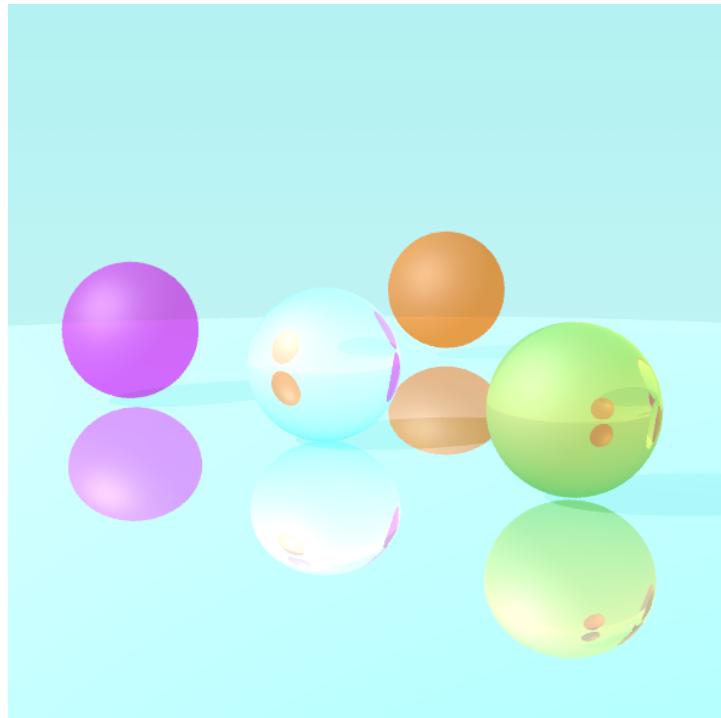


VisRayTrace 2021-22



Gràfics i Visualització de Dades

Pràctica 1 - Grup B01

Nazar Puriy
Gerard Perelló
Ferran Espuña
Masa Gulyás

Introducció	3
Funcionalitats implementades	3
Fase 1	3
Fase 2	3
Fase 3	4
Opcionals	4
Comentaris d'implementacions	5
Reflexos i multiple-scattering	5
Penombres	5
Exemple	5
Spotlight	6
Exemple	7
Ombres atenuades i de colors	7
Exemple	8
Ambient Occlusion	9
Exemple	10
Defocus Blur	11
Exemple	11
Textures sobre el pla afitat i l'esfera	12
Hit triangle	14
Creació de nous objectes: box. Bounding Box.	15
Bounding Box Matrix - Optimizing Mesh	15
Coordenades esfèriques per a dades reals.	16
Dobles valors per a les dades reals.	16
Altres objectes paramètrics	17
Conclusions	19

1. Introducció

En aquesta pràctica s'ha implementat el model de Blinn-Phong, juntament amb l'algorisme de RayTracing i altres algorismes de gràfics per computador amb la finalitat d'aconseguir imatges realistes de mons 3d i de representacions de dades reals.

En aquest informe es pot trobar, primer de tot, una llista de les tasques realitzades i per quins membres del grups han estat fetes. Tots seguit trobarem explicacions de detalls d'implementacions (en les que hi hagi especial interès per fer-ho) amb exemples d'imatges obtingudes. També s'inclouen les característiques utilitzades per aconseguir aquestes visualitzacions: comentaris de les possibles variables en el JSON o valors de constants utilitzats. Aquestes constants es troben generalment al principi dels arxius, sobretot en **scene.cpp** i **RayTracing.cpp**. Les imatges d'aquest document s'inclouen també en la carpeta zip en la que es troba l'informe, a més d'altres que ens han semblat interessants.

A més a més, volem destacar que en el **Main.cpp** hi ha un switch amb el que podem seleccionar quina visualització carregar (per no haver d'introduir els JSON manualment).

2. Funcionalitats implementades

Fase 1

[X] Background amb degradat	Tots
[X] Hit triangle	Ferran Espuña
[X] Hit boundary object	Ferran Espuña
[X] Hit cilindre	Nazar Puriy
[X] Transformacions Translació i Escalat amb gismos esferes	Masa Gulyás
[X] Pla de terra	Masa & Gerard
[X] Gizmo Triangle	Gerard Perelló
[X] Gizmo Cilindre	Masa Gulyás
[X] Noves dades	Gerard Perelló

Fase 2

[X] Antialiasing	Nazar Puriy
[X] Gamma Correction	Nazar Puriy

[X] Blinn-Phong	Nazar Puriy
[X] Ombres amb objectes opacs	Nazar Puriy
[X] Reflexions	Ferran Espuña
[X] Transparències	Masa Gulyás
[X] Visualitzacions amb dades reals	Gerard Perelló

Fase 3

[X] Texture mapping en el pla	Masa Gulyás
[X] Material textura	Masa Gulyás
[X] Nova escena de dades	Gerard Perelló

Opcionals

[X] Nous objectes paramètrics (BOX)	Gerard Perelló
[X] Penombres	Nazar Puriy
[X] Diferents tipus de llums	Nazar Puriy
[X] Multiple-scattering	Ferran Espuña
[] Escena CSG	-
[X] Ambient occlusion	Nazar Puriy
[X] Defocus blur	Nazar Puriy
[X] Més d'una propietat en les dades reals	Gerard Perelló
[X] Bounding Box	Gerard Perelló
[X] Bounding Box Matrix	Gerard Perelló
[] Animacions amb dades temporals	-
[X] Ombres atenuades segons objectes transparents	Nazar Puriy
[X] Colors d'ombra segons els colors dels objectes transparents	Nazar Puriy
[X] Mapeig de les dades reals en una esfera	Gerard Perelló
[] Ús de diferents paletes	-
[X] Texture mapping en l'esfera	Masa Gulyás

3. Comentaris d'implementacions

En aquesta secció es comenten les diferents implementacions incloent imatges generades d'exemple. Notem que no s'ha descrit exhaustivament tot el codi ja que la majoria de detalls venen perfectament indicats per part del professor. En aquesta secció sobretot hi ha comentaris d'implementacions opcionals o de decisions de disseny.

