




# RAYTRACING

Rendering Avanzado

Alejandro López Vizuite  
Claudia Ochagavías Recio

 Universidad  
Rey Juan Carlos

 ETSIJ  
Escuela Técnica Superior  
de Ingeniería Informática

## ÍNDICE

### I. Introducción

### II. Bloque 1

- A. Algoritmo de ray casting e iluminación directa
- B. Geometría: Esferas y triángulos individuales
- C. Materiales: lambertianos, con reflexión tipo Phong, y mapeado de texturas
- D. Luces puntuales
- E. Algoritmo de traza de rayos recursivo con 4 rebotes
- F. Incorporación de sombras

### III. Bloque 2

- A. Modelos externos descritos como mallas de triángulos
- B. Supermuestreo (supersampling) 4x en el plano de imagen para Antialiasing
- C. Iluminación global por Monte Carlo con 32 muestras por rebote
- D. Vídeo walk-through de una escena con camino de cámara pre-grabado

### IV. Resultados

### V. Referencias

## I. Introducción

El siguiente trabajo corresponde a la práctica 1 de la asignatura de Rendering Avanzado: Crear un trazador de rayos. En esta pequeña memoria explicaremos cada una de las funcionalidades incluidas en el código, detallaremos cómo se han implementado y mostraremos imágenes de los resultados obtenidos.

El código también ha sido comentado para que resulte más fácil de comprender y seguir.

## II. Bloque 1

- A. Algoritmo de ray casting e iluminación directa
- B. Esferas y triángulos individuales

Inicialmente se implementó el algoritmo de raycasting básico e iluminación directa. Para ello se parte de una escena inicial con una cámara y cuatro objetos: Una esfera, un triángulo y dos modelos compuestos por triángulos, un color de fondo, y una luz, incluyendo la ambiental.

Se lanzará un rayo desde la cámara para cada píxel. Con ese rayo, se pasará por toda la lista de objetos y con cada uno de ellos se verá si hay intersección. Si ocurre, se calculará el color y se visualizará en pantalla. Si no hay intersección, se visualizará el color de fondo.

**Se comienza con la esfera:** Para ello, creamos la función “intersectionSphere”. Primero calculamos el vector hacia la esfera y comprobamos que sea positivo. Obtenemos la distancia entre el punto en el que interseccionamos y el centro, y luego restamos la distancia obtenida a la distancia al centro. Si la distancia de intersección está dentro del radio, entonces obtenemos intersección, y devolveremos la longitud “t” para crear el rayo.

**Aplicado a triángulos:** A partir de la función “intersectionTriangulo”, basada en el algoritmo de Möller-Trumbore [1] comentado en tutoría. Calcularemos la intersección a partir de las coordenadas baricéntricas, que obtendremos en el vector “CoorBar”. También, y al igual que con la esfera, obtendremos la longitud “t” para crear el rayo.

**Aplicado a modelos:** Como los modelos están compuestos por triángulos, para cada uno de ellos aplicaremos la función “intersectionTriangulo”, de la misma manera que con el triángulo anterior.

### TRANSFORMACIONES

Antes de comprobar cada intersección, se han tenido en cuenta las transformaciones realizadas a cada objeto:

**Traslación:** A partir de la matriz de traslación.

**Rotación:** A partir de las matrices de rotación X, Y y Z.

**Escala:** A partir de la matriz de escala.

Todas ellas se multiplican en el orden escala - rotación – transformacion para obtener la nueva posición del modelo. Para la esfera se multiplica a “esfera-center”, y se incluye en el vector “newPos”, mientras que para los triángulos se multiplica para cada uno de los vértices. Hay que destacar que la escala de la esfera va en

función de un solo valor (radio), por lo que aumentamos el radio en el valor máximo de los valores XYZ de la matriz escala.

