



# miniRT

Mi primer RayTracer con miniLibX

*Resumen: Este proyecto es una introducción al bonito mundo del Raytracing. Una vez completes este proyecto serás capaz de renderizar simples **CGI** (computer generated images, o imágenes generadas a ordenador) y nunca tendrás miedo de implementar fórmulas matemáticas otra vez.*

# **Índice general**

I.	Introducción	2
II.	Instrucciones generales	3
III.	Parte obligatoria - miniRT	4
IV.	Parte bonus	9
V.	Ejemplos	11

# Capítulo I

## Introducción

Cuando se trata de renderizar imágenes tridimensionales hay dos posibilidades: “Rasterización”, utilizada por prácticamente todos los motores de renderizado por su eficiencia, y “Ray Tracing”.

El método de “Ray Tracing”, desarrollado por primera vez en 1968 (pero mejorado desde entonces) es todavía a día de hoy más caro computacionalmente que la “Rasterización”. Como resultado, todavía no está adaptado a usos de tiempo real pero produce un grado de realismo varias veces superior.



Figura I.1: Las imágenes de arriba se han renderizado utilizando Ray Tracing. Impresionante, ¿verdad?

Antes de que puedas empezar a producir imágenes de tan alta calidad, debes perfeccionar lo básico: el `miniRT` es tu primer ray tracer programado en C, sencillo y humilde pero funcional.

El objetivo principal del `miniRT` es demostrarte a ti mismo que eres capaz de implementar fórmulas matemáticas o físicas sin ser matemático, implementaremos solo la parte más básica del ray tracing así que mantén la calma, respira profundamente y **don't panic!** Después de este proyecto serás capaz de mostrar imágenes bonitas para justificar el número de horas que has estado en la escuela...

# Capítulo II

## Instrucciones generales

- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un 0 si hay algún error de norma dentro.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) ni tener comportamientos indefinidos. Si esto pasa tu proyecto será considerado no funcional y recibirás un 0 durante la evaluación.
- Toda la memoria alocada en heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el subject lo requiere, deberás entregar un **Makefile** que compilará tus archivos fuente al output requerido con las flags **-Wall**, **-Werror** y **-Wextra**, por supuesto tu **Makefile** no debe hacer relink.
- Tu **Makefile** debe contener al menos las normas **\$(NAME)**, **all**, **clean**, **fclean** y **re**.
- Para entregar los bonus de tu proyecto, deberás incluir una regla **bonus** en tu **Makefile**, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos **\_bonus.{c/h}**. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la **libft**, deberás copiar su fuente y sus **Makefile** asociados en un directorio **libft** con su correspondiente **Makefile**. El **Makefile** de tu proyecto debe compilar primero la librería utilizando su **Makefile**, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo a tu repositorio **Git** asignado. Solo el trabajo de tu repositorio **Git** será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus compañeros. Si se encuentra un error durante la evaluación de Deepthought, la evaluación terminará.

# Capítulo III

## Parte obligatoria - miniRT

Nombre de programa	miniRT
Archivos a entregar	Todos tus archivos
Makefile	all, clean, fclean, re, bonus
Argumentos	una escena en formato *.rt
Funciones autorizadas	<ul style="list-style-type: none"><li>• open, close, read, write, printf, malloc, free, perror, strerror, exit</li><li>• Todas las funciones de la librería de matemáticas (-lm man man 3 math)</li><li>• Todas las funciones de la miniLibX</li></ul>
Se permite usar libft	Sí
Descripción	El objetivo de tu programa es generar imágenes utilizando el protocolo del ray tracing. Cada una de las imágenes generadas representará una escena, vista desde un ángulo y una posición específica, definida por simples objetos geométricos, y cada una con su propio sistema de iluminación.

Los principios son los siguientes:

- Deberás usar la `miniLibX`. Ya sea de la versión disponible en el sistema operativo, o de su fuente. Si eliges trabajar con fuentes, deberás aplicar las mismas reglas que para tu `libft`, tal y como están descritas en la parte de **Instrucciones generales**.
- La gestión de la ventana debe permanecer limpia: cambiar de ventana, minimizar, etc.

