Pràctica Ray Tracing

Laura Galera Alfaro, u1959173

23 de gener de 2022

1 Objectiu

Aquesta pràctica pretén implementar l'algoritme de ray tracing que consisteix a traçar multitud de rajos des de diferents fonts de llums, calculant i analitzant els rebots dels rajos sobre diversos objectes. D'aquesta manera s'obté una escena més realista si es compara amb la pràctica passada. Ara existeixen ombres, reflexos i refraccions.

2 Escena

L'escena està composta per tres tipus de figures: triangles, esferes i plans. He considerat que el millor és mantenir una escena simple per entendre els reflexos i les refraccions. D'aquesta manera, a l'objecte `Scene` hi ha una llista de figures que constitueixen l'escena, en total 2 triangles, 3 plans i 5 esferes. Encara que cada figura geomètrica té atributs diferents, cadascuna té un tipus de material declarat al fitxer `materials.js` i que he recuperat de la pràctica anterior. No obstant això, els materials tenen dos nous coeficients. Per una banda, el coeficient de reflexió del material i per una altra l'índex de refracció.

Altres elements de l'escena són les llums, que en aquest cas n'hi ha dues. Per les llums he seguit l'esquema ja vist a la pràctica 4, on cada llum té 3 colors: ambient, especular i difusa. També cada llum té com a atribut la posició i com a novetat el booleà per indicar si està encesa o no. Tots els valors es poden modificar a partir de la funció interactiva que s'ofereix a l'HTML.

Finalment, a destacar, la càmera. A raó de guardar un conjunt de vistes, fora de l'escena es defineix una llista d'objectes càmera. Amb aquesta llista de càmeres, dins de `Scene`, es crea un objecte `careTaker`, el mateix del patró *memento*. D'aquesta manera es gestiona la lògica de canviar de càmera quan se selecciona una de les guardades. L'escena també guarda la càmera actual, que per defecte és la primera de la llista. Així, doncs, quan se selecciona una càmera a l'HTML, es recupera de l'objecte `careTaker` i s'assigna a la càmera de l'escena.

Respecte a la càmera, aquesta classe és la mateixa de la pràctica passada, les novetats que ofereix són que cada càmera té un nom i un *field of view*. Ara bé, algunes funcions de rotació i translació s'han modificat per poder treballar amb les llibreries que hi ha a l'exemple, la resta és igual.

3 Algoritme ray tracing

L'algoritme de ray tracing, el qual es pot veure a la funció `rayTracing(Scene, Screen)`, es podria dividir en 2 funcionalitats. Inicialment, hi ha la part de computar els rajos per cada píxel de la pantalla. Això es pot veure a la funció `computeRay(incX,incY,P0,Cam,x,y)` que retorna la direcció del raig, mentre que l'origen del raig és la posició de la càmera. Aquesta informació es guarda en una tupla `ray` i és el que es passa a la funció `intersectScene(Scene, ray, depth, maxDepth)` per calcular el color dels objectes amb els quals intersecciona el raig, i que forma part de la segona part. Per la pràctica, la primera part ja venia feta i s'esperava programar la segona.

3.1 Intersecció escena

La funció `intersectScene(Scene, ray, depth, maxDepth)` és una funció recursiva que retorna el color del píxel. El paràmetre `depth` serveix per aturar la recursió i *maxDepth* és per defecte 2, però a l'HTML es permet modificar.

L'estructura d'aquesta funció es basa a trobar l'objecte més proper amb el que intersecciona el raig, d'aquí la funció `computeHit(Scene, ray)`. En cas que no interseccioni amb cap objecte, el color del píxel és el del fons, ja que el raig va a l'infinit. Altrament, quan

el raig intersecciona amb un objecte, s'ha de retornar el color de l'objecte i va en funció de les llums i també de les propietats del material.

En primer lloc, s'utilitza la tècnica de *phong* ja vista amb anterioritat, però sobre la que es comenta algun aspecte. Com que hi ha múltiples llums, es calcula *phong* per cada llum i el color acaba essent la suma de totes les llums. Un cop fet això, ve la part complicada que són les reflexions i refraccions, bàsicament perquè depenen dels rebots. Per aquesta raó, primer es mira si cal continuar rebotant abans de calcular-les. En cas que sí, es comprova si l'objecte és refractant o no. He preferit no barrejar les dues coses i fer que els materials siguin refractants o reflexants, però no ambdós perquè costava entendre com es veien, però es podria fer igual. En cas que sigui refractant, es crea un nou raig amb origen la intersecció i una nova direcció i es propaga la crida recursiva. Quan es torna de la crida, el color se suma al de *phong* multiplicat per l'índex de refracció. El mateix passa per la reflexió, això no obstant, com és d'esperar, el raig té una direcció diferent, a més de multiplicar pel coeficient de reflexió.

