

Pràctica 1

RayTracing: Fase 2

GiVD - curs 2022-23

Fase 2: Materials i Il·luminació

Índex

1. INTRODUCCIÓ	1
2. PASSOS A SEGUIR	1
PAS 1. Abans de començar la Fase 2: Prova el fitxer twoSpheres.json	1
PAS 2. Considera les llums puntuals a la teva escena i implementa el shading de Blinn-Phong:.....	2
PAS 3. Afegeix ombres	4
PAS 4. Afegeix recursió en el teu mètode RayPixel per a que tingui en compte els rajos reflectits.	6
PAS 4. Afegeix recursió en el teu mètode RayPixel per a que tingui en compte els rajos dels objectes transparents i un nou material anomenat Transparent:	7
PAS 5. Visualization mapping:.....	9
EXTENSIONS OPCIONALS.....	12
Temps estimat de cada estudiant: 4 hores presencials + 10-12 hores de treball setmanal individual fora de l'aula.....	12
Material de referència:	12
APÈNDIX:.....	12
COM ES CARREGUEN ELS MATERIALS.....	12
DES DE LES DADES DE MÓN REAL COM ES FA EL MAPEIG A MATERIALS ? ÚS DE LES PALETES (o COLORMAPS).....	13

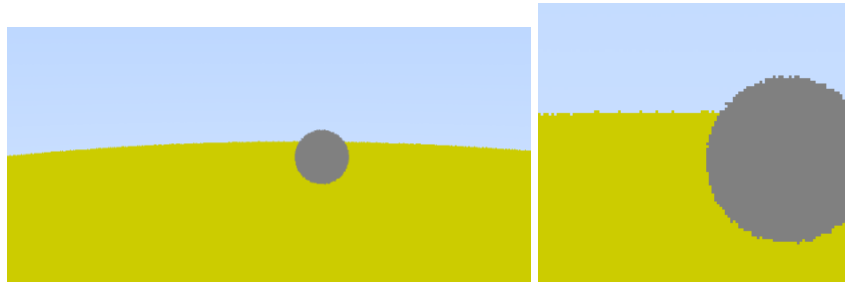
1. INTRODUCCIÓ

Al final d'aquesta fase, veuràs ja escenes correctament il·luminades. Crearàs diferents tipus de Materials i els assignaràs als objectes que has creat a la fase anterior. Afegiràs també llums i més mètodes de *shading* com el model de Phong i de Blinn-Phong. Acabaràs incloent ombres i rajos secundaris per modelar reflexions i transparències.

2. PASSOS A SEGUIR

PAS 1. Abans de començar la Fase 2: Prova el fitxer twoSpheres.json

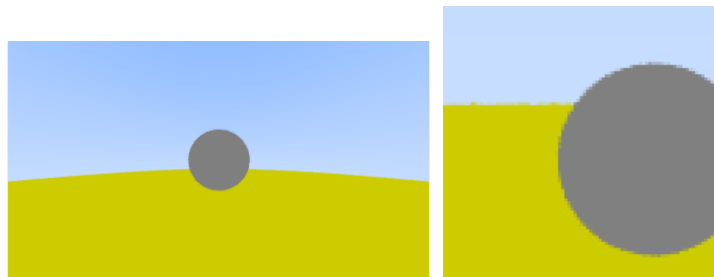
1. Calcula la imatge resultant de visualitzar les dades del fitxer [twoSpheres.json](#), amb el *setup* que tens per defecte i amb el *shading* de Color, hauria de sortir la següent visualització:



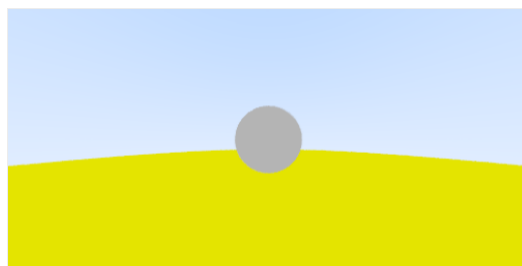
2. Fixa't que si agafes una captura de la imatge i l'escales una mica, en les fronteres de les esferes es veu un escalonat (o *aliasing*). Per a solucionar-lo, es poden tirar varis rajos per píxel i fer el promig dels valors calculats per tots els rajos. On afegiries un atribut *numSamples* que defineixi el nombre de rajos per píxel i així controlar aquest fet? Pots usar la funció `glm::linearRand(minValue, maxValue)` inclosa en el fitxer `glm/gtc/random.hpp`. Cal que incloure aquest fitxer a la classe `Ray.hh`

De vegades, quan s'acumulen valors als colors, pot ser que acabin donant valors fora del rang entre 0 i 1. No es mala idea el truncar els colors entre aquests dos valors abans de pintar (pots utilitzar el mètode `glm::clamp`)

Un cop implementat, usant 10 rajos per píxel, t'hauria de donar una visualització com:



3. Els colors es veuen una mica apagats. Per aclarir la imatge, s'utilitza una correcció del color final calculat. Aquest fet s'anomena *Gamma Correction*. Es tracta de fer l'arrel quadrada de cada canal del color just abans de pintar-lo. On faràs aquesta correcció?



PAS 2. Considera les llums puntuals a la teva escena i implementa el shading de Blinn-Phong:

En aquest punt aniràs carregant llums per a visualitzar les escenes. Recorda que les tens en a la classe `SetUp.hh` i caldrà que les carreguis des d'un fitxer de `setup` o de configuració. A la classe `Setup.cpp`, en el mètode `Setup::read()` es crida a la Factory de les llums per a crear-les (`Model/Modelling/Lights/LightFactory`). Per ara, aquesta Factory només pot crear llums puntuals.