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

### PHONG

Comenzaremos explicando la iluminación de tipo Phong. Para obtener el color final de cada píxel, utilizamos la función “iluminacionPhong”. Las variables que le pasamos son el punto de intersección (“posPixel”), la normal, la posición de la cámara (“pos”), la posición de la luz, la intensidad de la luz, la intensidad ambiental, y las componentes del color difuso, especular y brillo, además de la variable R, cuyo valor calcularemos en la función.

Recordemos que para la iluminación de Phong en triángulos primero se calculan las coordenadas baricéntricas a partir del punto intersección, y después se calcula el color en ese punto, no se calcula el color de los vértices y luego se interpola.

Calcularemos primero los tres vectores, el de visibilidad, el de luz, y el vector reflejado. Seguidamente devolveremos el color del sombreado. Para ello sumaremos la intensidad ambiental más la componente obtenida de color difuso más la componente obtenida de color especular. Destacar que hemos multiplicado la componente ambiental por 0.3, para que a la hora de ver los resultados no influya tanto en la escena.

### MAPEADO DE TEXTURAS

Recordamos que tenemos el vector difuso: Este almacenará el color difuso del material, o bien, si hay textura, almacenará el color de la textura para cada punto. La implementación es la siguiente:

**Esfera:** Si existe textura, calculamos las coordenadas UV de la esfera y obtenemos el color de la textura. Si no hay textura, guardamos el color difuso del material.

**Triángulos:** Por cada material de cada vértice comprobamos si hay textura. Si es así, interpolamos las coordenadas UV del triángulo utilizando las coordenadas baricéntricas, y cambiamos el color difuso del material por su color de textura.

**Modelo:** Para cada triángulo del modelo se hará lo mismo que en el caso anterior.

### ¿QUÉ OBJETO ESTÁ DELANTE?

Para ver qué objeto está delante, tendremos la variable “zpos”. Para cada uno de los objetos con el que el rayo intersecciona, calcularemos la intersección. Si la variable z es menor que la variable z anterior, se sustituirá el valor. Al final, solo se pintará el color del objeto que esté delante de todos los demás.

Además, al hacer la intersección, se comprueba que el objeto esté delante de la cámara. De esta manera, no se tendrán en cuenta los objetos que se encuentren detrás (aunque fueran los que más cerca estén).

## D. Luces puntuales

Además de la luz inicial que había en la escena, hemos incluido otra luz. Se realiza un bucle, y para cada una de las luces se calcula el color mediante la iluminación de Phong, que después se sumará a los colores

obtenidos por las luces anteriores. Posteriormente explicaremos cómo afectará cada luz en el trazado de rayos recursivo.

### E. Algoritmo de traza de rayos recursivo con 4 rebotes

Después de realizar un raycasting inicial e iluminación directa mediante Phong, implementamos el algoritmo de traza de rayos recursivo con 4 rebotes.

Para cada objeto: Recordamos que en la iluminación de Phong hemos obtenido la variable R. Esta se utilizará para las llamadas recursivas. También incluiremos la variable “numRebotes”. Llamaremos a la función de Raytracing recursivamente en función de este número (en nuestro caso, 4).

```
color = color + iluminacionPhong(rayo, N, pos, luzPrueba->position, luzPrueba->color,
fondo.ambientLight, difuso, mat->specular, mat->shininess, R);

color = (color*(Vector(1.0, 1.0, 1.0) - mat->reflective) + (Raytracing(rayo, R,
la_escena, rebotes - 1)*mat->reflective));
```

En la primera llamada a la función de Raytracing, calcularemos el color del píxel, tal y como habíamos hecho hasta ahora. Ese color lo multiplicaremos por 1 menos la componente reflectiva del material (por ej., si se reflejara el rayo 0.8, solo nos quedaríamos con un 0.2 de ese primer color calculado).