A continuació es parla amb més precisió dels detalls importants que s'acaben d'introduir:

3.1.0.1 Intersecció objecte

La funció `computeHit(Scene, ray)` retorna una variable `hit` amb un atribut `isHit` per indicar quan hi ha intersecció o no. Cal mirar per tots els objectes de l'escena, si el raig intersecciona amb algun i guardar-se les dades del "cop" més proper. Aquestes dades són l'objecte, el punt d'intersecció, la distància i la normal de l'objecte sobre el punt d'intersecció. Com que el càlcul de la intersecció és diferent per cada tipus d'objecte, hi ha tres funcions que retornen la distància del raig al punt d'intersecció.

Les funcions són `intersectPlane(plane, ray)`, `intersectSphere(sphere, ray)` i `intersectTriangle(triangle, ray)`. Els algoritmes d'interseccions els he buscat a internet. En el cas del pla, la intersecció és relativament fàcil. Llavors, pel triangle, s'utilitza la funció del pla, però controlant que el punt estigui dins del triangle, on faig servir dues funcions més que hi ha als apunts. Pel cas de l'esfera, he seguit l'explicació del tutorial Ray Tracing in One Weekend . En tots els casos deixo un marge de 0.003 respecte

a la distància, ja que si no es produeixen errors d'aproximació que poden desembocar en figures amb soroll o interseccions errònies.

Una vegada es coneix la distància, calcular el punt d'intersecció és tan simple com sumar a l'origen del raig, la direcció del raig però amb longitud la distància trobada. A l'hora de calcular la normal, aquesta depèn de la superfície i per aquesta raó hi ha 3 funcions més.

3.1.0.2 Phong

Per *phong* vaig tenir algun problema perquè no calculava bé el vector R i hi havia figures sense la component especular. Finalment, vaig trobar un tutorial força útil amb shaders, però que vaig poder implementar sense dificultat.

La diferència respecte a la pràctica anterior són les ombres i és que quan entre el punt d'intersecció i la llum hi ha algun objecte, el color ha de tenir només la component ambient. En canvi, quan no hi ha intersecció, és a dir, la llum il·lumina directament a l'objecte, llavors es calcula amb la component especular i difusa. La funció `inShadow(interPoint, lightPoint, Scene)` retorna cert quan està ocult, fals altrament.

3.1.0.3 Reflexió

Trobar la direcció del raig reflectit es basa a aplicar la fórmula

$$\vec{d} - 2\vec{n}(\vec{d} \cdot \vec{n})$$

que és exactament el que fa la funció computeReflectionDirection(ray, hit).

3.1.0.4 Refracció

En el cas de la refracció, per calcular la direcció del raig he seguit els apunts de *Ray Tracing in One Weekend*. La funció `function computeRefractionDirection(ray, hit)` aplica la llei de Snell a partir de l'índex de refracció "eta" i "eta prima". "eta" he fet que sigui 1 i "eta prima" depèn del material.

4 Resultats

A continuació es presenta una galeria de captures modificant paràmetres de la llum, la profunditat màxima de rebot i canviant de vista.