1. Modifica el fitxer `setupRenderOneSphere.json` per incloure una llum ambient global ambient a (0.1, 0.1, 0.1) i una llum puntual a l'escena a la posició (2, 8, 10), amb una $l_a = (0.3, 0.3, 0.3)$, una $l_d = (0.7, 0.7, 0.7)$, una $l_s = (1.0, 1.0, 1.0)$ i un coeficient d'atenuació de $0.5 + 0.01d^2$. Et pots "inspirar" en el fitxer `setupRenderSpheres.json` per veure el format d'una llum puntual.

2. Implementa Blinn-Phong per fases implementant una nova estratègia de *shading* anomenada [BlinnPhongShading](#) per calcular el color. Revisa els paràmetres necessaris per a poder passar tota la informació que necessita el mètode de Blinn-Phong (mira les transparències de teoria). Recorda que hauràs de modificar també la classe [ShadingFactory](#) per incloure aquesta nova estratègia.

Per calcular la il·luminació de Blinn-Phong, necessitaràs les llums i el materials de cada objecte. A la pràctica base, la classe [Material](#) té les components difosa, ambient, especular, el coeficient de reflexió especular o *shininess* i la seva opacitat. Si necessites més atributs en algun moment del desenvolupament, afegeix-los a la classe tot tenint en compte que cal que es llegeixin des fitxer json (mira l'apèndix d'aquest document per a saber-ne més).

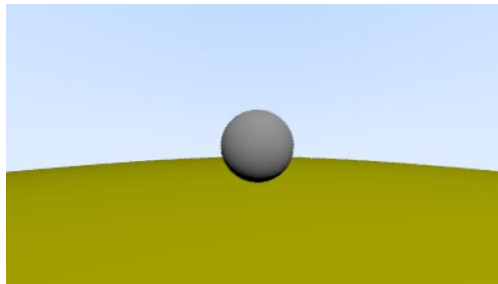
Usant el fitxer [twoSpheres.json](#), comprova pas a pas que el mètode de Blinn-Phong et dona els colors correctes, mirant cada component per separat, raonant a cada resultat per què et donen les visualitzacions corresponents, tenint en compte que les esferes tenen com a K_d el color definit abans, una $K_a = (0.2, 0.2, 0.2)$, una $K_s = (1.0, 1.0, 1.0)$ i una *shininess* de 10.0.

Afegeix els teus resultats pas a pas en el README del teu projecte:

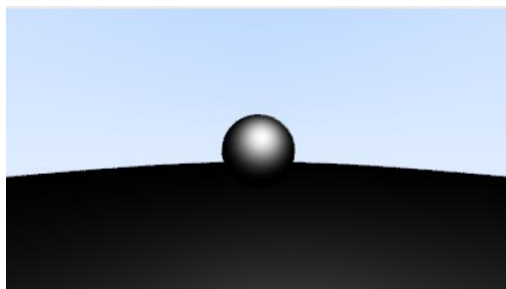
- a. Si només es calcula la component ambient, s'obté una imatge semblant a:



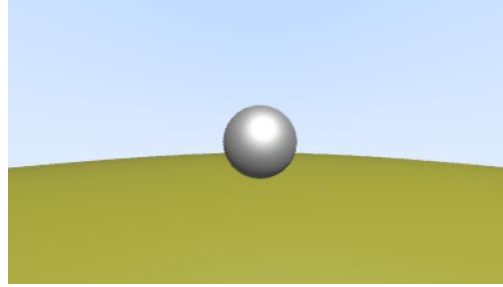
- b. Amb només la component difosa:



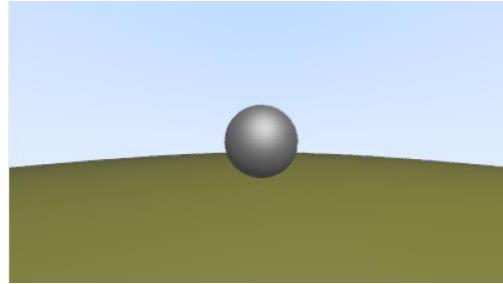
- c. Amb només l'especular: Fixa't que has d'ampliar els valors del material que es llegeixen del fitxer de dades.



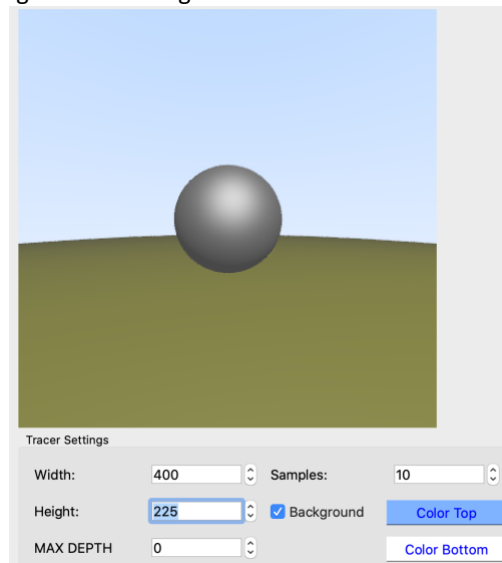
d. Ara les tres juntes:



e. I afegint atenuació amb profunditat:



f. I afegint l'ambient global



3. Implementa *Phong Shading* seguint les mateixes pautes del pas anterior. Fes captures de pantalles amb els resultats obtinguts. Què necessites canviar? Fes la captura final per a comparar-la amb el mètode de *Blinn-Phong Shading*.
4. Implementa *Cel Shading* per a aconseguir visualitzacions semblants a les dels dibuixos animats. Necessites afegir informació en el material? Fes captures de pantalles amb els resultats obtinguts.