Reflexos i multiple-scattering

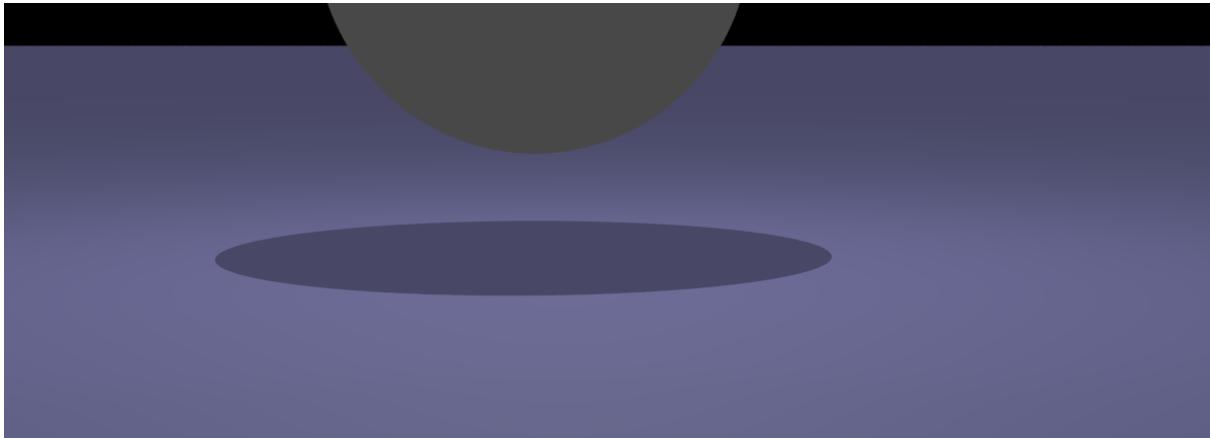
S'ha programat per a cada material diferent un mètode `getOneScatteredRay`

Penombres

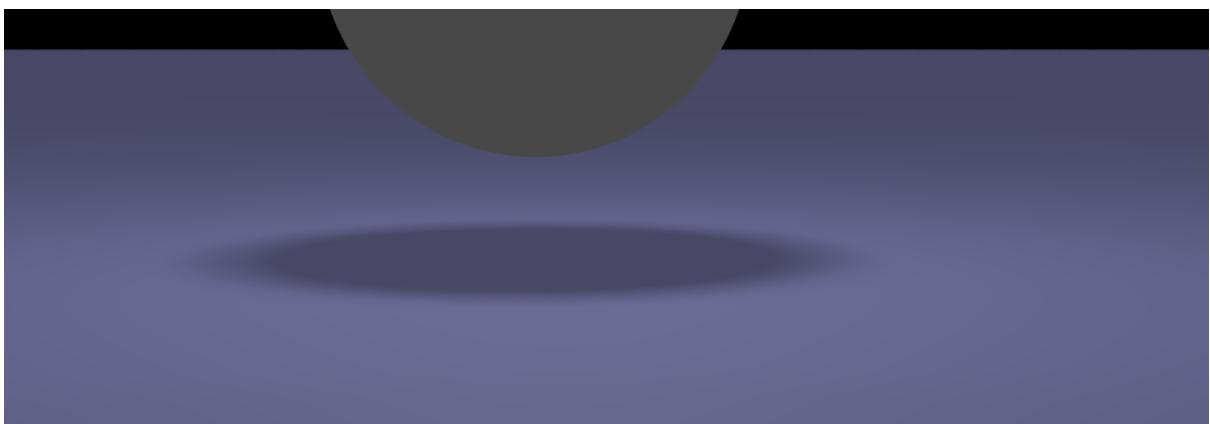
Per a la creació de penombres s'ha creat un altre tipus de llum anomenada `AreaLight` (representa una llum rectangular). Aquesta llum té les mateixes característiques que la llum puntual més alguns petits detalls addicionals. El punt clau es troba en el mètode de shading (Blinn-Phong). Quan es detecta que és una llum d'aquest tipus es procedeix anàlogament com amb la llum puntual però amb un petit detall. El mètode de shadow es crida amb n llums aleatòries que depenen de l'àrea de la nostra llum. Per generar aquestes llums aleatòries en la pròpia llum tenim un centre fix i un centre mòbil. Aleshores la classe d'aquesta llum té un mètode que mou el centre mòbil de forma aleatòria dins del rectangle de la llum i a continuació es crida el mètode de shadow que realitzarà tots els càlculs com si es tractés d'una llum puntual amb el centre aquest punt aleatori. Finalment dels shadows obtinguts es fa una mitjana i es resejea el centre aleatori per a que coincideixi amb el centre fix.

Exemples

Opció `Penombres` al main. S'han “apagat” els rajos secundaris per no generar il·luminació extra i que es pugui veure l'efecte amb més claredat. `AA_SAMPLES = 40` i `RAYTRACING_MAXDEPTH = 0` i `RAYTRACING_MULTIPLE_MAXDEPTH = 0` en el mètodes `RayTracing.cpp` i `AO=false` en `scene.cpp`. Podem veure en la primera imatge l'efecte amb el tipus de llum puntual i en la segona amb la llum en àrea.



Llum puntual efecte d'ombra afilada



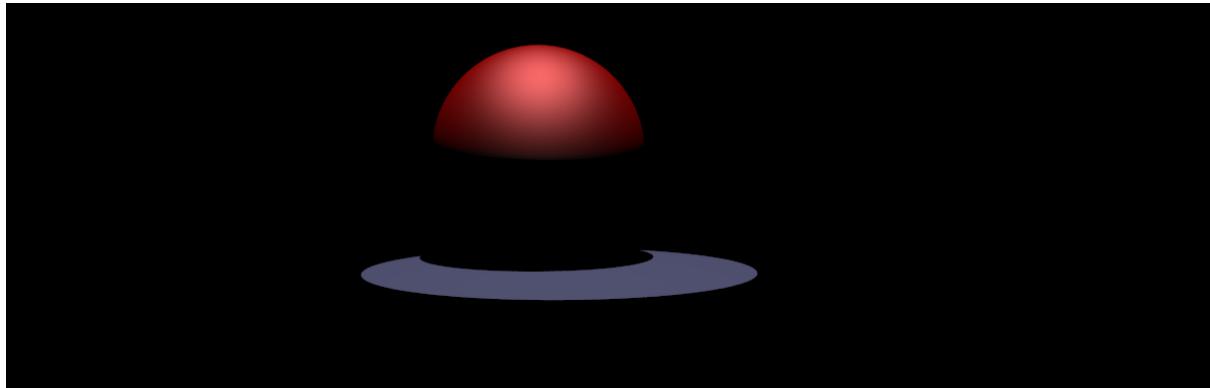
Llum en àrea efecte d'ombra suau

Spotlight

S'ha afegit una spotlight o llum en forma de con. Funciona igual que la point light, però afegeix dos valors més que són un vector cap a on apunta i amb quin angle ho fa. Si el cosinus de l'angle és més petit que el cosinus del producte escalar entre el vector de la llum i el vector amb el que mirem nosaltres a aquesta (des del nostre punt) retornem la ponderació habitual de l'atenuació (com en point light). Si el que passa és que és més gran vol dir que estem fora del rang de la llum i, per tant, retornem un 0.

Exemples

Opció **Transparencies** al main. Constants **AA_SAMPLES = 40**, **RAYTRACING_MAXDEPTH = 0** i **RAYTRACING_MULTIPLE_MAXDEPTH = 0** en el mètodes **RayTracing.cpp** i els dos amb **AO=false** en **scene.cpp**. Si ens fixem les llums ambient es troben a 0 també per donar aquest tipus d'efecte.



