Carrera: Ingeniería en Informática

Asignatura: Computación Gráfica

INSTITUTO UNIVERSITARIO AERONÁUTICO

SEGUNDO PARCIAL

UNDEF
Universidad de la Defensa Nacional

Iluminación global mediante trazado de rayos recursivo (ray-tracing)

Alumno	Fecha: 11/11/2025
Sangüesa, Agustín	

NOTA	Diego Llorens
.....	.....

Índice

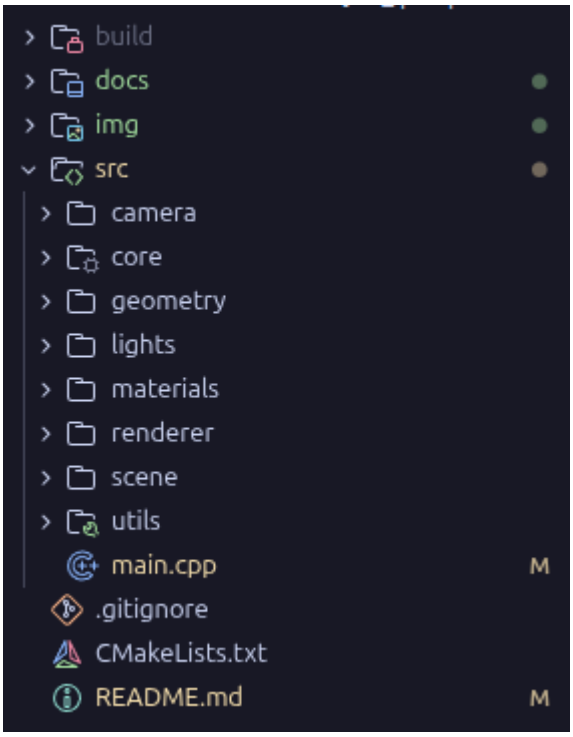
Introducción

Este trabajo presenta la implementación de un trazador de rayos (ray tracer) escrito en C++ desde cero, con el objetivo de estudiar de forma práctica cómo se forma una imagen sintética a partir de una escena 3D. El enfoque es: se construyen las piezas esenciales (cámara, geometría básica, materiales, luces y un integrador de iluminación) para lograr imágenes con iluminación local y efectos globales como sombras, reflexión y refracción.

El objetivo era crear una escena base o una escena libre, por cuestiones prácticas y de aprendizaje, se decidió implementar ambas, una realizada completamente por IA (escena base) y una realizada a mano (libre). Ambas escenas utilizan formas geométricas básicas como esferas y triángulos.

Estructura del Código

La estructura del código fue pensada de la siguiente manera (por una cuestión de costumbre personal, los nombres de directorios y las funciones estan en ingles, los comentarios en español):



Donde tenemos 4 carpetas principales y un archivo main.cpp que es nuestro punto de entrada para crear las escenas.

- **build:** Como su nombre indica, es la carpeta donde realizaremos el build/compilación de nuestra aplicación C++, una vez realizado el mismo, tendremos un archivo raytracer el cual utilizaremos para crear las escenas.
- **docs:** Contiene la consigna del trabajo en formato markdown.
- **img:** Directorio donde se guardan las imágenes generadas.
- **src:** Código fuente, dentro de él tenemos todos los directorios y archivos que hacen nuestra aplicación.

La arquitectura de la aplicación y distribución de carpetas fue pensada por mi, no se utilizó IA en esto, me base en lo que consideré más modular y prolijo.

Estructura de subdirectorios src:

En esta carpeta tenemos todos los archivos de encabezado de nuestro proyecto C++, no hay archivos cpp como tal, nuestro único archivo cpp es el main.cpp que es el punto de entrada como bien se mencionó anteriormente.