- Deberás tener al menos estas 3 figuras simples: **plano**, **esfera** y cilindro.
- Si aplica, todas las intersecciones posibles y el interior de los objetos deberán gestionarse correctamente.
- Tu programa debe ser capaz de escalar las propiedades únicas de los objetos: diámetro para la esfera, y ancho y alto para el cilindro.
- Tu programa debe ser capaz de aplicar traslaciones y rotaciones para los objetos, luces y cámaras (excepto de las esferas y las luces, que no se pueden rotar).
- Gestión de luz: **puntos de brillo**, sombras duras, luces de ambiente (los objetos nunca están completamente en la oscuridad). Deberás implementar luz de ambiente y luz difusa.
- Tu **miniRT** deberá ser capaz de renderizar cualquier resolución positiva (utiliza 1080p por defecto para tu evaluación).
- El programa mostrará la imagen en una ventana y respetará las siguientes normas:
  - Presionar **ESC** debe cerrar la ventana y terminar el programa limpiamente.
  - Al hacer clic en la cruz roja de la ventana, esta debe cerrar y el programa terminar limpiamente.
  - El uso de **imágenes** de la **miniLibX** se recomienda encarecidamente.
- Tu programa debe aceptar como primer argumento la descripción de una escena con extensión **.rt**.
  - Cada elemento puede estar separado por una o más líneas vacías.
  - Cada dato de cada elemento puede estar separado por uno o más espacios.
  - Cada elemento puede estar puesto en cualquier orden en el archivo.
  - Los elementos definidos por una letra mayúscula solo debe declararse una vez en la escena.

- Cada elemento tendrá como primer dato el identificador (compuesto por uno o dos caracteres), seguidos de todos los datos requeridos para cada tipo de identificador en un orden estricto:
- Luz de ambiente:

```
A 0.2 255,255,255
```

  - identificador: **A**
  - ratio de luz ambiente en el rango [0.0,1.0]: **0.2**
  - colores R,G,B en range [0-255]: **255, 255, 255**
- Cámara:

```
C -50.0,0,20      0,0,1    70
```

  - identificador: **C**
  - coordenadas x,y,z del punto de vista: **0.0,0.0,20.6**
  - vector de orientación normalizado. En el rango [-1,1] para cada eje: **0.0,0.0,1.0**
  - FOV : Campo de visión horizontal en el rango [0,180]
- Luz:

```
L  -40.0,50.0,0.0  0.6     10,0,255
```

  - identificador: **L**
  - coordenadas x,y,z del punto de luz: **0.0,0.0,20.6**
  - el ratio de brillo en el rango [0.0,1.0]: **0.6**
  - (parte bonus) colores R,G,B en el rango [0-255]: **10, 0, 255**
- Esfera:

```
sp  0.0,0.0,20.6  12.6   10,0,255
```

  - identificador: **sp**
  - coordenadas x,y,z del centro de la esfera: **0.0,0.0,20.6**
  - diámetro de la esfera: **12.6**
  - colores R,G,B en rango [0-255]: **10, 0, 255**

- o Plano:

```
pl  0.0,0.0,-10.0  0.0,1.0,0.0  0,0,225
```

- o identificador: **pl**
- o coordenadas x,y,z: **0.0,0.0,-10.0**
- o vector de dirección 3D normalizado en el rango [-1,1] para los ejes x,y,z: **0.0,0.0,1.0**
- o colores R,G,B en rango [0-255]: **0, 0, 255**

- o Cilindro:

```
cy  50.0,0.0,20.6  0.0,0.0,1.0  14.2  21.42  10,0,255
```

- o identificador: **cy**
- o coordenadas x,y,z: **50.0,0.0,20.6**
- o vector 3D normalizado de orientación en el rango [-1,1] para los ejes x,y,z: **0.0,0.0,1.0**
- o el diámetro del cilindro: **14.2**
- o la altura del cilindro: **21.42**
- o colores R,G,B en rango [0,255]: **10, 0, 255**

- Ejemplo de la parte obligatoria con una escena .rt simple:

```
A 0.2                                     255,255,255
C -50,0,20      0,0,0      70
L -40,0,30          0.7      255,255,255
pl 0,0,0          0,1,0,0    255,0,225
sp 0,0,20          20        255,0,0
cy 50.0,0.0,20.6 0,0,1.0  14.2  21.42  10,0,255
```

- Si hay algún fallo en el archivo de configuración el programa debe salir limpiamente y devolver "Error\n" seguido de un mensaje de error explícito de tu elección.
- Para la defensa, sería ideal que tuvieras un conjunto de escenas enfocadas a ver qué es funcional, para facilitar el control de los elementos a crear.

