Gràfics

Informe
Pràctica 2

Oleksandr Danylenko
Xavi Cano
A05

Index

1.Objectes

Respecte al material, l'hem definit com a public.

Esfera:

Per calcular l'esfera necessitem dues coses, el radi i el punt central.

Hi han tres casos de intersecció entre el raig i l'esfera:

- El raig no toca la esfera(no hi ha intersecció)

- El raig toca l'esfera en un punt (toca només en un punt)

- El raig intersecta l'esfera en dos punts, on escogim la mes propera a l'observador sense que sigui negativa.

Al principi al crear l'esfera hem fet els càlculs a ma, però no ens acabava de sortir molt be i vam recórrer a utilitzar el càlcul de la Viquipèdia , linea 24 de Object.cpp

Pla:

Amb el pla , ens han sorgit molts problemes a l'hora de fer els càlculs:

Hi han tres casos de intersecció entre el raig i l'esfera:

- El raig no toca el pla(no hi ha intersecció)

- El raig es paral·lel al pla.

- El raig interseca en un punt del pla.

Problemes amb errors de precisió alhora de calcular els punts del pla:

Al principi vam fer els càlculs i ens mostrava be el pla però no el color, això ho vam poder corregir l'ultim dia amb un fabs(valor).

2.Lums

Per crear la llum, hem creat una classe llum i un llistat de llums.
Els atributs utilitzats en aquesta classe son els mateixos que en la pràctica anterior amb petites diferencies.

3.Camara

Per calcular be la càmera, hem afegit dos atributs (amplada i alçada) i a partir d'aquests atributs hem calculat l'angle d'obertura i els matrius de Projection i ModelView, Camera.cpp linea27.

4.Raigs

Fem el càlcul de les matrius viewInverse * projectInverse abans de calcular cada raig per estalviar els càlculs.

Ens han sorgit molts problemes alhora d'entendre be com fer el càlcul de les matrius de Projecció i la modelView per poder calcular el raig principal, la raó principal era perquè fèiem els càlculs en un ordre incorrecte.

No hem aconseguit fer el raig primari sense normalitzar la seva direcció.

5. Blinn-Phong,Ombres,Reflexions

- Per fer el calcul de Blinn Phong hem utilitzat el codi de la practica anterior i a mes el pseudocodi de les transparències de teoria per calcular ombres i reflexions, separant-lo en varies funcions.

- Per evitar els problemes de selfShadowing i selfreflections hem afegit una epsilon al càlcul de reflexions i ombres.

- No utilitzem payload.numBounces perquè tenim el calcul de color separat en varies funcions per això hem afegit una variable addicional int reflect_number a la funció CastRay

Breu explicació de les funcions usant pseudocodi:

object_color: ens calcula el color Blinn Phong para una llum la ombra i totes les reflexions derivades.

object color
    si hi ha ombra, fer calcul d'ombra
    sino, fem Blinn Phong + (Reflexions * ks)
    per calcular reflexions fem Castray(raigReflexio,payload,reflect_number+1)

shadowInfo comprova si hi ha algun objecte entre mig de la llum i el nostre objecte
    Al primer objecte que troba retorna true

CastRay(ray ,payload,reflectnumber)
    Per cada llum fa el calcul de object_color i el suma al ambient_global