- **camera:** Aca se encuentra el "ojo" de la escena, contiene el archivo Camera.h, el trabajo que tiene es generar los rayos primarios que salen desde el POV (point of view o punto de vista en español) hacia los píxeles de la imagen. El render toma esos rayos y calcula el color de cada píxel de la escena. Esta cámara acepta diferentes parámetros como lookFrom (posición de la cámara), lookAt (a que punto mira), vup (es la orientación de la rotación), vfovDeg (es el FOV de la cámara, básicamente la apertura, muy común de configurar en juegos), aspect (relación de aspecto, básicamente ancho y altura).
  Breve explicación de cómo funciona: Construye un rectángulo en base a los parámetros que le dimos (fov, ancho y alto) y para un pixel con coordenadas normalizadas, crea un rayo que va desde lookFrom hacia el punto correspondiente del rectángulo.
- **core:** Contiene los archivos Ray.h y Vec3.h. El primero es un rayo definido por un origen y una dirección. Simplemente tiene un constructor y la función at(t) la cual devuelve el punto en el espacio al avanzar una distancia escalar *t* a lo largo del rayo. La cámara lo usa para crear un Ray como tal por pixel, y luego con componentes que tenemos en la geometría determinamos si golpea en una superficie. El segundo (Vec3.h) es un vector de 3 componentes (x,y,z) el cual lo usamos para todo, sean posiciones, direcciones y colores RGB. Tiene operaciones básicas como suma, resta, multiplicación escalar, normalización, producto punto y producto cruz. También tiene utilidades de iluminación como el reflect para los reflejos, refract para refraccion y schlickFresnel para mezclar reflexion/transmision en materiales transparentes. La idea de crear este Vec3.h era el poder modularizar/reutilizarlo tanto para colores/direcciones.
- **geometry:** Como bien dice su nombre, es la geometría de nuestro proyecto, consta de 4 archivos. Hittable.h, Plane.h, Sphere.h y Triangle.h. Hittable es una interfaz base que se utiliza para determinar los impactos. Plane.h es un plano infinito. Sphere.h es

una intersección por ecuación cuadrática, está planteada siguiendo la teoría vista en clase. Triangle.h es una intersección Möller–Trumbore también planteada en clase. ¿Cómo se integra esto con lo demás? La escena mantiene un *std::vector<std::shared_ptr<Hittable>> objects* y busca la intersección más cercana llamando al metodo hit de Hittable.h en cada objeto. Además, aquí también determinamos las sombras y el tipo de material que incidimos. Tenemos un Integrator el cual usa el método HitRecord para leer la normal y el material, para entonces calcular el color (difuso/especular, sombras, reflexión, refracción). Para las sombras, si choca contra un objeto el rayo, se considerará la luz oculta y seteamos castsShadow=true.

- **lights:** Contiene un archivo PointLight.h el cual es una luz puntual. Tiene como parámetros la posición, el color y la intensidad.
- **materials:** Contiene un archivo Material.h que utilizamos para definir cómo se ve una superficie cuando recibe luz. Por lo que, tiene muchos parámetros configurables como Ka, Kd, Ks, shininess, reflectivity, transparency, ios, fuzz, emissive, castsShadow. Además tiene funciones helper para determinar si la superficie es reflectiva o no. Además tenemos materiales concretos como Dielectric, Metal y Lambertian que se pedía en la consigna.
- **renderer:** El renderer tiene 2 archivos, Integrator.h (mencionado anteriormente) y Renderer.h. La función es convertir la escena en una imagen, toma la cámara y genera rayos por pixel y calcula el color final mezclando luz directa (Phong) y efectos globales (reflexion y refraccion). El Renderer.h recorre la imagen pixel por pixel y va obteniendo el valor de cada rayo. Tiene varios parámetros configurables como el ancho y el alto, el spp que es el samples per pixel, para suavizar bordes, mientras mas alto mas tiempo demora el renderizado, y el maxDepth que es el límite de rebotes para los rayos recursivos. El Integrator.h calcula el color de un rayo. Si no pega en nada, devuelve el color de fondo de la escena, si pega en un objeto calcula el color final en base al objeto y sus propiedades.
- **scene:** El scene contiene un Scene.h que básicamente es el organizador de la escena. Este guarda todos los objetos, luces y se comunica con el renderer para determinar por ejemplo, donde choca un rayo, si una luz está tapada, etc.
- **utils:** Contiene archivos auxiliares, entre ellos un archivo para generar la imagen PPM, otro para generar la imagen en PNG.

Por último, nuestro punto de entrada es el main.cpp, quien hace la magia utilizando todos estos headers que acabamos de mencionar.
Como bien se mencionó anteriormente, la función es orquestar y generar la escena en base a diferentes parámetros que podemos pasarle.
En principio define argumentos/parámetros, los cuales son: spp, maxDepth, scene (final/base, final es la escena libre y base la de 3 esferas), camera, out, height, width.
El flujo de trabajo es el siguiente:

1) Parseo de argumentos.
2) Construcción de la escena (base o final)
3) Render de la imagen.