# Capítulo IV

## Parte bonus

El Ray-Tracing puede incorporar muchas más cosas como reflexión, transparencia, refracción, objetos más complejos, soft shadows, cáustica, iluminación global, bump mapping, renderizado de archivos .obj, etc.

Sin embargo, para el proyecto `miniRT`, queremos dejar las cosas sencillas para tu primer raytracer en tu camino por el CGI.

Así que aquí tienes una lista de elementos sencillos que puedes implementar, si quieres implementar elementos más grandes te recomendamos programar desde cero un nuevo ray-tracer más tarde en tu vida de desarrollador después de que este pequeño esté terminado y perfectamente funcional.



Figura IV.1: Un punto, una skybox y una brillante Tierra texturizada con bump-mapping



Los bonus serán evaluados si y solo si la parte obligatoria está PERFECTA. Por PERFECTA queremos naturalmente decir que tiene que estar completa, sin fallos, incluso en los casos de errores tontos como mal uso, etc. Significa que tu parte obligatoria debe tener TODOS los puntos, de otro modo serán completamente IGNORADOS.

Lista de bonus:

- Añadir reflexión especular para tener un modelo de reflexión de Phong completo.
- Disrupción de color: tablero de ajedrez.
- Luces multipunto y de colores.
- Otra figura de segundo grado: conos, hiperboloides, paraboloides...
- Gestionar texturas bump mapped.



Tienes permitido utilizar otras funciones para completar la parte bonus siempre y cuando su uso se justifique durante la evaluación. Se permite también modificar el formato del archivo de la escena de acuerdo a tus necesidades. Sé inteligente.

# Capítulo V

## Ejemplos

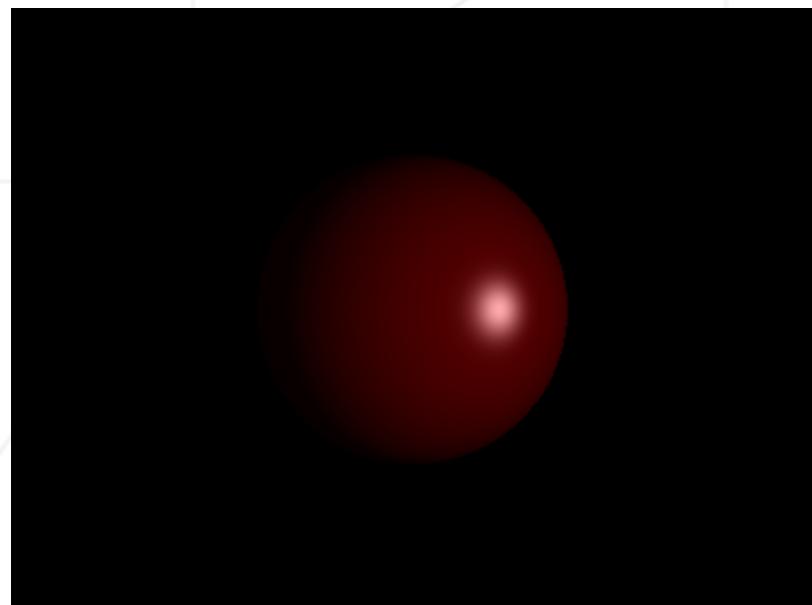


Figura V.1: Una esfera, un punto del luz, algo de brillo (opcional)