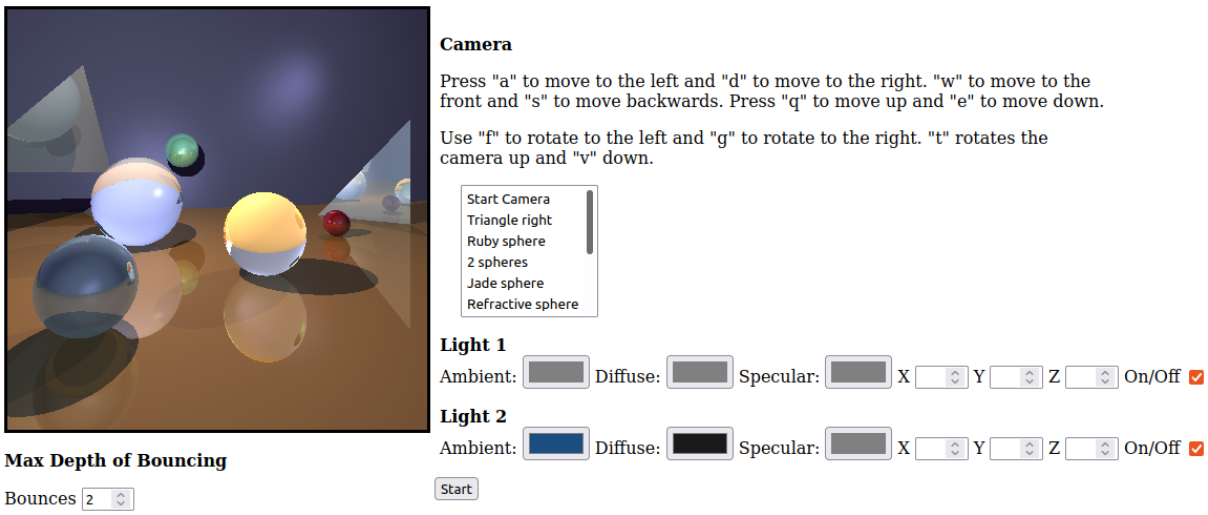

Figura 1: Pàgina inicial

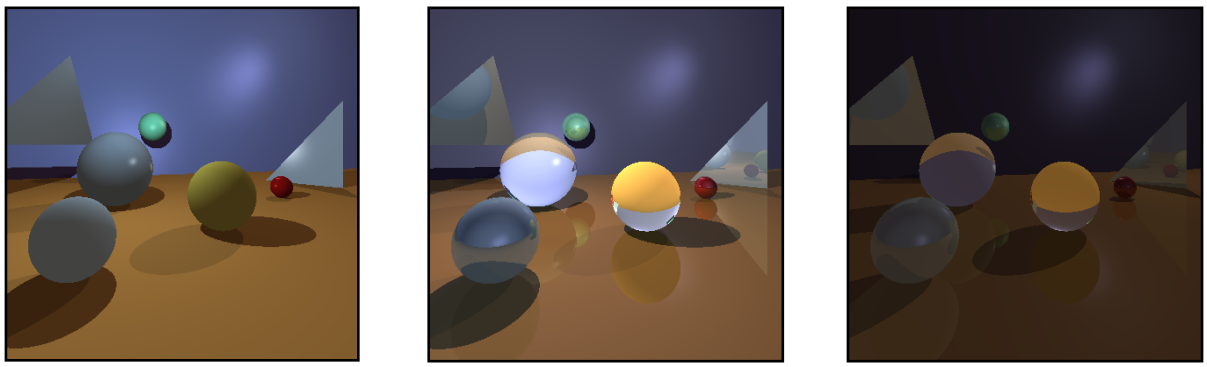

Figura 2: 0 rebots (esquerra). 1 rebot (centre). Llum 1 apagada (dreta)

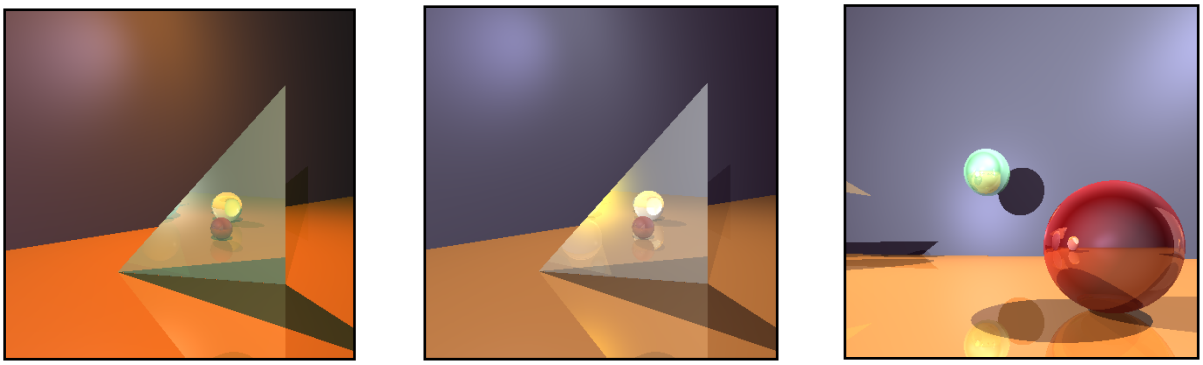

Figura 3: vista "Triangle right" amb colors diferents (esquerra i centre). Vista "Ruby sphere" amb colors diferents (dreta)