A ello se le sumará el color calculado llamando de nuevo a la función. Podemos observar que los parámetros de la función están actualizados, “rayo” es el punto de intersección calculado, que funciona como nueva “pos”, y R es el vector reflejado, que funciona como el nuevo “dir”. Además el número de rebotes disminuye a 1. Así, se seguirán de nuevo todos los pasos especificados de la misma manera que se hizo anteriormente, y cuando el número de rebotes llegue a 0, se devolverá la luz ambiental para ir retornando las llamadas recursivas con ese color de base, y así obtener el color final del píxel.

```
if (rebotes == 0) {
    return fondo.ambientLight * 0.3;
}
```

### F. Incorporación de sombras

Se ha incorporado el cálculo de sombras con el trazado recursivo de rayos. Así, dentro de la función Raytracing, para cada luz, primero comprobaremos si hay sombra (función “haySombra”), que nos devolverá un true o un false. Si la hay, únicamente sumaremos al color la luz ambiental. Si no la hay, calcularemos el color por Phong y realizaremos las llamadas recursivas, como explicamos previamente. Haremos lo mismo para los tres tipos de objetos diferentes de la escena.

Función “haySombra”: Primero, habremos tenido en cuenta las transformaciones de los objetos dentro de la escena: traslación, rotación y escala. Comprobaremos si existe intersección con cada uno de los objetos. En el momento en el que se encuentre una intersección, la función devolverá un “true”. Destacar que las intersecciones solo valorarán si el objeto está entre la luz y la posición desde la que se lance el rayo. Si la distancia al objeto fuera mayor que la distancia a la luz, esta intersección no se tendrá en cuenta, ya que no habría sombra.

### III. Bloque 2

#### A. Modelos externos descritos como mallas de triángulos

Referido a los modelos externos descritos como mallas de triángulos, se han incluido los objetos de tipo “modelo”, y se ha operado con cada triángulo del mismo (descrito previamente).

#### B. Supermuestreo (supersampling) 4x en el plano de imagen para antialiasing

Si escogemos la opción “Supersampling”, calcularemos cuatro diferentes direcciones del rayo, aumentando y disminuyendo en 0.5 en los ejes X e Y, para coger así cuatro muestras de color bastante cercanas al punto XY exacto. Con estas muestras, dividiremos el color total obtenido entre el número de muestras, para devolver el color final en XY mejorando los problemas de aliasing.

Cabe destacar que, aún incluyendo un offset en el rayo, sigue produciéndose algo de ruido en la imagen, intensificándose en el renderizado por Supersampling.

#### C. Iluminación global por Monte Carlo con 32 muestras por rebote

Si escogemos la opción “Montecarlo”, se calculará la iluminación ambiental mediante el algoritmo de Monte Carlo con 32 muestras por rebote. En cada intersección del rayo, se lanzarán los 32 rayos aleatoriamente en una semi esfera, por la cual se calculará un color en función del resultado obtenido. Cada rayo aportará 1/32 de la luz ambiental en el punto calculado.

Adicionalmente, hemos creado otra opción, “MonteCarlo + Supersampling”, que ejecutará ambos modos a la vez.

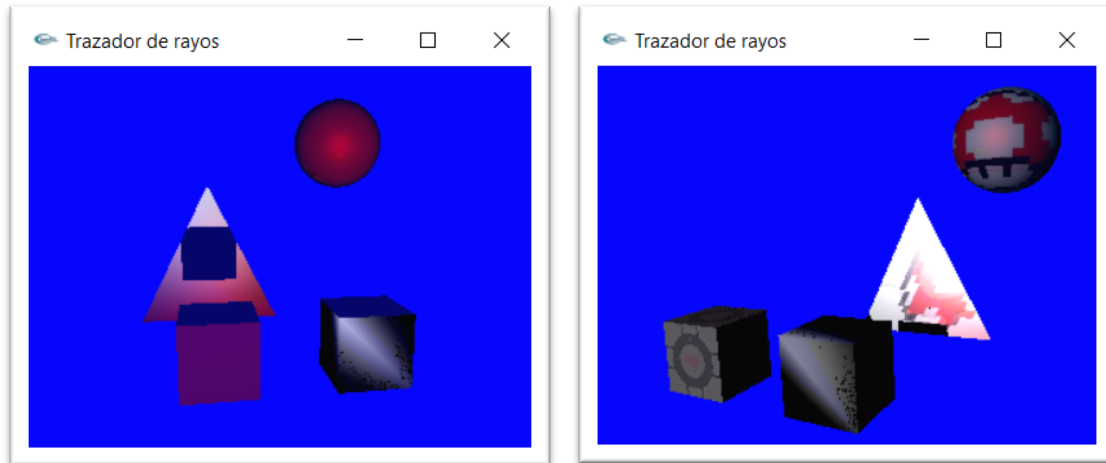
#### D. Vídeo walk-through de una escena con camino de cámara pre-grabado