4) Escritura a archivo (PPM/PNG)
5) Output en consola con el resultado final (exito/error)

Flujo técnico paso a paso:

1) Parseo CLI

- Lee --width, --height, --spp, --max-depth, --scene, --camera, --out.
- Valores por defecto: scene=final, camera=frontal.

2) Construcción de la escena

- Si –scene base:
  - Planos simples.
  - Tres esferas: difusa, metálica y dieléctrica.
  - Dos luces puntuales.
  - Cámara pinhole según –camera
- Si –scene final:
  - Habitación con planos (piso, techo y paredes)
  - Dos luces puntuales, en el techo y en la pared izquierda.
  - Espejo armado con triángulos y un marco alrededor.
  - Distintos POV de cámara con el parámetro –camera.

3) Configuración de la cámara:
- Frontal: Vista por default de la escena, desde el frente.
- Superior: Vista elevada con leve inclinación desde arriba de la escena.
- Lateral: Vista desde la izquierda de la escena.

4) Render:
- Crea Renderer(width, height, spp, maxDepth)
- Recorre cada píxel y por cada uno genera 1 o más rayos
- Usa el integrador para luz directa, rayos recursivos y colores.
5) Escritura de imagen
- Dependiendo si se especifico ppm o png utiliza una función auxiliar distinta. Si es PNG, primero genera un PPM temporal y lo convierte a PNG con pnmtopng y convert. Si es PPM lo escribe directo.
- Aplica la corrección gamma al exportar.

Decisiones de diseño

Desde un principio mi idea era separar y modularizar lo más posible el código, para facilitar su lectura y que sea adaptable a la escena que queramos generar. Este tipo de

proyectos tienen mucha matemática/geometría/física por detrás y a veces puede ser complicado comprenderlo, por lo que separar todo en funciones claras creo que fue un acierto.

Como bien se explicó antes, cada carpeta/archivo cumple una función específica, no hay código repetido y esta lo mas simplificado posible. El enfoque estaba 100% en que cada archivo tenga sus responsabilidades claras.

En cuanto a la forma de programar y la lógica, la idea era guiarse con los apuntes vistos en clases por lo que se utilizó esta teoría para realizar todo.