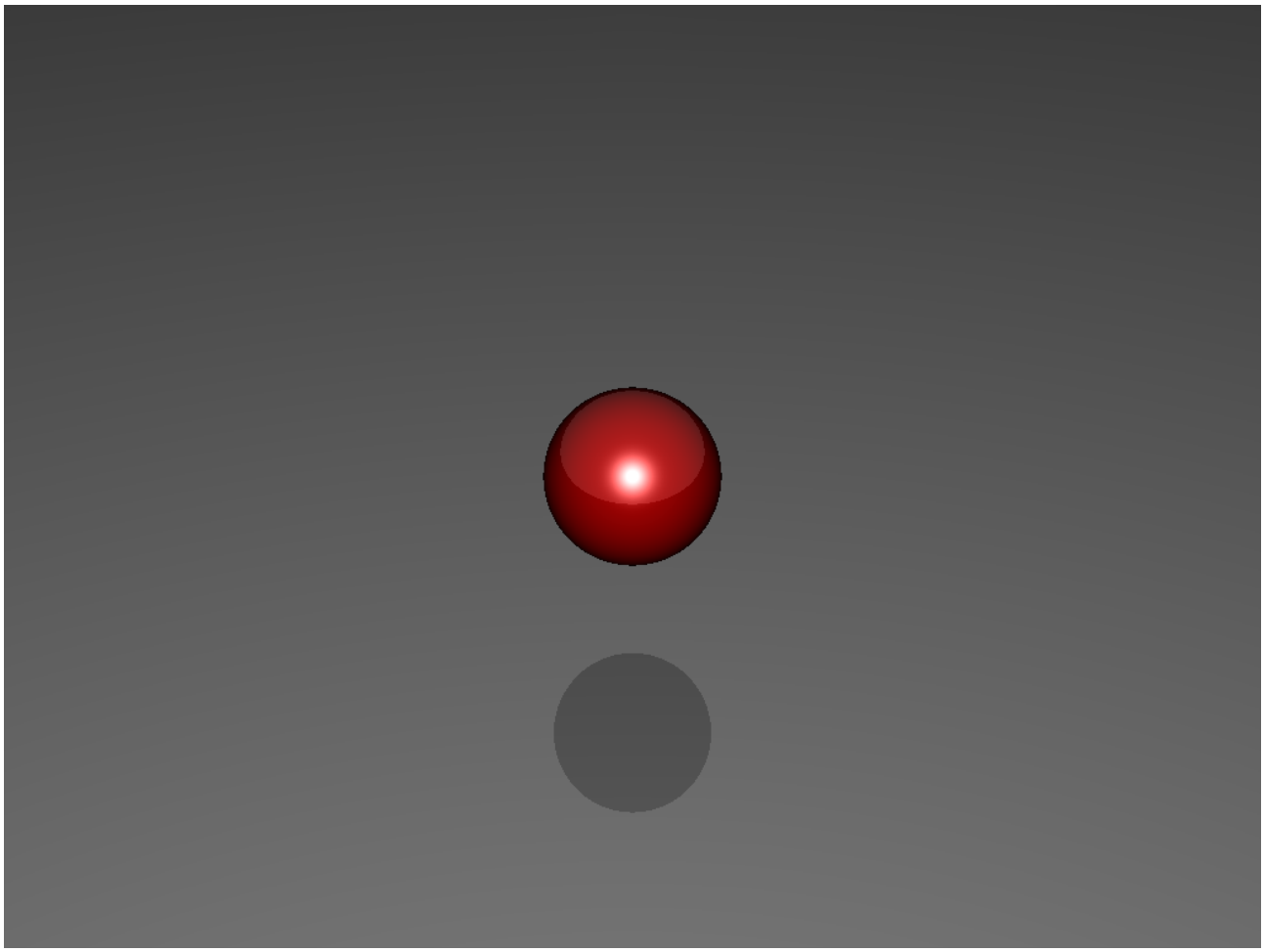
6.Conclusions

- Hem vist que ens ha faltat base d'algebra per poder entendre be molts dels càlculs , però que amb l'ajuda de la Viquipèdia, stackoverflow i les transparències de teoria hem aconseguit solucionar la majoria dels problemes que ens han sorgit.