Llum en forma de con projectada en una esfera.

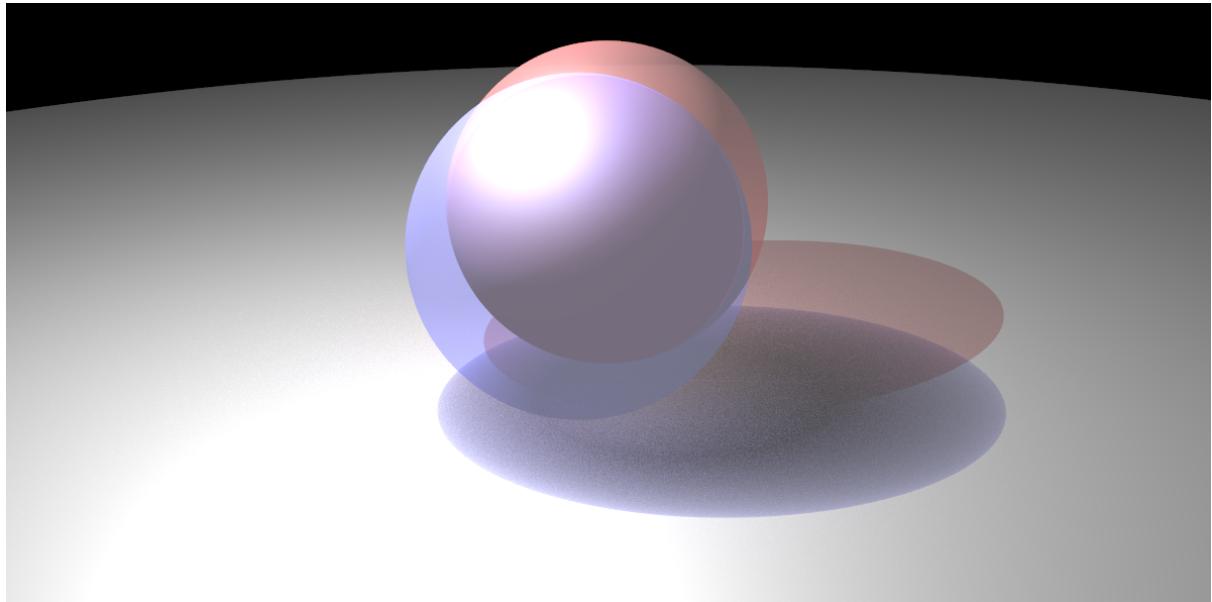
Ombres atenuades i de colors

L'objectiu principal és modificar els valors de les llums difuses i especulars segons els objectes que es trobin entremig de la llum i del punt. Per fer-ho ens guardem, en forma de vector, el factor per al que acabarem multiplicant l'efecte de la llum. Aleshores en aquest mètode el que fem és des del punt llançar un raig fins la llum i anar apuntant els xocs, en línia recta, que va fent. Per tant tenim un bucle while que genera rajos i analitza xocs. Si el raig xoca un objecte opac retornem (0,0,0) i sortim del mètode. Si l'objecte que xoquem és transparent ens guardem en un diccionari el seu identificador (cal afegir identificadors al JSON o bé en carregar els objectes) juntament amb la informació de on ha xocat i la kt que hi ha en aquell punt. Si en xocar amb l'objecte el seu identificador ja estava al diccionari el traiem d'aquest i amb el nou punt i el que tenim guardat calculem la distància recorreguda. Aquesta distància juntament amb la màxima que podem recórrer dins l'objecte (donada en el JSON) ens permet obtenir un factor $(1-\min(\text{dist}/\text{distmax},0))$ d'atenuació que multipliquem per la Kt que ens dona el color. Això ho fem fins arribar a la llum (ènfasis en que anem en línia recta) o trobem un objecte opac. Si en sortir encara hi ha presents objectes en els diccionaris multipliquem únicament per la seva kt ja que a l'entrar en l'objecte i no sortir-ne és com si el freguéssim.

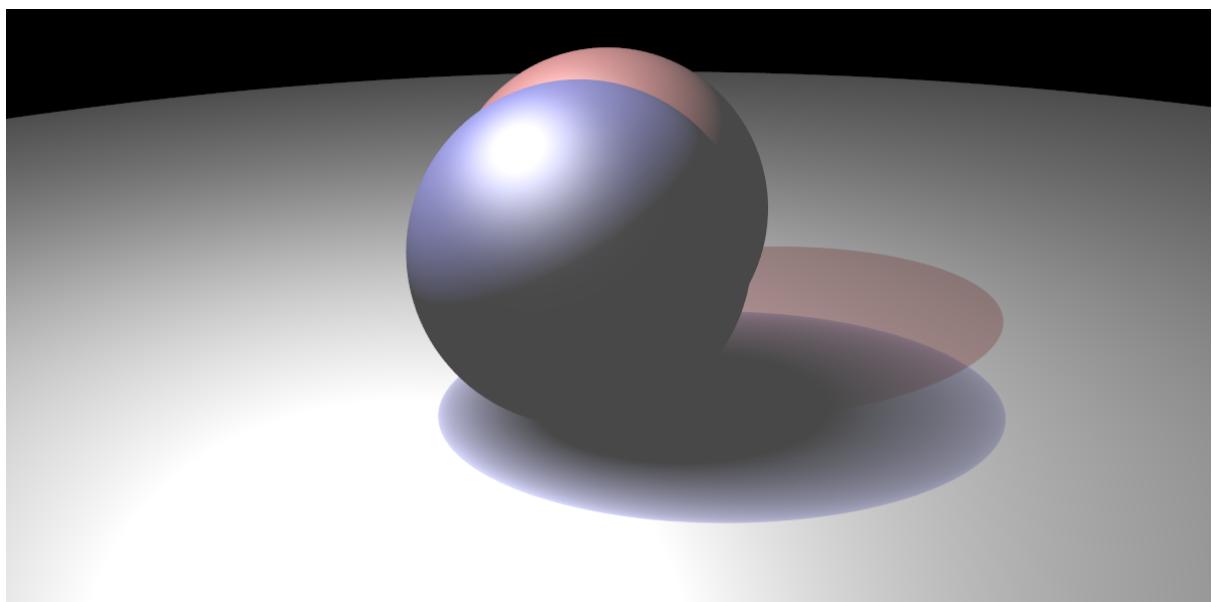
Exemples

Opció **Transparencies** al main. La primera imatge té els rajos secundaris activat i s'ha realitzat amb les constants AA_SAMPLES = 40, RAYTRACING_MAXDEPTH = 0 i RAYTRACING_MULTIPLE_MAXDEPTH = 0 en el mètodes RayTracing.cpp i els dos amb AO=false en scene.cpp. El segon exemple permet veure amb més claredat l'efecte obtingut ja que les reflexions indirectes no generen llum; obviament les boles no es veuen transparents perquè no

hi ha rajos reflectits. D'aquesta forma podem veure les ombres de colors, l'ombra blava atenuada i fins i tot un color lila entremig de les dues.