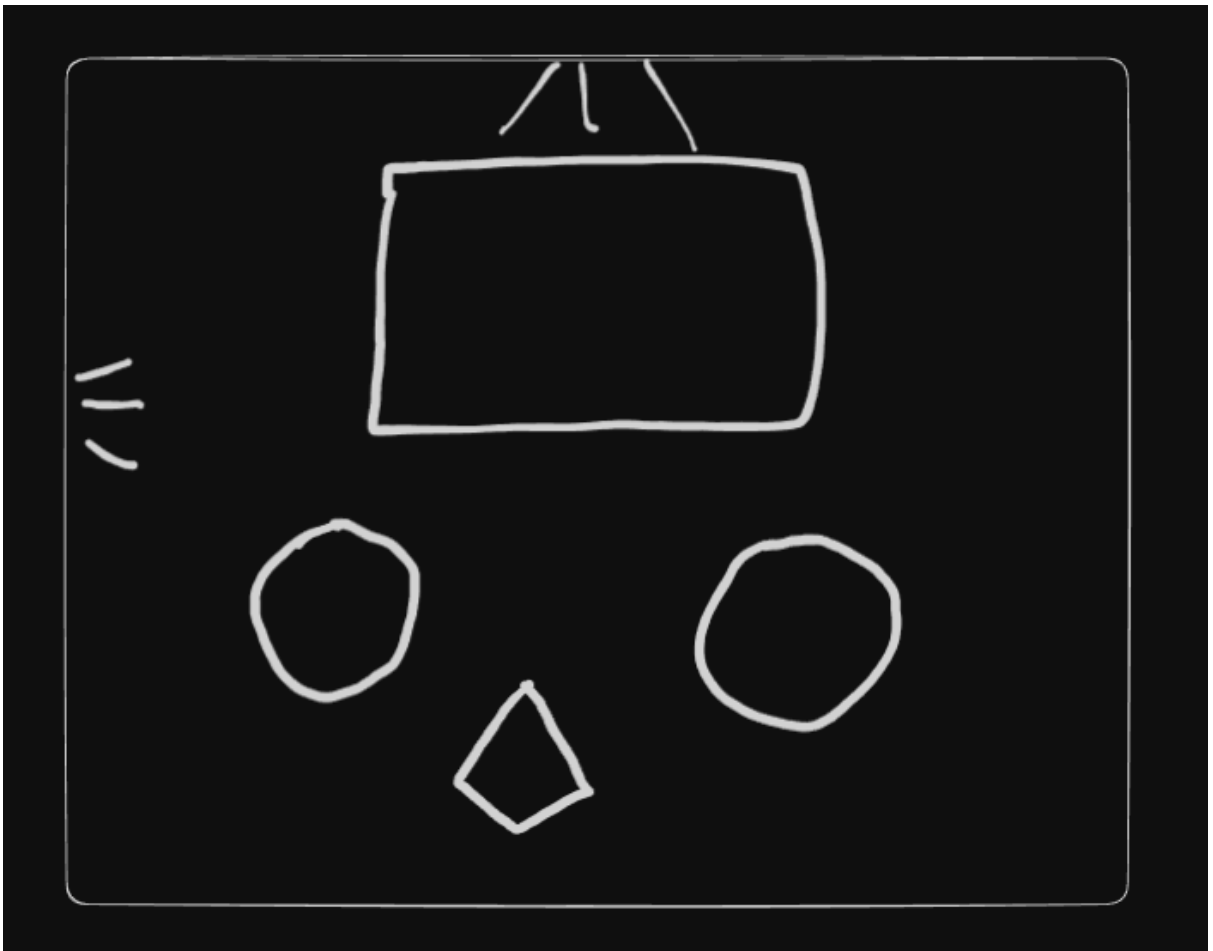
Problemas encontrados

En lo personal se dificultó el encontrar una posición de cámara adecuada, al principio no lograba encontrar bien la posición + fov adecuado que quería, por lo que ahí es donde la IA me ayudó bastante para posicionar las cosas.

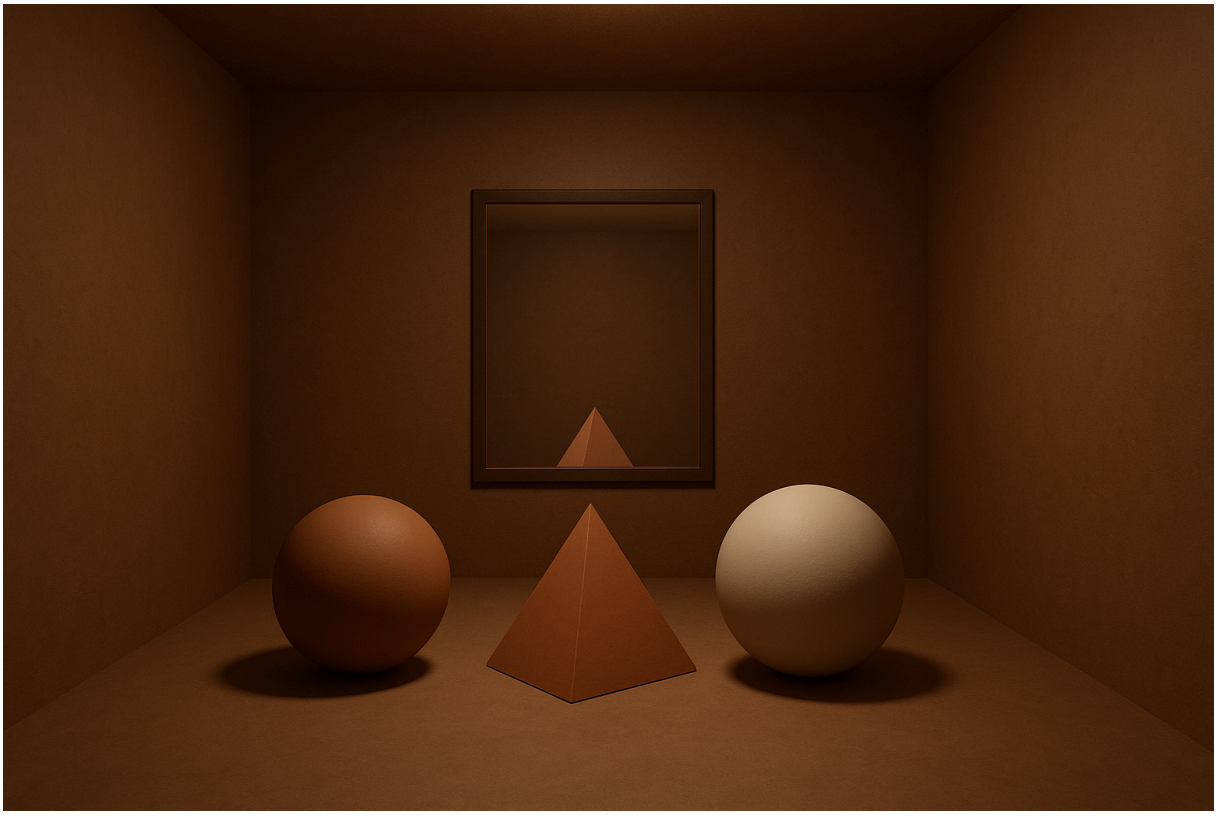
Otro factor que es importante mencionar es que, no pude asistir a todas las clases ya que el trabajo no me lo permitió, por lo que tuve que además de revisar los apuntes de clase, buscar videos en youtube y ayudarme con IA para entender mejor todos los conceptos.