Figura V.2: Un cilindro, un punto de luz

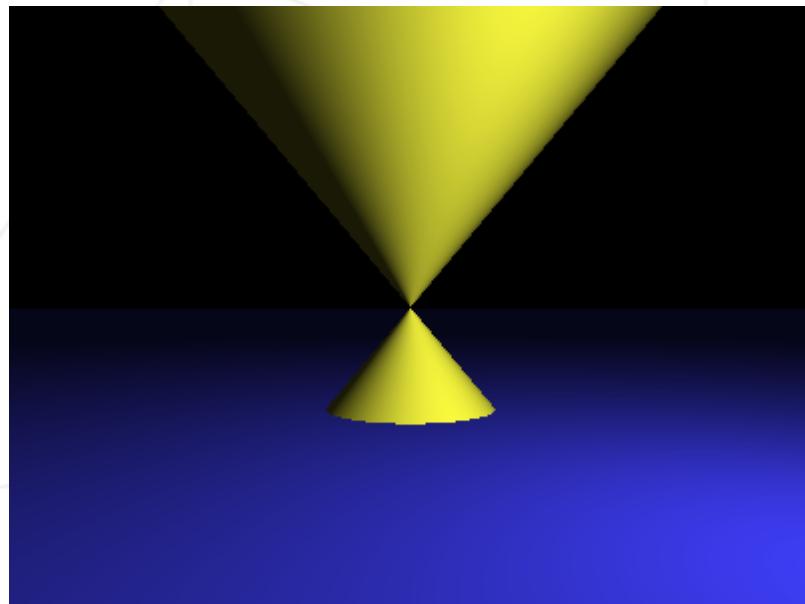


Figura V.3: Un cono (opcional), un plano, un punto de brillo

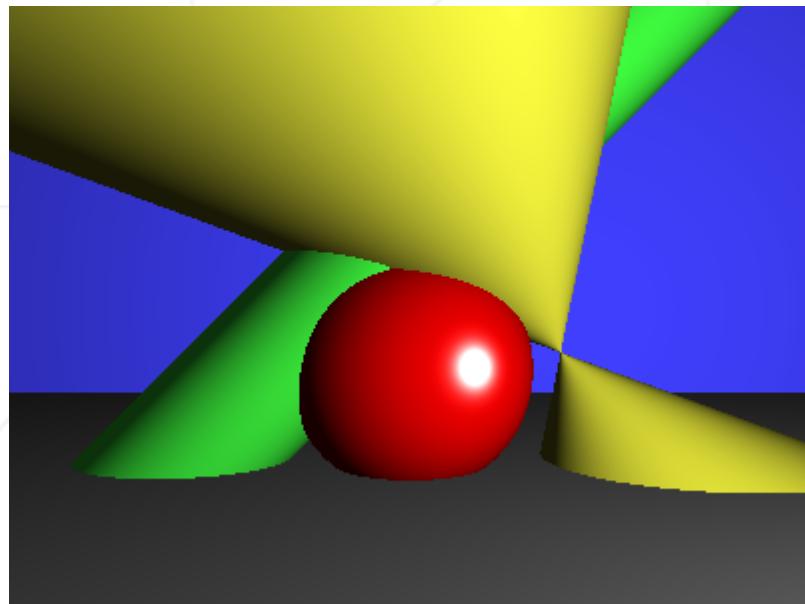


Figura V.4: Un poco de todo, incluyendo dos planos

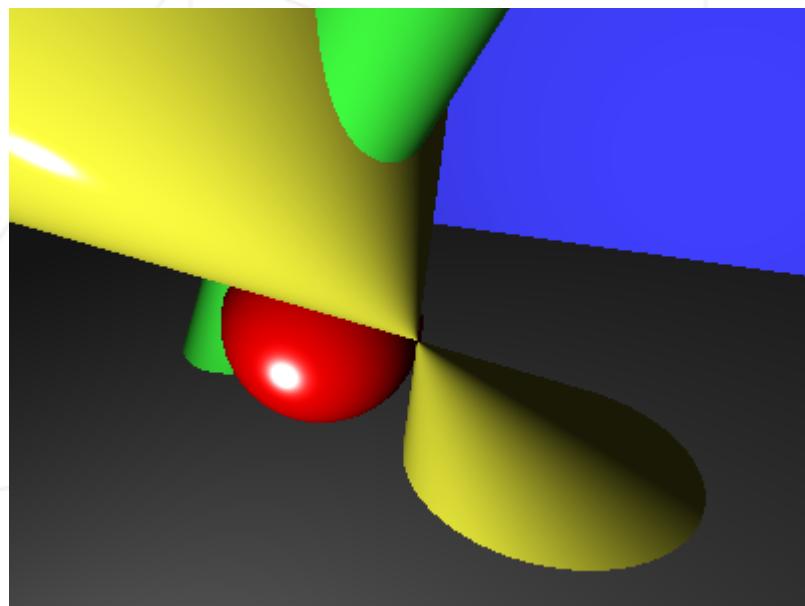