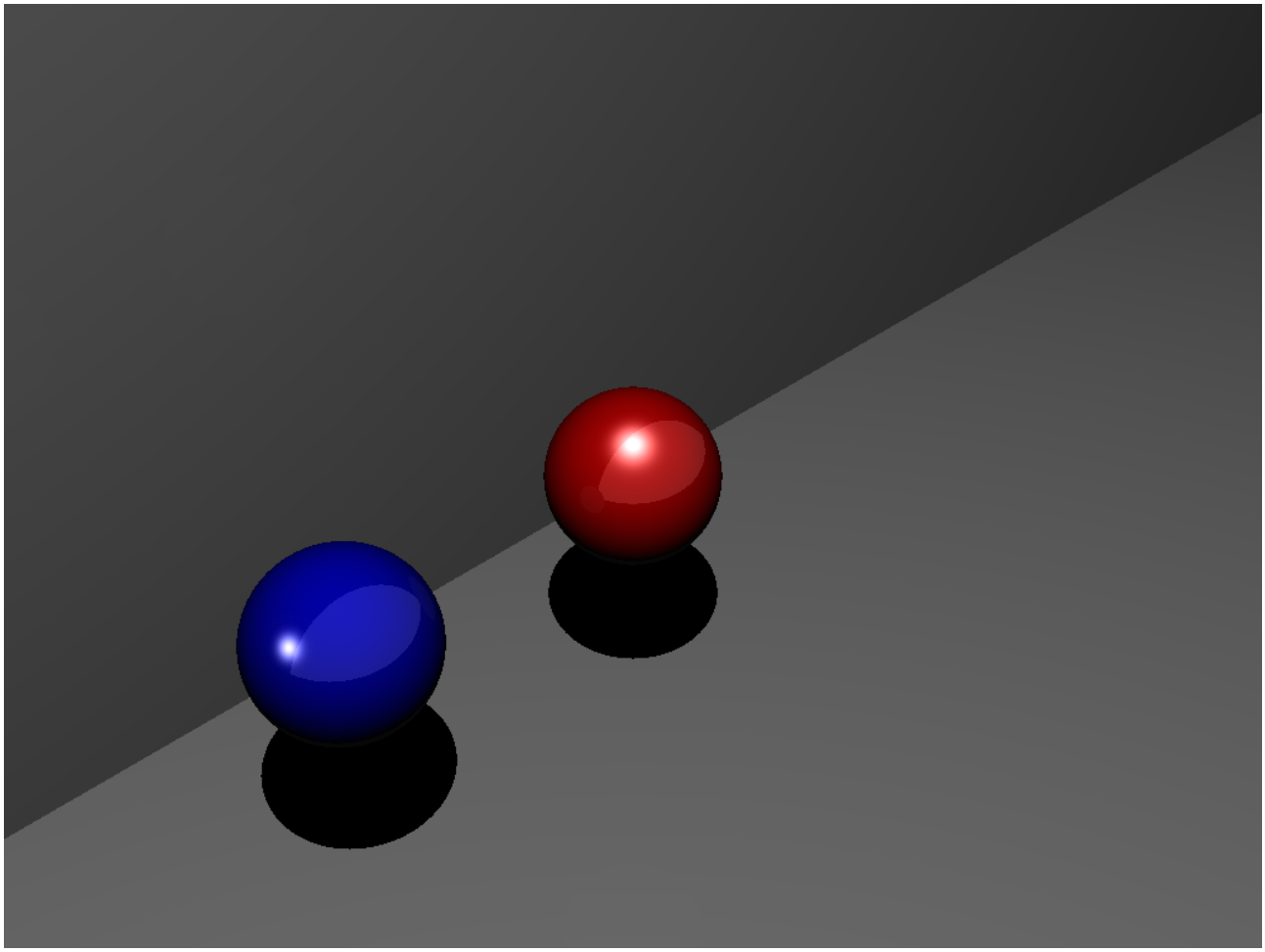
- Ens hagués agradat poder tenir mes temps per poder implementar totes les parts opcionals.