Croquis o dibujos

Mi capacidad artística es muy limitada y no se dibujar, por lo que solo plantee una escena en mi cabeza y la dibuje en 2D. Luego al modelo GPT-5 le promptee mi escena un poco más detallada por lo que el flujo de planeamiento de la escena fue el siguiente:



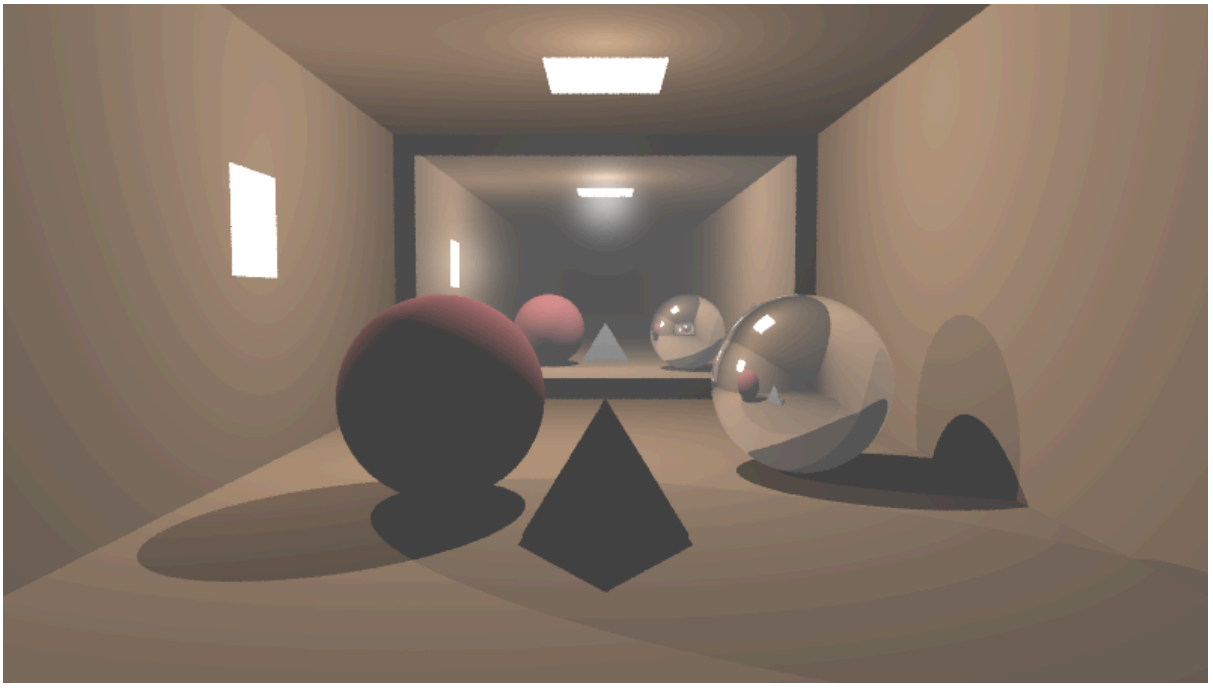
A partir de ese pésimo dibujo, la IA hizo su magia y obtuve esta escena, mucho más cercana a lo que yo tenía en mi cabeza:



Resultados Obtenidos

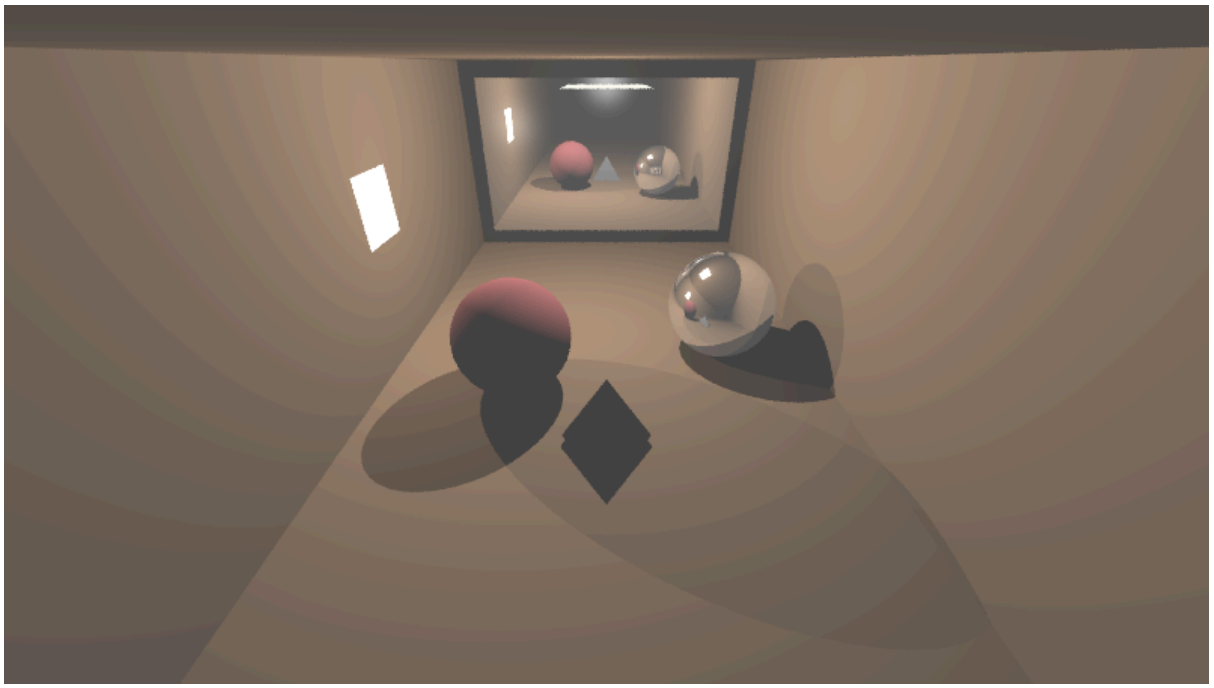
Escena Libre

Frontal

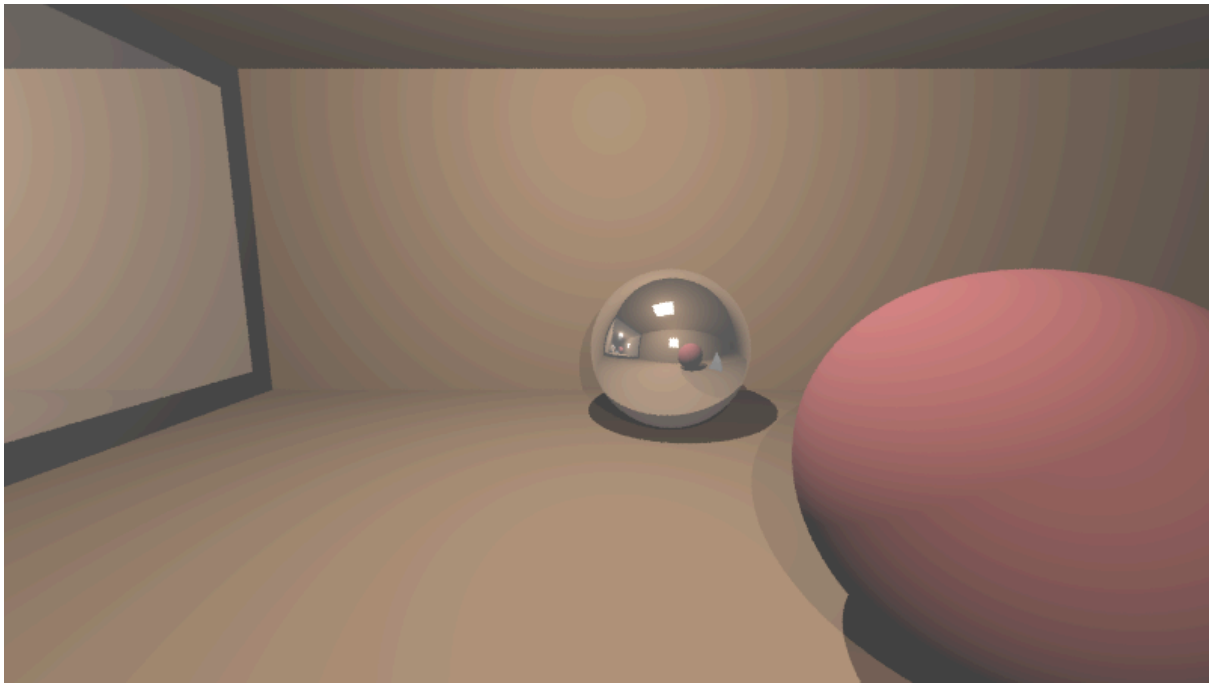


Tenemos un POV frontal donde vemos 1 esfera sólida roja, un tetraedro en el centro y una esfera reflectante a la derecha. En el fondo un espejo. También tenemos 2 luces puntuales, una en el techo y la otra en la pared izquierda. Como se puede ver, las sombras generadas tienen coherencia, si miramos la esfera de la izquierda, genera 2 