Figura V.5: La misma escena desde otra cámara

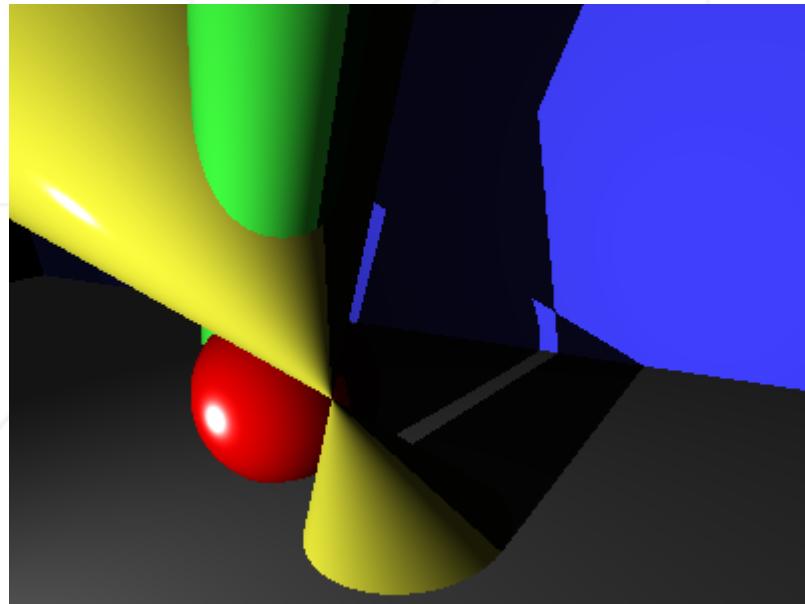


Figura V.6: Y ahora con sombras

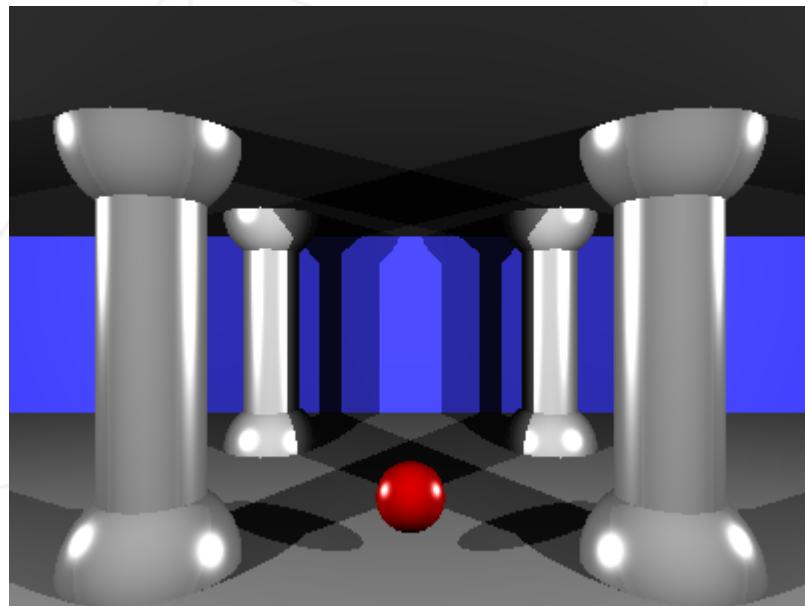


Figura V.7: Con múltiples puntos de luz

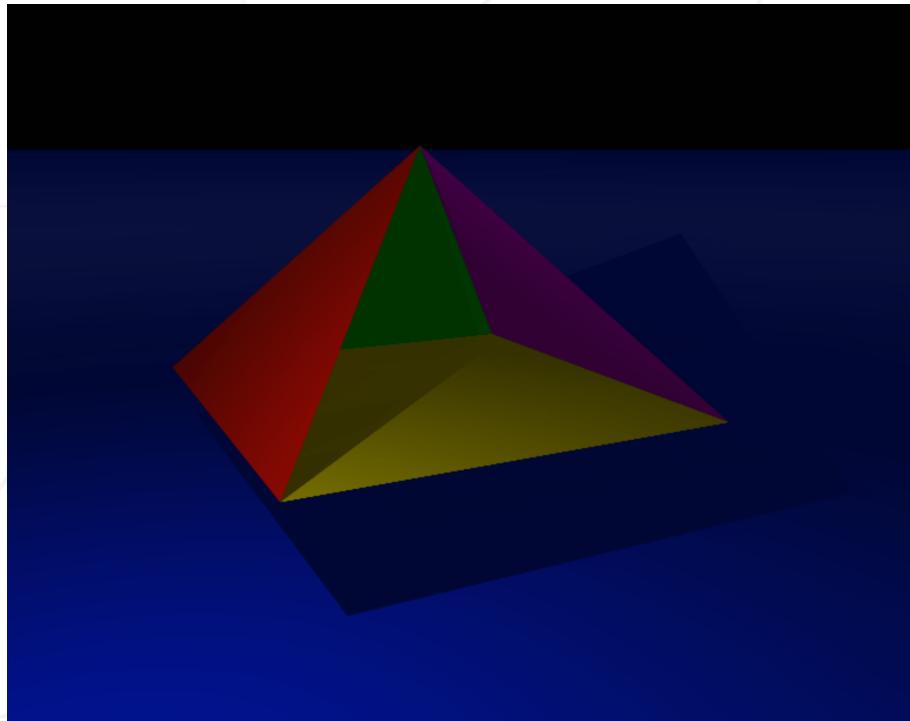