Efecte de transparències amb Ray Tracing encés



Efecte de transparències sense rajos secundaris

Ambient Occlusion

El que pretén ambient occlusion és que la llum ambient global i la generada per les llums no sigui constant i depengui de l'oclusió real dels objectes. Per utilitzar això tenim dos mètodes, en `scene.cpp`, encarregats de calcular un factor per el qual multipliquem les llums ambient.

Aquests dos mètodes són cridats en el mètode shading on és calcula la il·luminació del model Blinn-Phong.

El primer mètode és el getGlobalAmbientOcclusion que s'encarrega de llençar NRAOG(constant que es troba a l'inici de scene.cpp) rajos de forma aleatoria i veure quins tenen impacte o quins no (arriben al cel). Posteriorment retorna la proporció de rajos impactats entre els llançats. Aquesta proporció l'utilitzarem per atenuar la llum ambient global.

El segon mètode és el getLocalAmbientOcclusion que s'encarrega de llençar NRAOL(constant que es troba a l'inici de scene.cpp) rajos aleatoris. En aquest cas tenim també una altre constant IGNDIST que ens indica la distància màxima en la que es pot trobar un exemple per ser oclusor de llum. Si tenim un hit amb un objecte sumem la proporció de la distància a la que ha estat el hit entre aquesta distància màxima. Finalment retornarem aquesta suma partida per el nombre de rajos llençats. Notem que si no hi ha hagut hit o bé ha sigut a més de la distància màxima sumarem un 1. La proporció retornada s'utilitzarà per atenuar les llums ambients generades per diferents llums de l'escena.

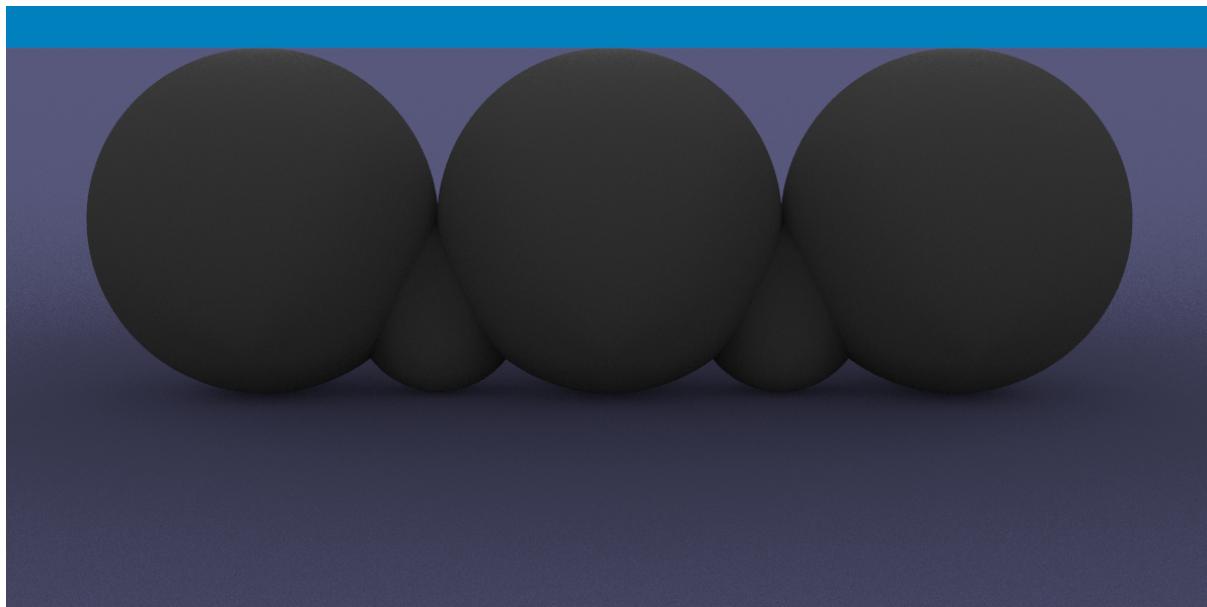
Una observació a comentar és que en el mètode de shading cal obtenir el pla perpendicular a la normal i una base unitària de R3 amb dos vectors d'aquest pla i la normal. D'aquesta forma podem generar rajos aleatoris en el semiespai definit per la direcció de la normal agafant valors positius d'aquesta en crear els vectors/rajos aleatoris. Notem que es pot desactivar ambient occlusion amb la constant AO a l'inici de scene.cpp

Exemples

Opció **AmbientOcclusion** al main. Per generar les screenshots s'han utilitzat les constants NRAOG = 20, NRAOL = 20 IGNDIST = 100.0 a scene.cpp, AA_SAMPLES = 20 i RAYTRACING_MAXDEPTH = 0 i RAYTRACING_MULTIPLE_MAXDEPTH = 0 a RayTracing.cpp ja que no ens interessa obtenir il·luminació causada pels rajos secundaris. Per altra banda les úniques llums presents a l'escena són les llums ambient. Podem veure en les següents imatges la comparació de ambient occlusion on i off.