PAS 3. Afegeix ombres

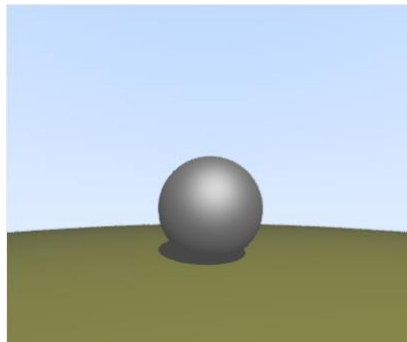
En aquest pas afegiràs ombres a tots els teus *shadings*.

1. Analitza el codi per saber com activar les ombres: Fixa't que en el codi tens implementat el *shading* alternatiu a [ColorShading](#), a la classe [ColorShadow.cc](#). Aquest mètode s'activa quan es

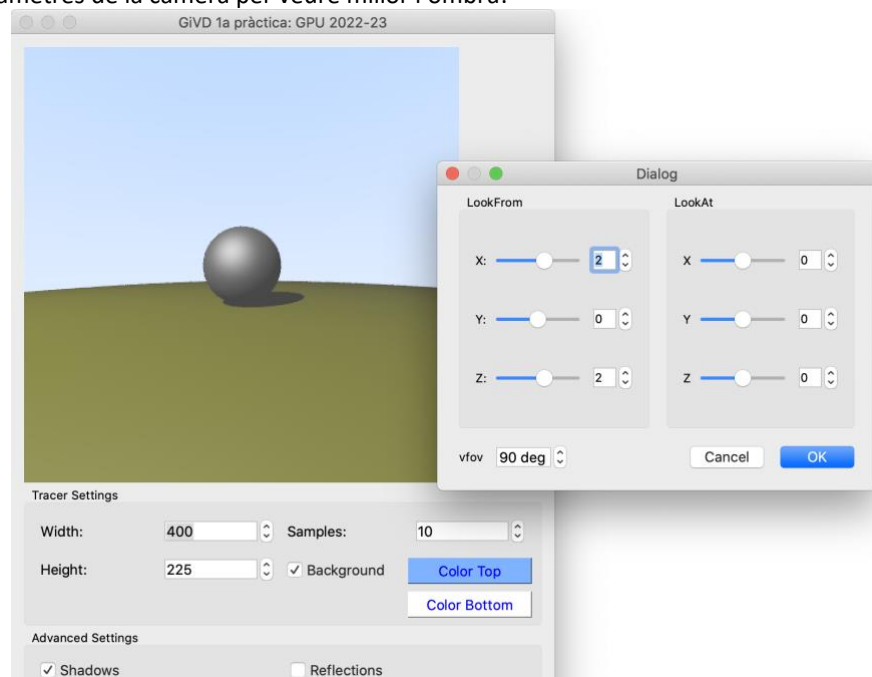
marca el flag de “shadow” de la interfície gràfica. A la classe [RayTracer](#), en inicialitzar el Raytracing, al mètode `init()`, es comprova el flag d’ombra i es canvia l’estratègia de *shading* a seguir (`ShadingFactory::switchShading(shared_ptr<ShadingStrategy> m, bool shadow)`). Per exemple, si es tenia el *shading* [ColorShading](#) activat, i es prem el flag de “shadow”, aquest mètode canviarà al *shading* [ColorShadow](#) i quan es desactivi, es tornarà a posar l’estratègia de [ColorShading](#). En aquest cas inicialment no es fa ombra, sinó que es posa el color complementari.

2. Implementa el raig d’ombra al mètode de Blinn-Phong: Per això, crea una nova estratègia de *shading* anomenada [BlinnPhongShadow](#). Després implementa el codi del mètode `computeShadow` de la classe [ShadingStrategy](#). Utilitza on creguis convenient, aquest nou mètode per a fer ombres en el teu Blinn-Phong.

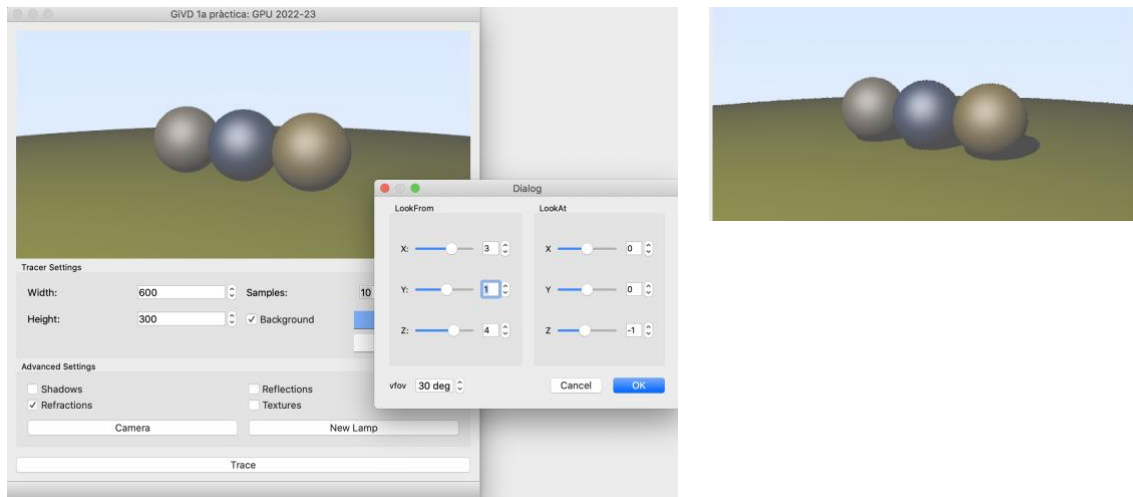
En el cas que hi hagi un objecte entre la llum i el punt on s’està calculant la il·luminació, quina component de la fórmula de Blinn-Phong s’haurà de tenir en compte?