sombras, una gracias a la luz superior y otra a la luz de la izquierda. Además, el tetraedro está detrás de la esfera sólida y se lo ve oscuro ya que tapa toda la luz proveniente de la izquierda.
Por otro lado la esfera reflectante y el espejo vemos que también son coherentes, reflectando toda la habitación.

Superior



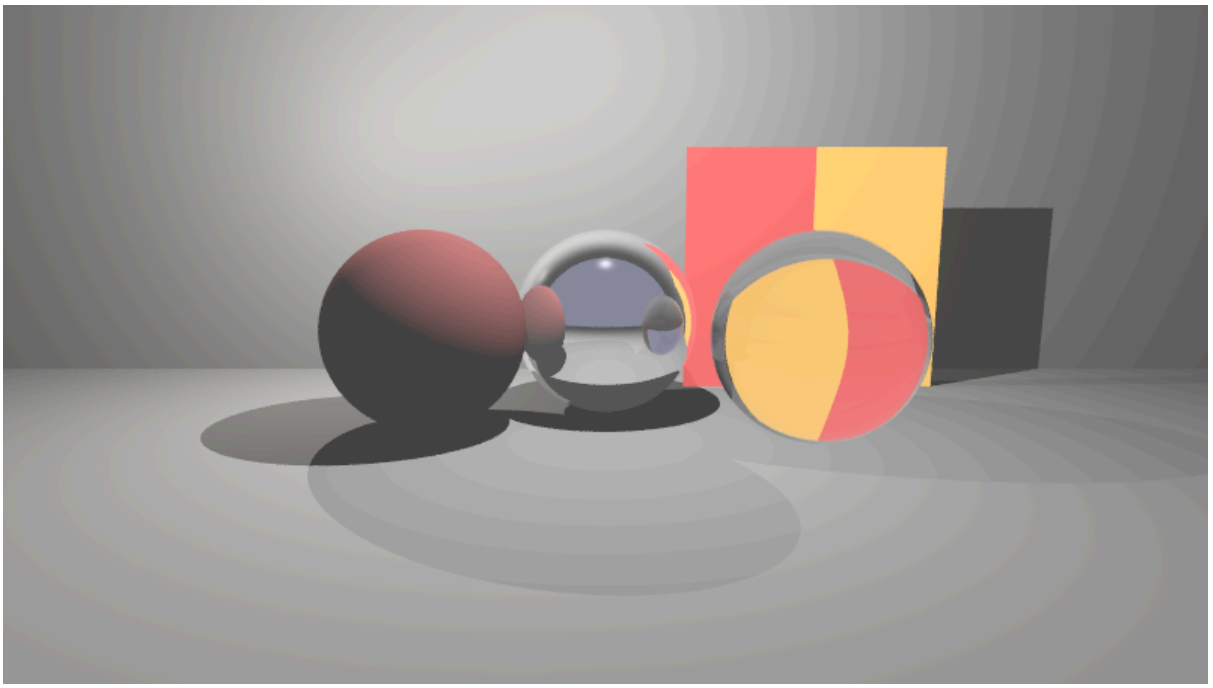
Lateral



Aca podemos ver desde otro POV como cambia la reflexión y sigue siendo coherente con el entorno.