Ambient Occlusion OFF



Ambient Occlusion ON

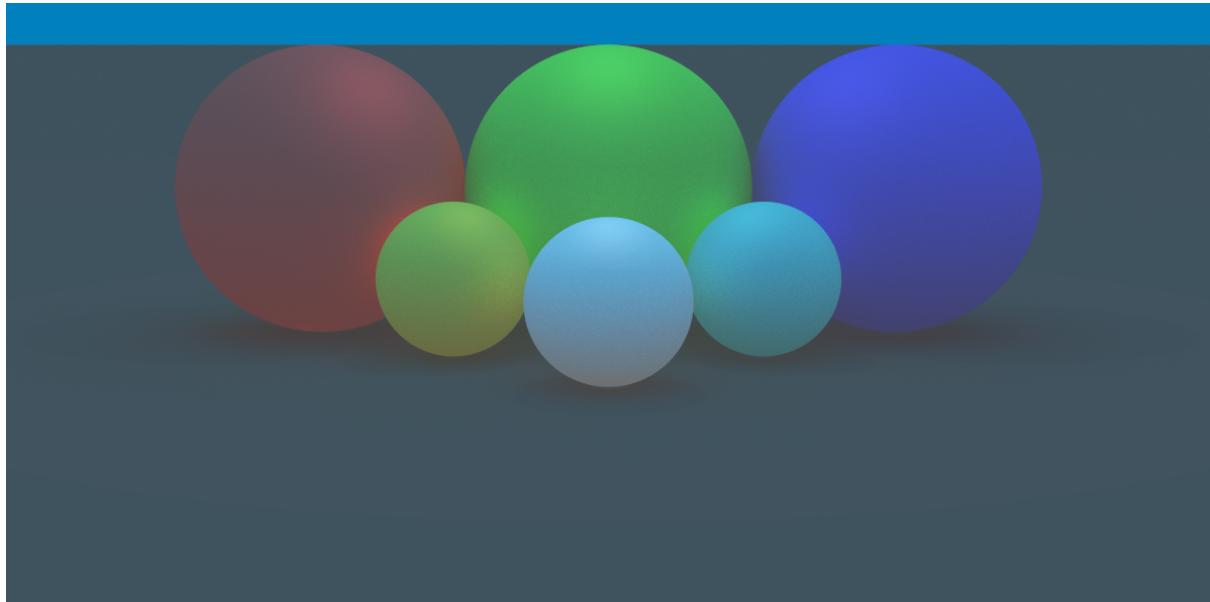
Defocus Blur

En aquest cas no s'ha de realitzar cap implementació. únicament en la càmara hem d'afegir un Json anomenat defocusBlur que conté dues variables: enables i aperture, autoexplicatives. Per explicar el funcionament fixem-nos que a RayTracing es crida cam->getRay(), aleshores aquí

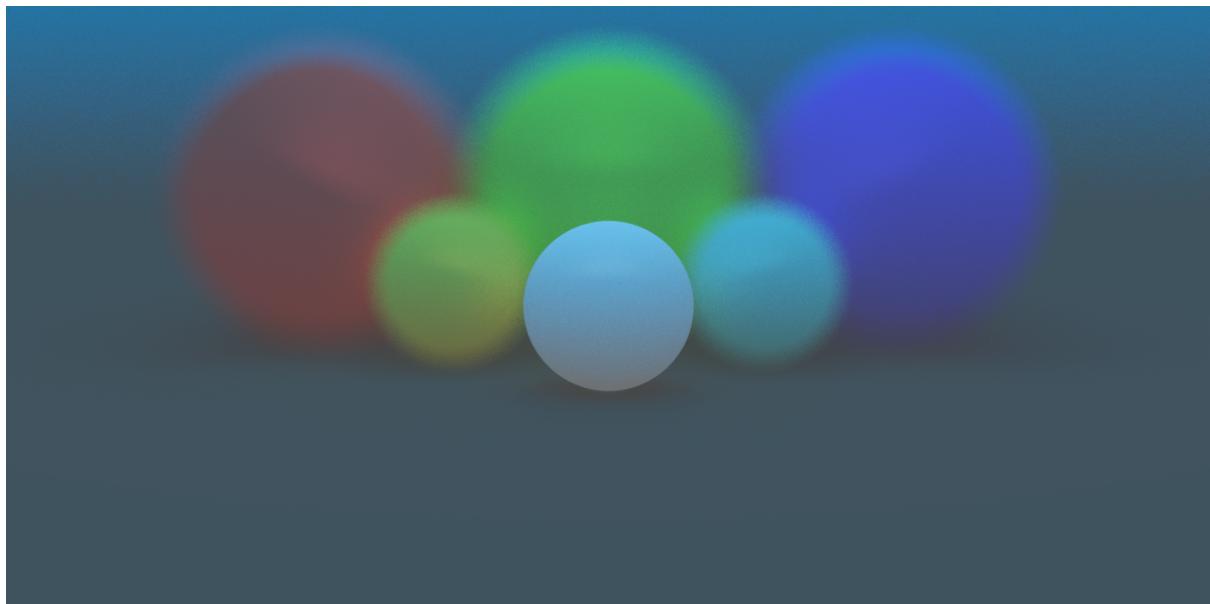
quan s'arriba a aquest mètode si el defocus.blur està activat es crida el mètode de getBlurRay que genera un punt aleatori en un disc de radi aperture i suma aquest al centre.

Exemples

Opció `DefocusBlur` al main. Per generar les screenshots s'han utilitzat `AA_SAMPLES = 60`, `RAYTRACING_MAXDEPTH = 6` i `RAYTRACING_MULTIPLE_MAXDEPTH = 0` a `RayTracing.cpp`. S'ha desactivat el `ambientOcclusion` en `scene.cpp`.



Defocus Blur OFF



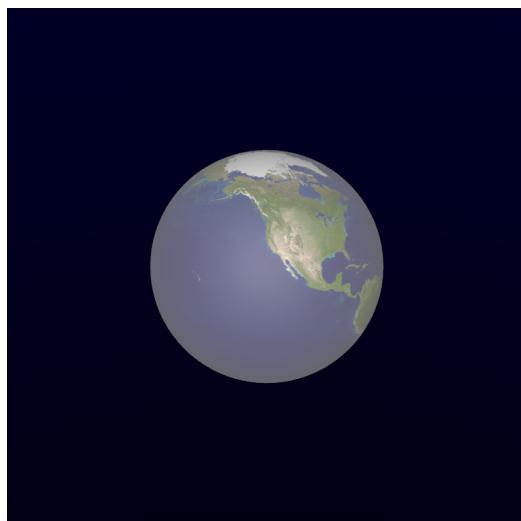
Defocus Blur ON

Textures sobre el pla afitat i l'esfera

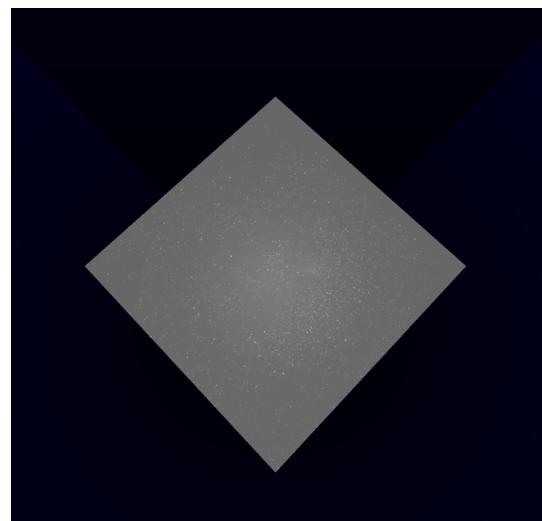
Per aplicar una textura a un objecte, hem creat un nou material **Texture**, on en lloc de retornar un **Kd** únic per tot el material s'obté el color d'un píxel de la imatge associada a la textura depenent de les coordenades paramètriques de l'objecte on s'ha produït l'impacte amb el raig. Tal i com s'indica a l'enunciat, calculem aquestes coordenades paramètriques per cada impacte en el mètode **closestHit** de l'objecte (encara que el material no sigui textura). El càlcul de les coordenades paramètriques es fa dins de cada objecte que ho impFittedPlanelements (i **Sphere**) en un mètode void anomenat **getParametricPoint**, que guarda aquesta informació en el paràmetre **uv** del **HitInfo** donat per **closestHit**, perquè més tard sigui accessible des de la classe **Texture**.