Pots canviar paràmetres de la càmera per veure millor l’ombra:



5. Implementa els *shadings* alternatius d’ombres al *Color Shading*, al *Normal Shading*, al *Phong Shading* i al *Cel Shading*. Fes captures de pantalles amb els resultats obtinguts.
6. Obre el fitxer [spheres.json](#) que trobaràs al campus en el fitxer [datasetsVis.zip](#) i el seu setup (`setupRenderSpheres.json`). Vigila que no són iguals als que has clonat en el teu projecte. Si jugues amb la càmera, obtindràs una visualització com la següent.



7. Fes una escena més complexa, ja sigui directament des de codi, ja sigui a partir del fitxer de [spheres.json](#) on com a mínim hi hagin 10 objectes (ja siguin esferes, o capses, o .objs).

PAS 4. Afegeix recursió en el teu mètode `RayPixel` per a que tingui en compte els rajos reflectits.

1. Afegeix recursió en el raig per a que es segueixin els rajos secundaris. Per a controlar el fi de la recursivitat en el mètode `RayPixel` és necessita la profunditat actual o *depth* i un atribut del setup, `MAXDEPTH`, per a controlar el fi de la recursivitat dels rajos reflectits.

T'aniria bé també incloure en la finestra principal de la interfície un `QSpinBox` on la que puguis graduar el `MAXDEPTH`. Connecta'l per a que refresqui l'atribut del `setup`.



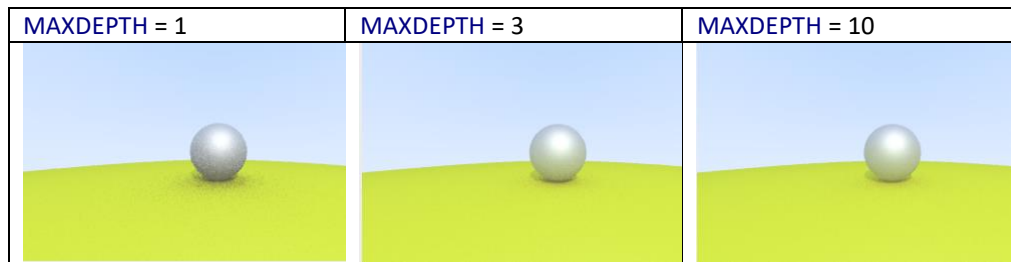
Per a calcular els rajos secundaris i el color que atenuarà la contribució d'aquell raig, a la classe `Material` tens definit el mètode `scatter()` que cal que sigui implementat per a cadascun dels diferents materials. A la pràctica bàsica tens aquest mètode ja implementat a la classe que representa el material `Lambertià`. Fixa't que ara retorna fals, però el valor de retorn d'aquest mètode, per a que es tingui en compte el raig secundari, cal que retorni cert.

Adequa el codi del mètode `RayPixel` per a que consideri rajos secundaris i controlï la recursivitat amb el valor `MAXDEPTH`.

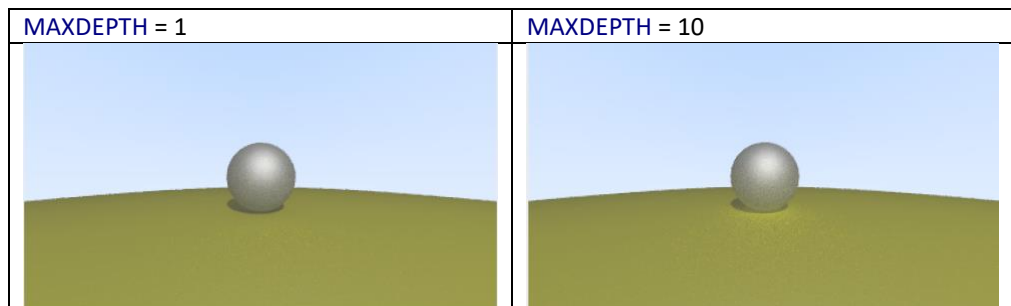
2. Testeja el mètode amb les dues esferes. Prova només amb el material. Si un raig té intersecció amb un objecte en un cert punt, el color es calcula com la suma del color de Blinn-Phong en el punt d'intersecció i el color calculat en la recursió ponderat pel color que retorna el mètode `scatter()`.

A les imatges es té una llum puntual situada al punt (2, 8, 10), amb una component ambient 0,3,0,3,0,3, una difusa de 0,7, 0,7, 0,7 i una especular de 1,0, 1,0, 1,0, amb una atenuació de $0.5 + 0.01d^2$, considerant que l'observador està situat en el punt (13, 2, 10) mirant cap el punt (0,0,0) i una intensitat ambient global a 0,1, 0,1, 0,1. Les imatges següents s'obtenen quan es considera que tot raig que no interseca amb l'escena pren el valor de *background* (sigui del raig primari o no).

Veuràs el material `Lambertià` dona objectes molt rugosos donada l'aleatorietat del raig reflectit.

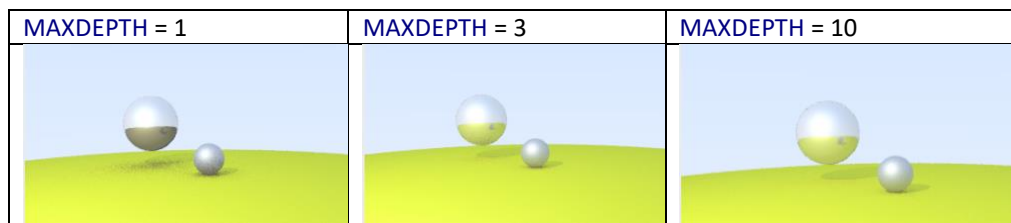


Els rajos secundaris que no intersequen amb l'escena cal que aportin la intensitat ambient global, i no el color de *background*, si és que es considera que el *background* no aporta res de llum a l'escena. En aquest cas obtindràs una imatge més fosca:



Pots jugar amb els paràmetres de la llum per a poder veure les imatges clares. Per exemple, pots pujar el valor de la difusa i baixar les de l'ambient, posar més llums, etc.