Escena Base (Generada 100% por la IA)

Frontal



Reflexión Personal

En lo personal es un trabajo que me gustó, me hubiera gustado poder asistir a todas las clases para no tener que ponerme a investigar la mayoría de las cosas por mi cuenta, pero mi trabajo no me lo permitió.

En cuanto al tecnicismo, yo programo principalmente en Python y Ruby, los cuales son lenguajes muy similares y no tienen una sintaxis tan estricta como lo es C++. Programe en C los primeros años de la carrera y hacía mucho tiempo no programaba en este lenguaje asique tuve que ayudarme con herramientas de IA para plasmar mis ideas y pensamiento en código. Puntualmente la IA que suelo usar es Claude o Codex. Para programar backend considero que hoy en día GPT 5 es muy útil, Sonnet 4.5 me gusta más para lo que es UI. Para este proyecto utilice solamente GPT 5 con Codex CLI.

Como mencione a lo largo del proyecto, la estructura/arquitectura del proyecto la plantee yo, tengo experiencia trabajando en proyectos grandes y tengo pulido el tema de modularizar cosas, y considero que la estructura que plantee acá está bastante robusta, quizás el main.cpp se podría separar más para no tener todas las escenas juntas, pero por simplicidad es el único que lo deje asi nomas.

Todas las funciones y lógica, las plantee por mi cuenta, siguiendo los ejemplos de las clases y los pseudocódigos que fueron proporcionados en clase (para los rayos recursivos y demás). Donde más utilice IA fue para las posiciones de los objetos, cámara, etc ya que se me

dificulta bastante ordenar y posicionar las cosas. También para la sintaxis ya que al no programar en C hace mucho tiempo, había cosas que realmente no recordaba.

Para testear si la reflexión estaba funcionando correctamente, lo que hice fue utilizar la esfera metálica y con diferentes posiciones de cámara ver cómo se desplazaban los reflejos. Después de probarlo desde varios ángulos todo parecía correcto ya que los desplazamientos tienen bastante sentido. No hice un testeo “teórico”como tal, fue 100% a ojo.

En cuanto a los conceptos y su dificultad para aplicarlos, estuvo bastante complicado como mencioné anteriormente porque no asistí a clases, entonces sin la explicación en clase se me hizo un poco complicado, pero entre youtube y IA logre entender todo. Lo que mas tiempo me llevo puntualmente fue la refracción, me dio algunos problemas, el resultado final no es el que mas me gusto pero fue lo que pude lograr (se ve en la imagen BASE más arriba). Básicamente el vidrio no mostraba bien lo de atrás, tuve varios problemas, primero los rayos que entraban a la esfera, pasaban la primera capa pero en la segunda no, debugueando logre solucionarlo. Lo que pasaba era que el rayo refractado estaba mal posicionado, se lanzaba del lado incorrecto de la superficie, entonces pasaba la primera capa de la esfera y se intersectaba con la propia esfera, haciendo que se vea opaca y no a través de ella. Después de iterar y debuguear y consultarle a diferentes IA logre solucionarlo, generando ese efecto de deformación pero viendo a través de la esfera. Considero también que me costó entender Fresnel, por eso también tuve problemas con esto.

La forma de corroborar la refracción en este caso fue poner una placa de 2 colores detrás de la esfera y corroborar que esta se refracta, moviéndola para diferentes direcciones.

Por último, como también expliqué más arriba, la consigna pedía o una escena base o una libre, hice las 2. Con el detalle de que, la escena base deje que la haga completamente la IA a modo de complemento, y la libre la hice yo. No se si cuenta como usar IA el tab del ide, autocompletado de nombres o variables, creo que hoy en día es lo más normal del mundo y ahorra muchísimo tiempo a la hora de programar, no lo considero como “usar IA”.

En conjunto con este informe se entregará un link al repositorio, el cual tiene un README.md con explicaciones de como compilar y correr este proyecto.