7.Banc de proves

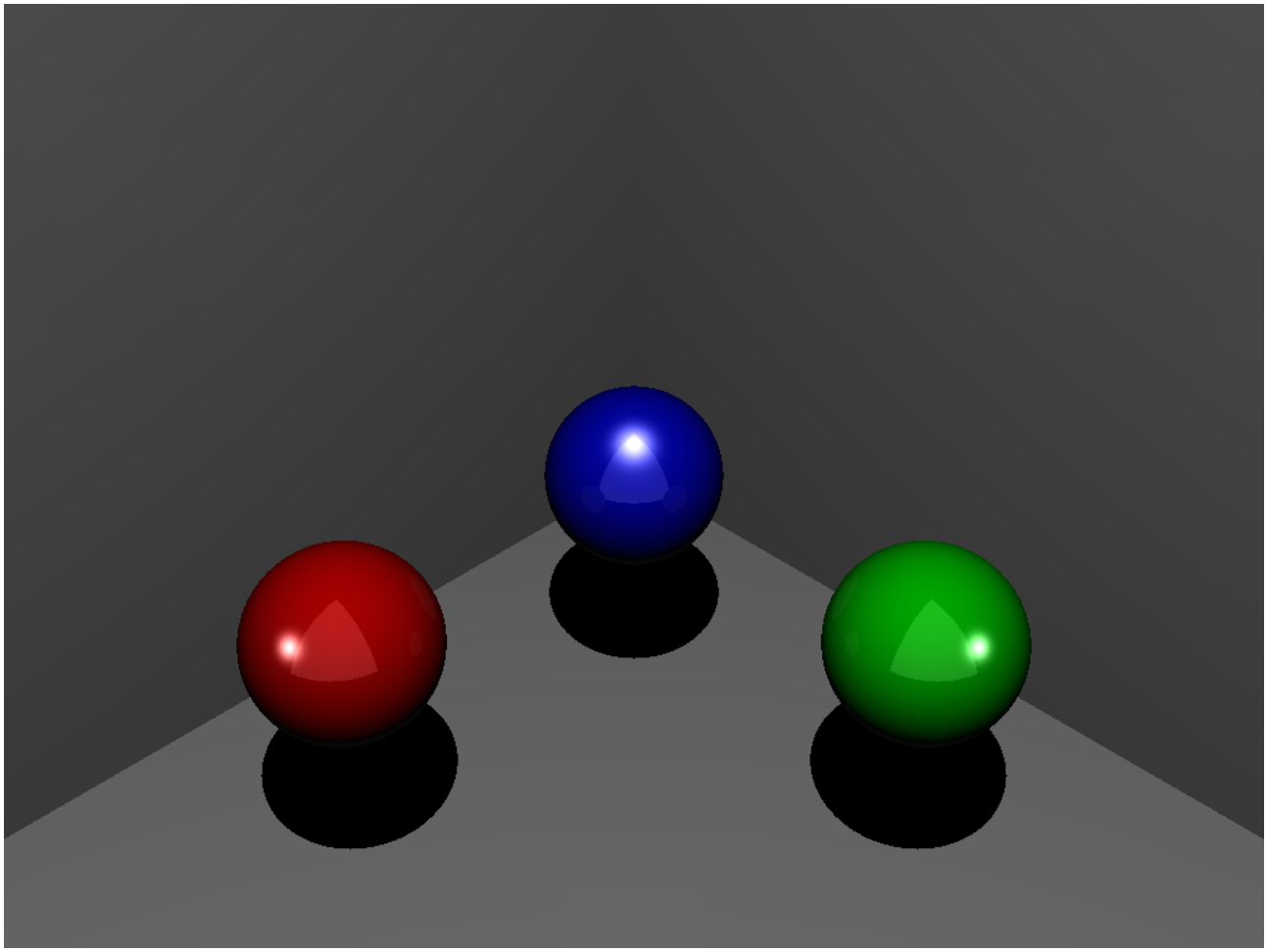
1. **1 esfera + 1 pla (1 reflexio):**



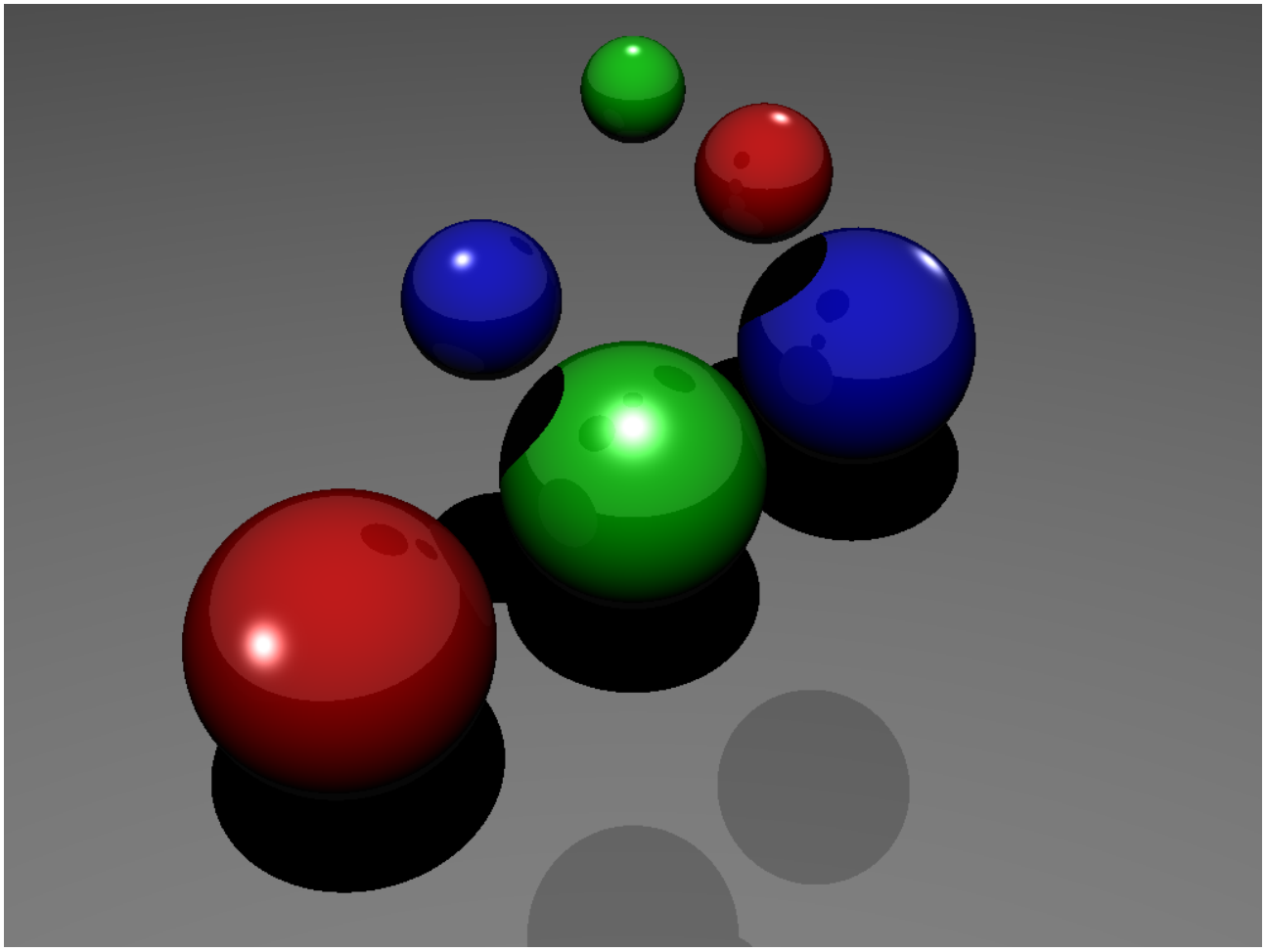
2. 2 esfera + 2 plans (1 reflexions):



3. **3 esferes + 3 plans (1 reflexions):**



4. **6 esferas + 1 pla (1 reflexio):**



5. **6 esferas + 1 pla (2 reflexio):**