Refract Function - http://glm.g-truc.net/0.9.4/api/a00131.html#gabbb4909d3e99a7a2411cc63252afbbd8