3. Fes una nova classe derivada de `Material` que codifiqui el tipus `Metal`. Defineix el mètode `scatter` per a que simuli la reflexió especular. Aquest mètode ha de calcular un únic raig reflectit i també retornarà el color (K_s) amb el que contribueix el color del raig reflectit al color final. Per a calcular el vector reflectit, podeu usar el mètode `glm::reflect`.
1. Fes una còpia del fitxer `twoSpheres.json` a `threeSpheres.json`. Afegeix a aquest nou fitxer una esfera metàl·lica a la posició $(-3, 1, 0)$, de radi 1.0 i amb un material metàl·lic de $K_d = (0.7, 0.6, 0.5)$, $K_a = (0.2, 0.2, 0.2)$, $K_s = (0.7, 0.7, 0.7)$ i una *shininess* de 10.0. Prova amb diferents nivells de recursivitat. La càmera utilitzada per les imatges de la taula col·loca l'observador al punt $(10, 2, 10)$ amb un vfov a 45 graus. Aquí considerem que els rajos secundaris que no intersequen, agafen el color del background.



4. Prova amb els altres tipus de *shading* per veure d'altres efectes en les reflexions.

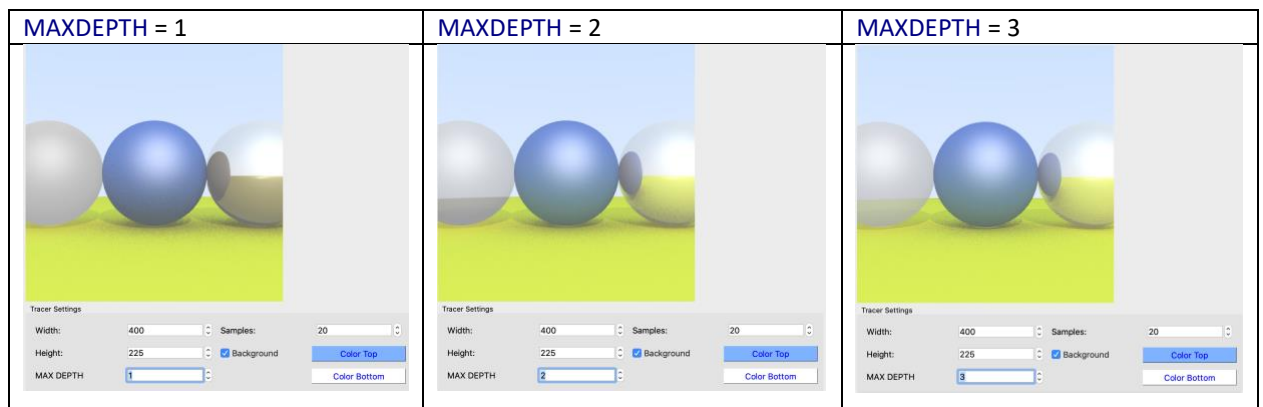
PAS 4. Afegeix recursió en el teu mètode `RayPixel` per a que tingui en compte els rajos dels objectes transparents

2. Afegeix recursió en el raig per a que es segueixin els rajos secundaris d'objectes transparents. Fes un nou `Material` anomenat `Transparent` (en el cas de `Windows`, `Transp`) que serveixi per fer transparències amb el seu propi mètode `scatter`. En aquest cas, ha de calcular un únic raig transmès i retornarà la K_t del material si el raig secundari és tramès o la K_s si el raig transmès és

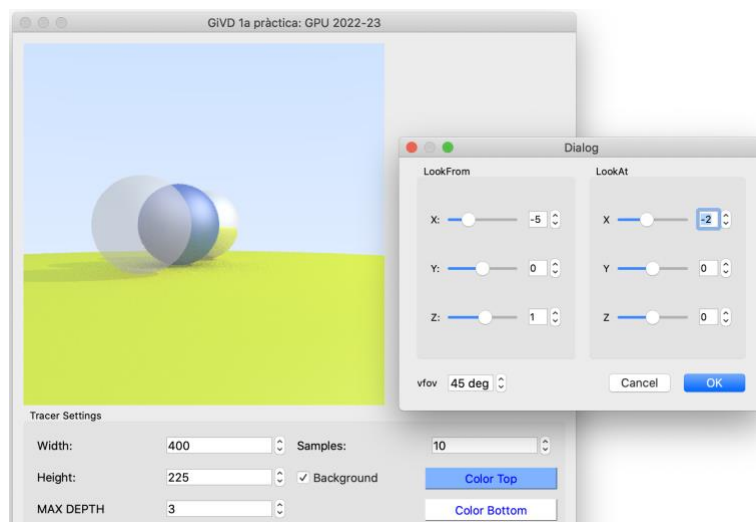
de reflexió interna. Recorda de matisar el color local obtingut per Blinn-Phong, o per l'estratègia de *shading* que estiguis usant, segons (1-kt) per així veure bé la transparència. Per a calcular el vector transmès, podeu usar el mètode `glm::refract`.