Exemples

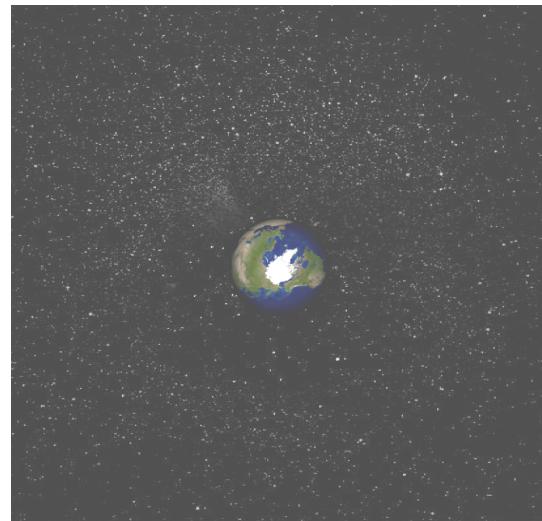
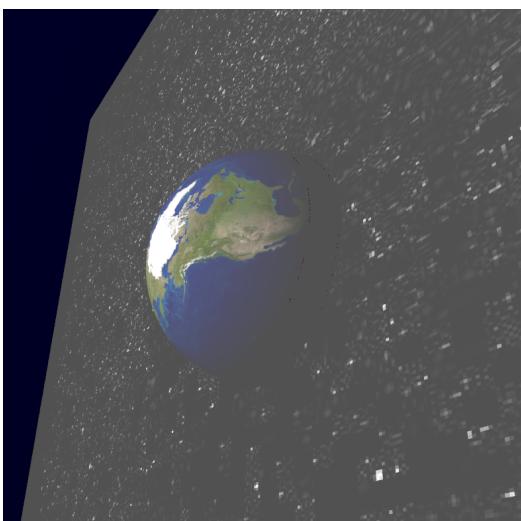
Seleccionant les opcions al main: **Earth**, **Stars**, **EarthWithStars**, **EarthWithStarsAndMoon**.



Textura (mapa del món) en una esfera

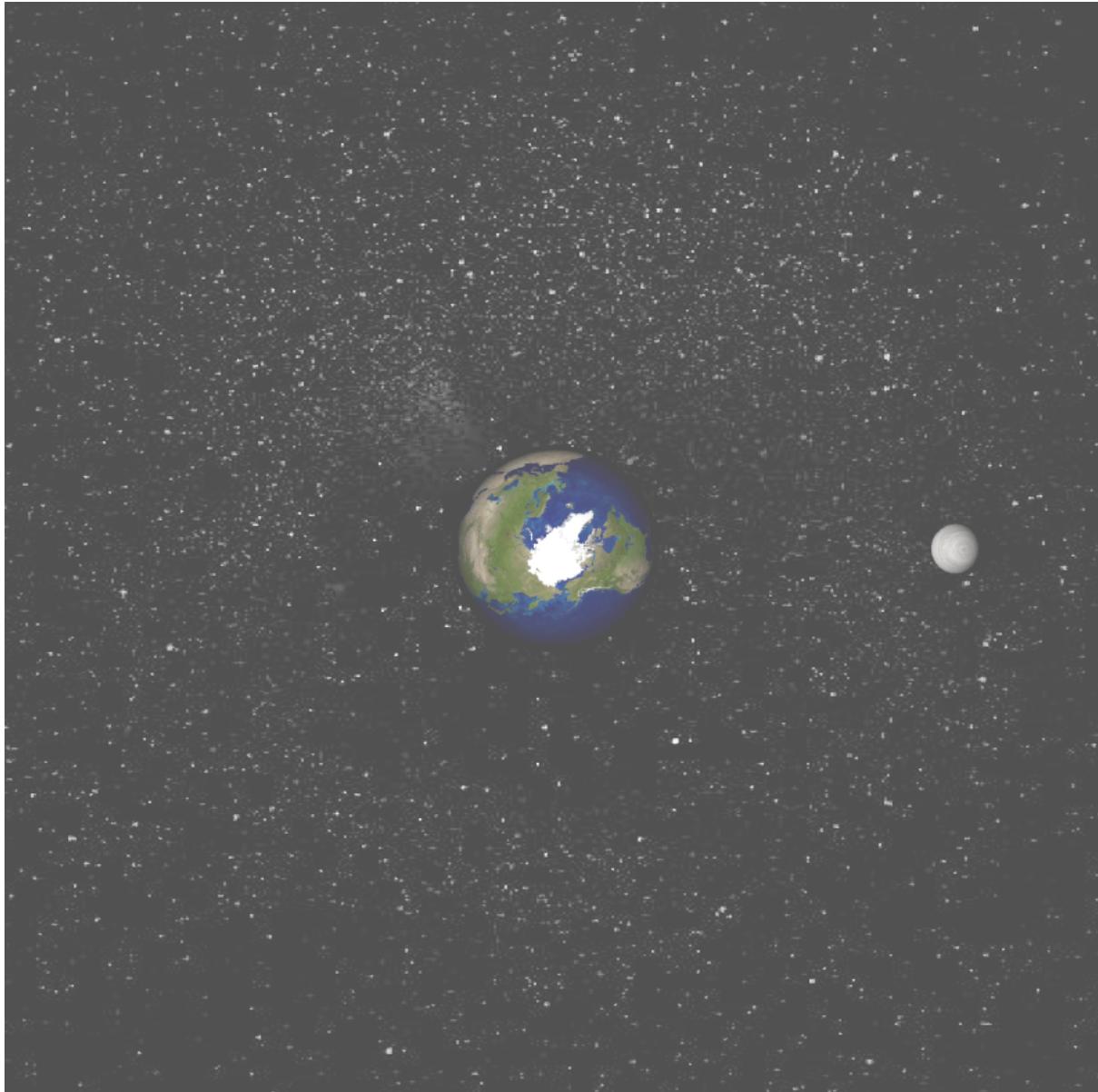


Textura (estrelles) en un pla afitat



Esfera sobre un pla afitat amb textures

Esfera sobre un pla afitat vist des de dalt



Dues esferes amb textura de la Terra i la Lluna sobre un pla afitat amb textura d'estrelles

Hit triangle

Es demanava trobar la intersecció (o no) entre raig i triangle. Vam trobar una manera de calcular les coordenades baricèntriques del punt d'intersecció respecte els tres vèrtexs del triangle que ens permetia fer una gran optimització:

Si l'origen del raig és constant per a tots els rafos, els vectors que uneixen aquest origen amb els vèrtexs del triangle són constants i defineixen una base de l'espai R^3 . En aquest sistema de referència, els vèrtexs corresponen als punts $(1,0,0)$, $(0,1,0)$ i $(0,0,1)$, i per tant defineixen el pla $x+y+z=1$. A més, els punts de l'interior del triangle corresponen amb els del pla que tenen les 3 components positives. Per tant només cal canviar la direcció del raig a aquesta base, comprovar aquest fet, i veure per què hem de multiplicar el vector per a que les coordenades sumin 1 ($1 / \text{suma de components}$) per obtenir el valor de t del raig.