Figura V.8: Un plano, tres triángulos (opcional), un cuadrado (opcional), y un punto de brillo bajo a la izquierda

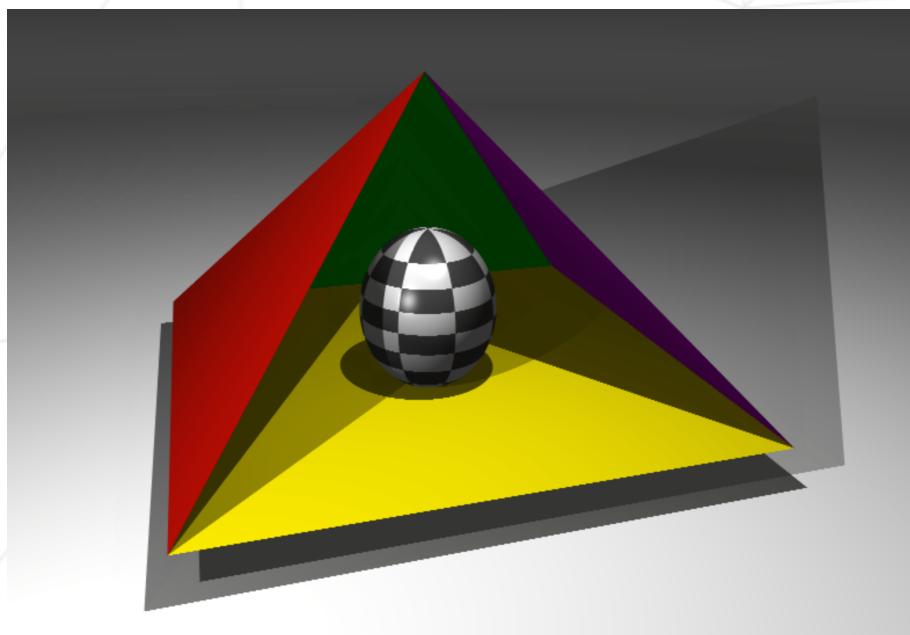


Figura V.9: Finalmente con múltiples puntos de luz y una esfera en el medio con la disrupción de color de un tablero de ajedrez brillante (opcional)