3. Tingues en compte que necessitaràs la `nu_t` per a definir el material transparent. En el fitxer `spheresMetalTransp.json` en trobaràs un exemple, tot i que ara el codi no està llegint aquesta `nu_t`. On hauries de llegir-la?

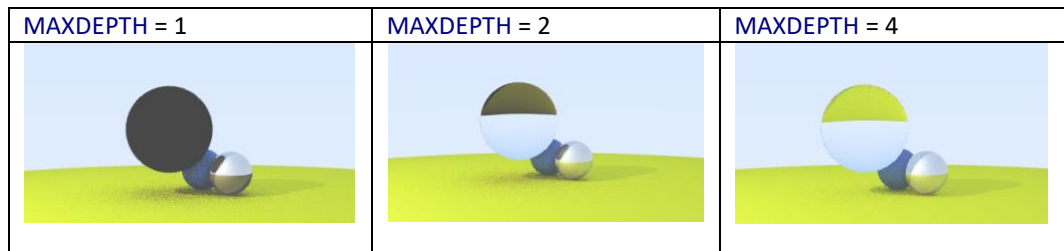
Les imatges obtingudes a continuació utilitzen el fitxer `spheresMetalTransp.json` i el `setupRenderTwoSpheres.json`, ajustant la ambient global a (0.3, 0.3, 0.3) i el `vfov` a 45 graus. Fixa't que l'índex de refracció (`nu_t`) de l'esfera transparent és 1.0. Fixa't que s'obtenen aquestes visualitzacions per què en cas de ser material transparent, es pondera el color de la il.luminació local (per exemple el color obtingut a cada pas amb Blinn-Phong) amb el factor (1-scatteredColor, on scatteredcolor és el color obtingut pel mètode scatter del material transparent)



O des d'un altre punt de vista i canviant lleugerament la "kd" de l'esfera transparent a [0.7,0.7,0.7].

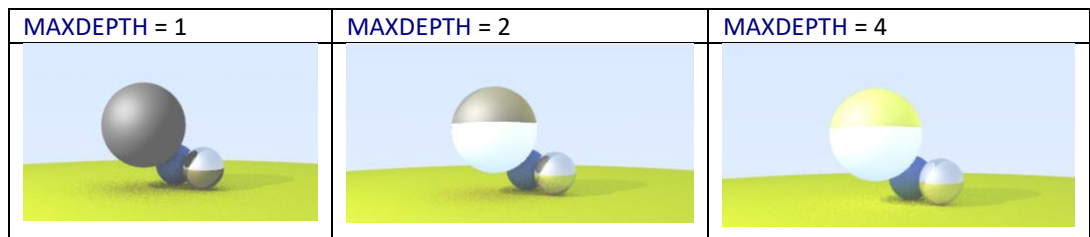


Les imatges obtingudes a continuació utilitzen el fitxer `fourSpheres.json` i el `setupRenderFourSpheres.json` que conté una esfera transparent en la posició (0.0, 1.0, 0.0) de radi 1.0 amb un índex de refracció de 1.5 i una $K_t = (1.0, 1.0, 1.0)$. Prova-ho amb diferents nivells de recursivitat. Per què si tens el `MAX_DEPTH` a 1, l'esfera no es veu transparent?



Si assignes el color ambient global enlloc del de *background* en els rajos secundaris que no intersequen amb res. Com et canvia la visualització? Raona el per què?

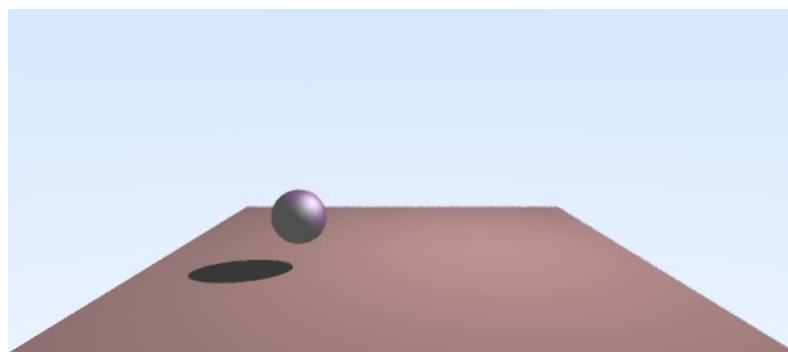
Si en canvi no ponderes el color local amb (1-colorScattered) pots obtenir visualitzacions com aquestes. Raona per què es veuen totes més clares que les anteriors.



PAS 5. Visualization mapping:

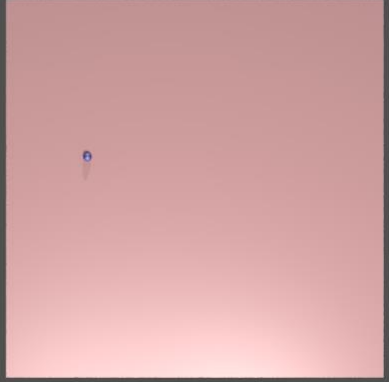
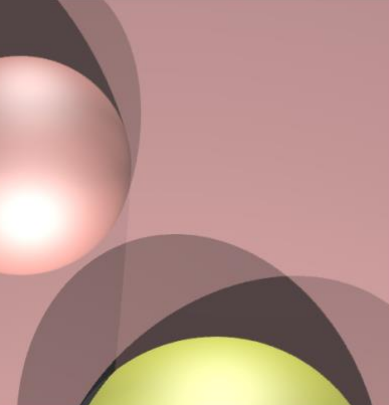
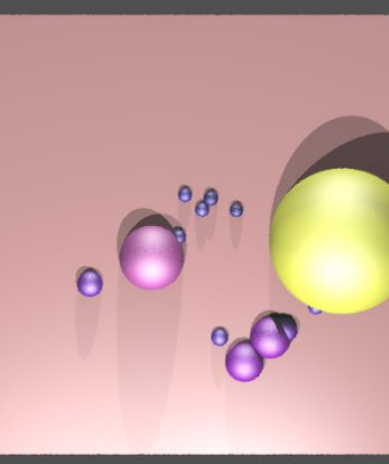
- Adapta ara la visualització per a mostrar dades des d'un fitxer. Usa el fitxer [data0.json](#) per a que es mapegi la única dada que hi ha en una esfera. Usa el fitxer [setupRenderData0.json](#) que trobaràs al campus en el fitxer [DatasetsVis.zip](#). Comprova que la transformació que has fet al pas 5 de la Fase 1 funciona correctament.