Això no és una optimització fins que ens adonem que podem trobar el vector director del raig canviat de base multiplicant-lo per la matriu inversa de la formada pels tres vectors mencionats anteriorment. Aquesta matriu es calcula només una vegada (en el primer raig) i després és sempre la mateixa, assumint que no hi ha rebots. Per tant és molt útil (més de 3 vegades més ràpid) en contexts on no hi ha rebots (`RAYTRACING_MAXDEPTH = 0`).

Creació de nous objectes: box. Bounding Box.

Aprofitant la intenció de crear les Bounding Box com a caixes per les mesh, s'han creat un nou tipus d'objecte anomenat **BOX**. Les Box es creen segons un punt màxim **Pmax** i un punt mínim **Pmin**. Aquests dos punts (vec3 els dos), ens serveixen de referència per a crear els 6 plans, paral·lels 2 a 2, que formaran les box.

Les Box queden encaixades en els eixos de coordenades i per tant, les normals als seus plans sempre seran de la forma $(+/-1,0,0)$, $(0,+/-1,0)$, $(0,0,+/-1)$. Això ho hem fet per a facilitar la feina a l'hora de l'objectiu real de les Box, la qual es crear les Bounding Box per a augmentar l'eficiència a l'hora d'interceptar altres objectes més difícils d'analitzar.

Per a fer el **hasHit()** i el **closestHit()** de les Boxes, el que fem és crear un nou raig infinit (que porta el **tmin** a menys infinit i el **tmax** a infinit) i iterar el **closestHit()** amb aquest raig per a cada un dels plans. A continuació es guarden les "t" del raig per parelles de plans paral·lels. D'aquesta manera aconseguim crear intervals de "temps" entre les dimensions per on passa el raig. En cas que hi hagi intersecció en el temps en dues o més dimensions, voldrà dir que el raig ha passat pel cub i per tant haurem de retornar un True. En cas contrari, haurem de retornar False.

La bounding box s'ha utilitzat a la MESH, donada la seva alta complexitat, per a poder tenir un major rendiment a l'hora d'interceptar-la. És molt ineficient comprovar el closestHit per tots els triangles de la mesh, quan el raig ni tan sols ha de passar per allà. La bounding box ens arregla aquest problema.

En la creació de la Mesh ens guardem les coordenades més grans i més petites de tots els vertex que la componen, d'aquesta manera, abans de crear els triangles, som capaços de generar un **Pmin** i un **Pmax** per a poder crear un box que rodejarà la mesh i serà capaç de, de forma molt fàcil i eficient, dir-nos si el raig passa per la mesh. En cas de que el raig hi passi, es comprovaran els triangles; en cas contrari, es retornarà un false directament.

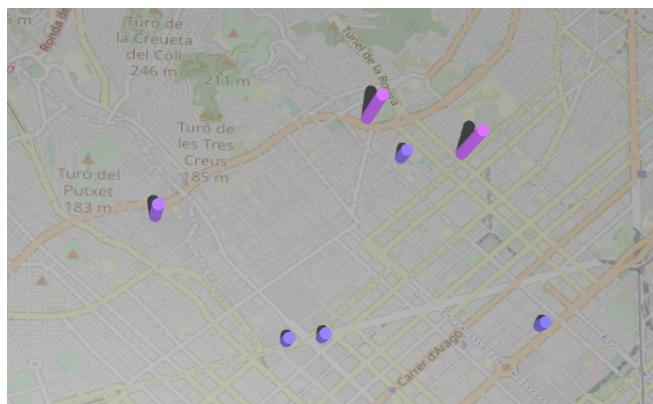
En el següent apartat s'explicarà una implementació encara més potent d'aquesta idea.

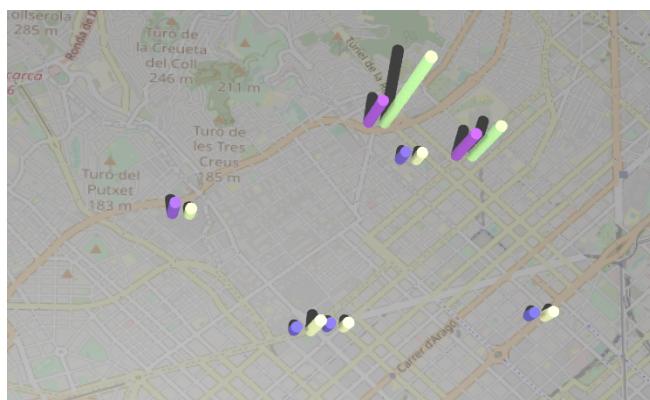
Bounding Box Matrix - Optimizing Mesh

L'idea de la Bounding Box és bona, però encara ens pot ajudar a optimitzar més el render de la Mesh

Coordenades esfèriques per a dades reals.

Dobles valors per a les dades reals.





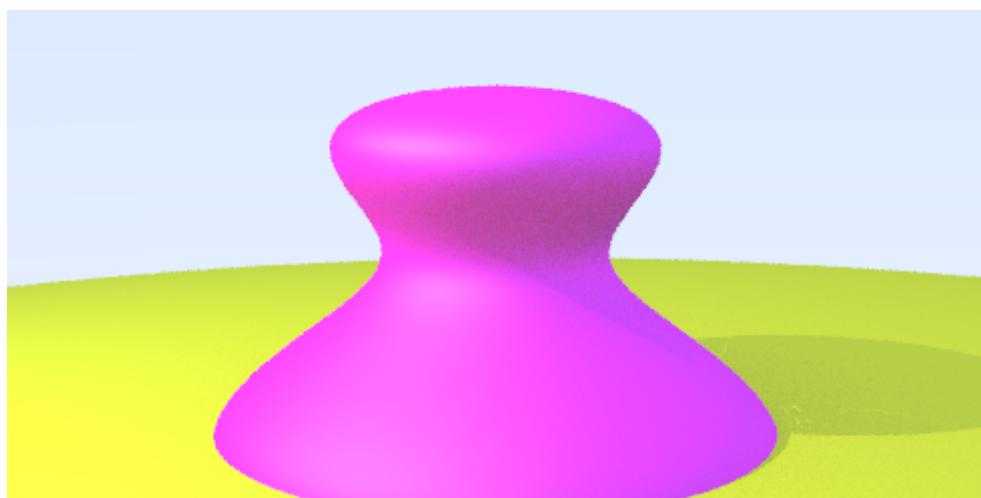
Altres objectes paramètrics

Hi havia com a tasca opcional crear nous objectes paramètrics, de manera semblant al que s'ha fet amb esferes, cilindres, plans, triangles... Hem decidit intentar anar un pas més enllà i plantejar-nos si podem visualitzar objectes del tipus $f(x) = 0$, on $f: \mathbb{R}^3 \rightarrow \mathbb{R}$ és una funció. Per temes de facilitat de lectura des d'un fitxer, així com de calcular-ne la derivada (veurem que això és convenient per diversos motius) ens hem restringit a funcions polinòmiques en les 3 variables, per exemple, $xy + z^2 - 1$.