Hemos realizado dos videos con unos trayectos cortos para mostrar un camino en la escena “test2”. El primero de ellos, mediante la técnica Raytracing, muestra una traslación y giro de la cámara. En cambio, en el segundo video, mediante la técnica de Montecarlo, se realiza un movimiento de zoom-in zoom-out a la escena.

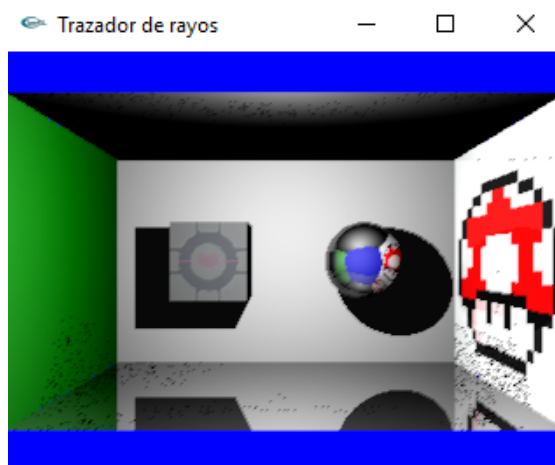


## IV. Resultados

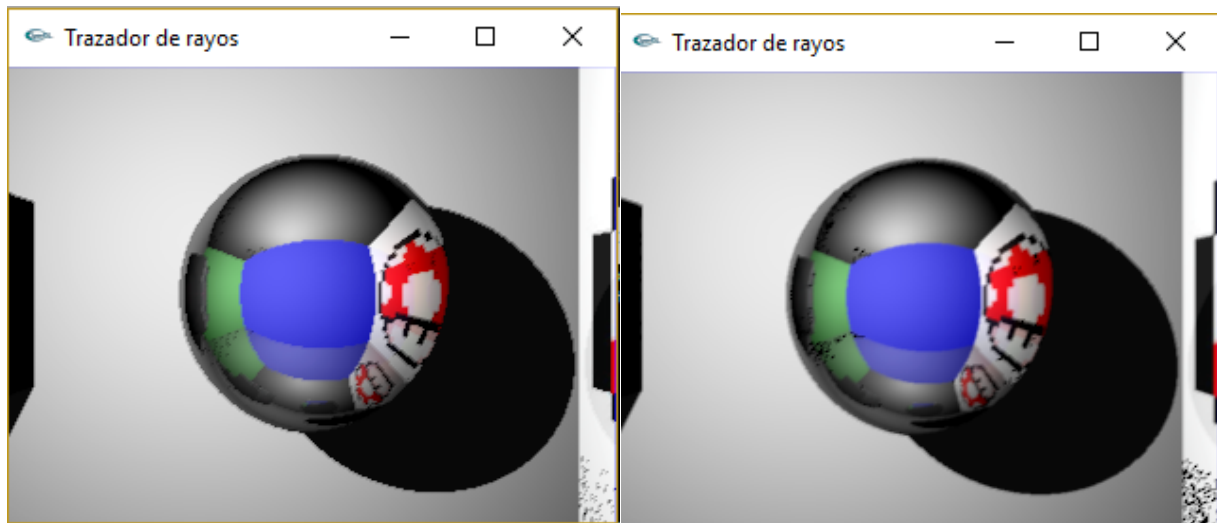
Respecto a los resultados, aquí podemos observar varias imágenes correspondientes a las distintas escenas testeadas:



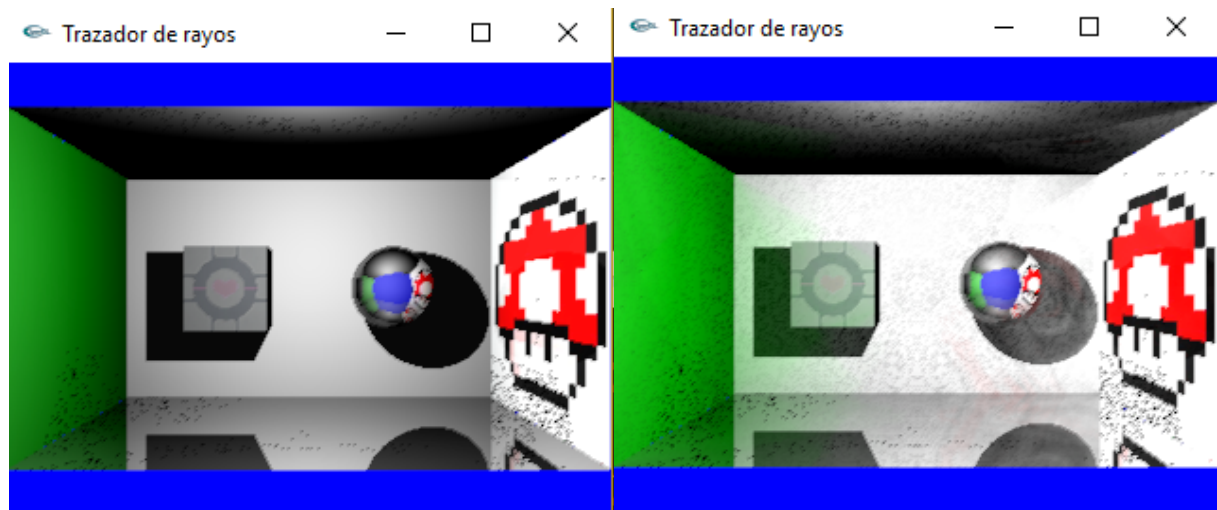
A la izquierda, escena “test”, a la derecha, “test1”. Como podemos observar, el raytracer funciona bien tanto con texturas como sin ellas.



La siguiente es la escena “test2” por raytracing. Se puede observar como las texturas y los colores aplicados a la escena se reflejan perfectamente en la esfera, así como que las sombras son correctas.



También para la escena “test2”, comprobamos la eficiencia de Supersampling, haciendo zoom en el borde de la esfera. Como se puede ver, en la izquierda se notan más los bordes que en la esfera de la derecha.



Como se puede apreciar, en la escena “test2”, la luz ambiental producida por el Monte Carlo (derecha) hace que la pared del fondo obtenga tonos verdosos de la pared cercana.





Esta es la escena “test3”, se ha creado un entorno de trabajo simple, únicamente una caja de Cornell y una esfera reflectiva, la cual sirve para comprobar el funcionamiento de traza de rayos recursiva, así como de las sombras producidas por más de una luz, produciéndose así una sombra oscura en la “fusión” de ambas sombras.

## V. Referencias

- [1] [https://en.wikipedia.org/wiki/M%C3%Beller%E2%80%93Trumbore\\_intersection\\_algorithm](https://en.wikipedia.org/wiki/M%C3%Beller%E2%80%93Trumbore_intersection_algorithm)

Otras referencias utilizadas:

<https://www.ics.uci.edu/~gopi/CS211B/RayTracing%20tutorial.pdf>