Utilitza la classe [FittedPlane](#) definida en el pas 5.b de la Fase 1, que permet definir un pla acotat per una caps 2D definida pel seu punt mínim i el seu punt màxim. Posa el pla $Y=-1$ com element de referència. Posa un valor de color difús en aquest pla afitat. A l'exemple, es considera un material Lambertian amb $K_a = (0.1, 0.1, 0.1)$, $K_d = (0.7, 0.4, 0.4)$ i $K_s = (0.0, 0.0, 0.0)$, *shininess* = 0 i s'ha situat el centre de l'esfera a $Y=0.0$ per veure millor l'efecte de l'ombra.

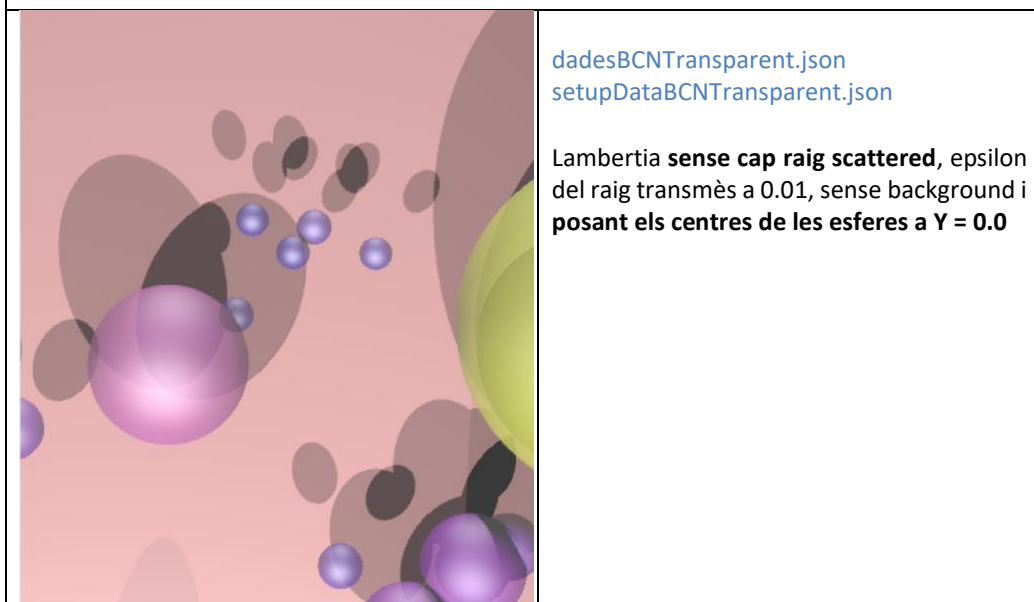
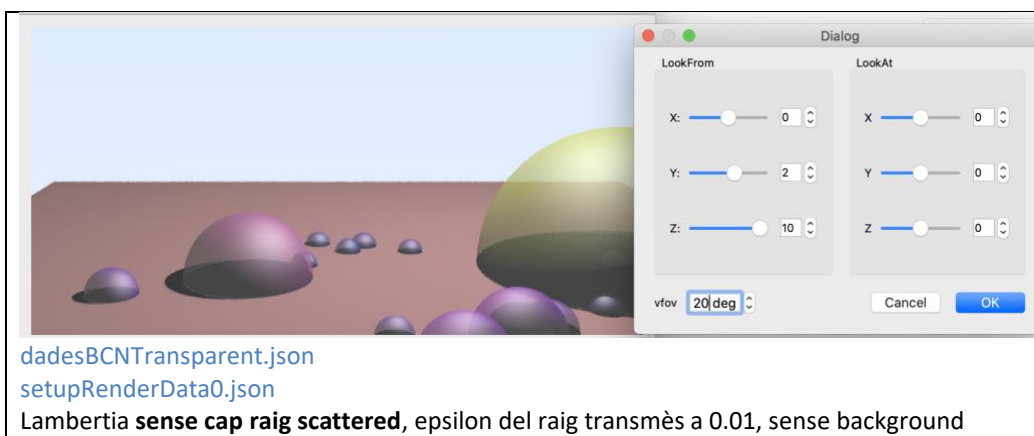
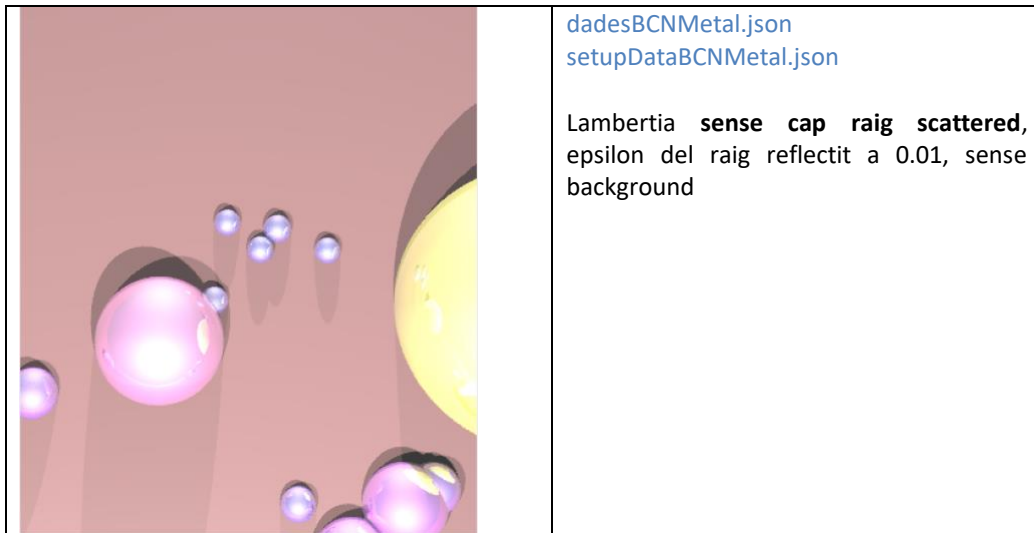