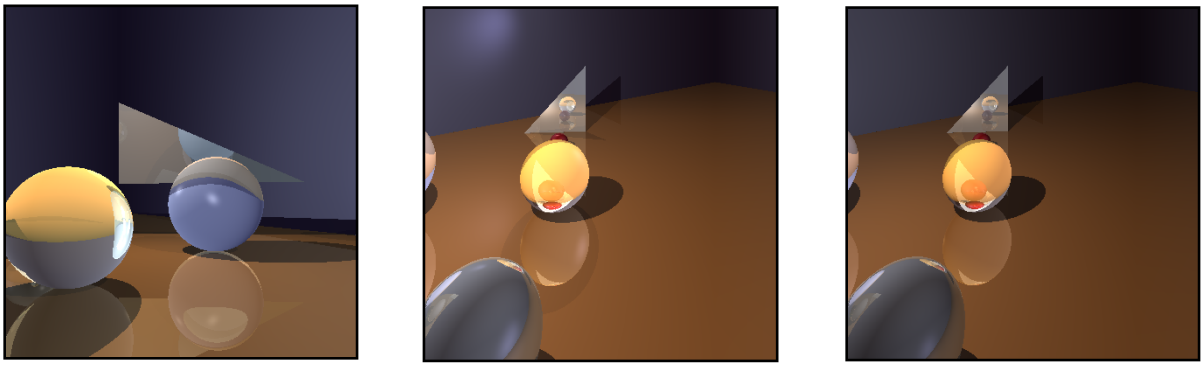

Figura 4: Vista "2 spheres" amb colors diferents (esquerra). Vista "Refractive sphere" amb 9 rebots (centre). Vista "Refractive sphere" amb 9 rebots i llum 2 apagada (dreta)

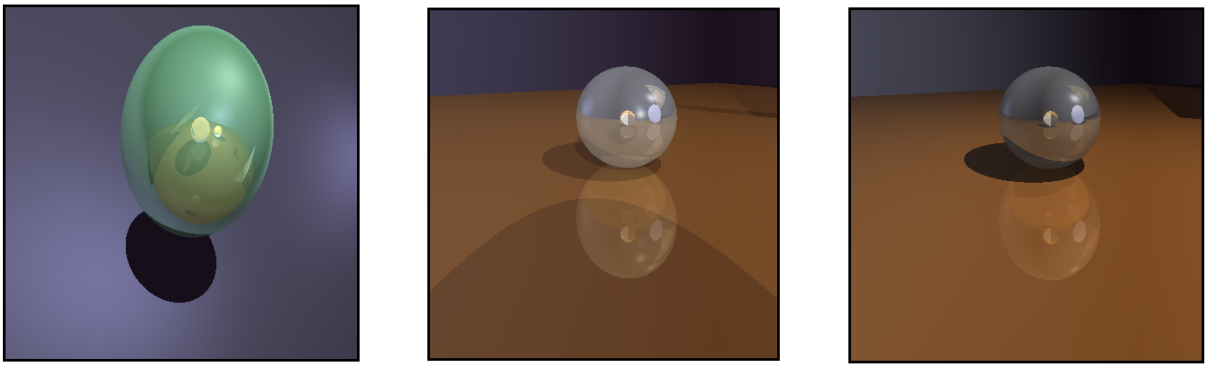

Figura 5: Vista "Jade sphere" amb 9 rebots (esquerra). Vista "Front sphere" amb 5 rebots (centre). Vista "Front sphere" amb 5 rebots i llum 2 apagada (dreta)

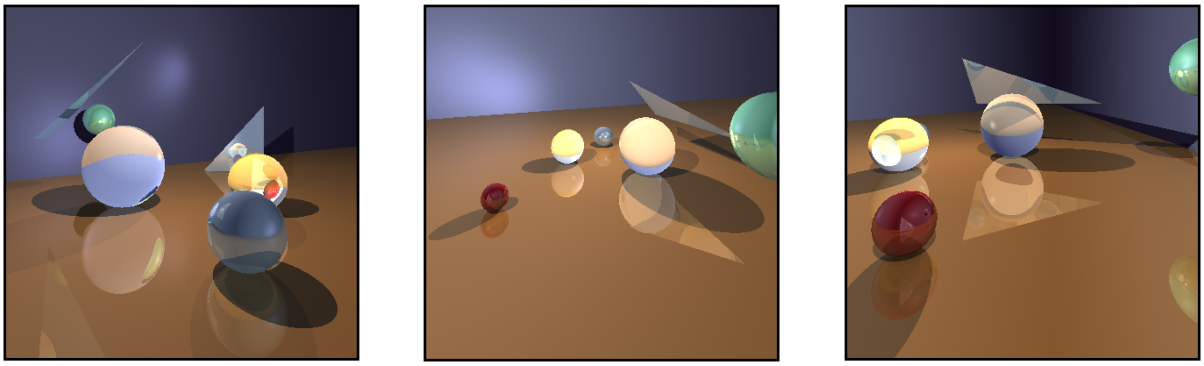

Figura 6: Vista "Left side" (esquerra). Vista "Back" (centre). Vista "Right side" (dreta)