Per llegir des d'un fitxer, introduïm el polinomi terme a terme, per exemple, el polinomi $f(x, y, z) = 0.9x^2 + z^2 + y^4 + y^3 - 3y^2 - 1 = 0$ s'escriu de la següent manera:

```
- "name": "Polinomi",
  "type": "polynomial",
  "monomis": ["0.9x2", "z2", "y4", "y3", "-3y2", "-1"],
```

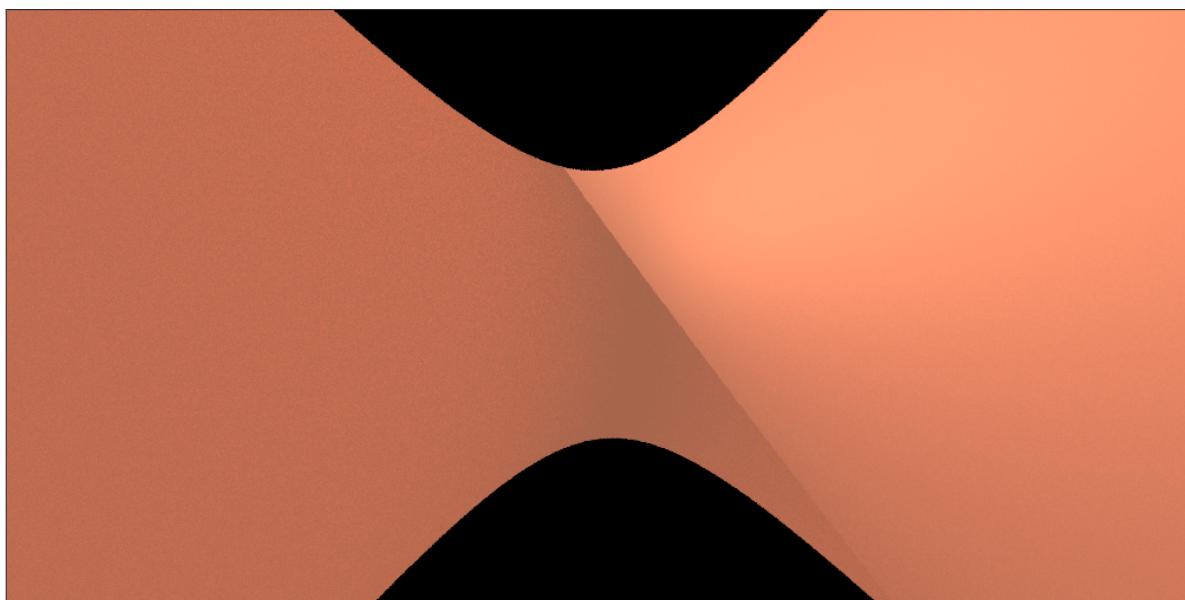
I, al visualitzar-lo, obtindrem la següent figura (hem posat una esfera a sota perquè s'apreciï l'ombra):



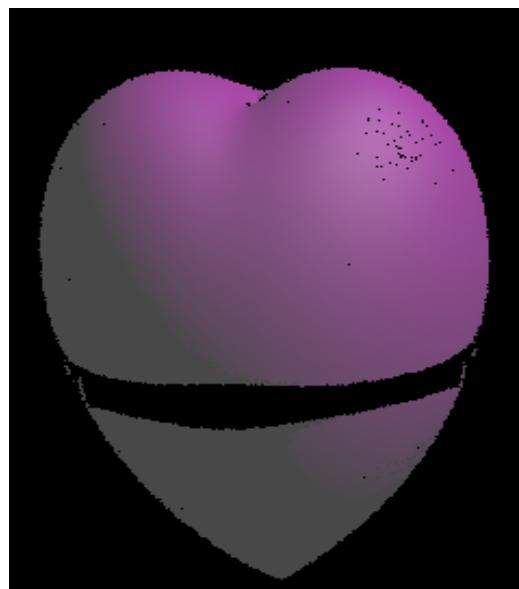
L'algorisme per a interseccar un raig amb una figura d'aquest tipus és relativament senzill, però cal una etapa de preprocés, on guardarem per a cada variable un nou polinomi, que és la derivada respecte a aquesta variable de f . El raig ens dona una correspondència amb valors reals (de t) i punts de l'espai, i f ens envia de nou a R quan el calculem en aquests punts. Aleshores es tracta de trobar els valors de t del raig per als quals $f = 0$. Per a fer-ho, utilitzem una versió “casolana” del mètode de Newton-Raphson (per a això fem servir la derivada / gradient de f , que ens permet derivar respecte t de manera senzilla i eficient). L'objectiu és trobar l'arrel més propera al t_{\min} del raig de f (evidentment, ha d'estar entre t_{\min} i t_{\max}).

Un cop trobat t (i per tant el punt d'intersecció) cal trobar la normal a l'objecte en aquest punt. Utilitzem el teorema dels valors regulars per concloure que aquesta normal serà el gradient de f al punt (per a això ens és útil de nou la derivada de f). Ens hem adonat que com més gran és el grau del polinomi, més extrems els seus valors i més

precisió necessitem per tal d'obtenir bones ombres i evitar altres problemes (punts de l'objecte que no veiem, punts que visualitzem però no hi són, soroll, etc. Per exemple, a continuació veiem un hiperboloide d'un full (polinomi de grau 2), amb les seves ombres reflexos lambertians aleatoris, etc. Ens hem pogut permetre obtenir aquest resultat bastant ràpidament ja que el mètode de Newton-Raphson convergeix ràpidament als zeros de f , i no hi ha problemes d'autointersecció amb l'ombra ja que la funció canvia ràpidament (no hi ha zeros de la derivada prop de zeros de la funció).



En canvi, vegem el que passa amb aquest cor, representat per un polinomi de grau 6:



Hem agut de desactivar tots els efectes (recursivitat, ambient occlusion, etc.) per poder augmentar la precisió del mètode, i tot i així no és suficient.

4. Conclusions

En aquesta primera pràctica, hem aconseguit implementar totes les tasques obligatòries, i també d'altres opcionals que hem trobat interessants. La nostra organització de treball va consistir en el repartiment setmanal de les tasques a realitzar entre tots els membres del grup. Cada un tenia una rama pròpia al control de versions, i feiem revisions de codi regularment abans de fer *merge* a la rama principal. En definitiva, la dinàmica de treball ha sigut òptima i hem pogut avançar setmanalment a un ritme adequat.