- Prova diferents visualitzacions de dades amb els diferents fitxers que trobaràs en el campus ([datasetVis.zip](#)). Vigila que les z 's de l'escena virtual corresponent a la latitud en els fitxers de dades geolocalitzades. Comprova que calcules correctament la transformació a fer, per què esperem que quan la latitud creixi, el valor de les z 's del món virtual decreixi.

Trobaràs en el campus més fitxers relatius a dades de Barcelona. Pots obtenir imatges com les següents (a continuació es donen les imatges amb els mapes al costat per fer-los servir de referència). Comença posicionant la única dada que hi ha al fitxer [dadesBCNOneValue.json](#) per veure que es col·loca bé l'esfera. Aquí estem usant MAXDEPTH = 0 com a nivell de recursivitat i el material Lambertia del pla no retorna cap raig scattered. Prova després amb [dadesBCN_Zoom.json](#) i [dadesBCN.json](#) per veure que les teves transformacions també segueixen funcionant bé.

		<p>dadesBCNOneValue.json setupDataBCNOneValue.json</p> <p>tenint en compte un valor mínim truncat a 0.1 a l'hora de calcular l'escala i multiplicant per 0.5 el rang màxim en x, y i z del món virtual</p>
		<p>dadesBCN_Zoom.json setupDataBCN_Zoom.json</p> <p>Tenint en compte un valor mínim truncat a 0.1 a l'hora de calcular l'escala de les Y's i multiplicant per 0.5 el rang màxim en x, y i z del món virtual</p>
		<p>dadesBCN.json setupDataBCN.json</p> <p>Tenint en compte un valor mínim truncat a 0.1 a l'hora de calcular l'escala de les Y's i multiplicant per 0.5 el rang màxim en x, y i z del món virtual</p>

3. Prova també les reflexions usant materials de Metall o Transparents, tal i com es mostra a les figures:



4. Pots consultar com fer paletes de colors a amb els materials i objectes que consideris, mostrant tant ombres com transparències . Prova a usar diferents gizmos. Lliura la imatge que has obtingut amb els paràmetres corresponents, juntament amb el lliurament de la pràctica. Prova a canviar la càmera al lookAt (0, 10, 10) amb un vup de (0, 1, 0) i prova amb diferents vfov's (20, 10).

EXTENSIONS OPCIONALS

- Permet realitzar ombres en qualsevol dels *shadings* que has implementat.
- Implementar diferents **gizmos** en el mapeig de dades (com BOX, MESH)
- Inclou altres tipus de llums, com llums direccionals i spot-lights. En aquesta fase heu considerat només llums puntuals. En el projecte base es proporciona la classe **Light** i el tipus de llum puntual (**PointLight**), amb la seva corresponent Factory (**LightFactory**). Podeu inicialitzar les llums en la constructora de l'escena inicialment per fer proves però després caldrà que les lleigiu del fitxer de configuració de la visualització (fitxers de tipus **setUp.json**). Les llums per tant es llegeixen des de la classe de lectura d'aquest tipus de fitxers (**SetUp**). Anima't a implementar una nova finestra a la GUI per a modificar les llums.
- Inclou penombres usant llums amb àrea o amb simplificacions que et permetin fer les penombres aproximades.
- En les transparències de teoria trobaràs més idees per a fer visualitzacions més realistes (*ambient occlusion*, etc.)
- En afegir materials transparents, pots calcular les ombres segons el color del material transparent que traspasa la llum.

Temps estimat de cada estudiant

4 hores presencials + 10-12 hores de treball setmanal individual fora de l'aula

Material de referència

- [1] <http://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/how-does-it-work>- Síntesi de l'algorisme de RayTracing.
- [2] https://www.flipcode.com/archives/reflection_transmission.pdf Breu article on explica com es dedueixen els càlculs dels rajos reflectits, els rajos transmesos i el cas de reflexió total interna.
- [3] <https://towardsdatascience.com/viz-palette-for-data-visualization-color-8e678d996077> Per a construir noves paletes
- [4] <https://colorbrewer2.org/-type=sequential&scheme=BuGn&n=9> Per a construir noves paletes

APÈNDIX:

COM ES CARREGUEN ELS MATERIALS

Quan es llegeix un objecte d'un fitxer d'una escena virtual, es carreguen les dades des del mètode **read()** de la classe **SceneFactoryVirtual**. Allà es creen els objectes usant el mètode **read()** de la classe **Object**, que es des d'on es delegarà la lectura de material i se'n fa la seva creació usant la Factory de materials (**MaterialFactory**). Quan facis un nou tipus de material, haurà de modificar aquesta classe per afegir-hi la seva creació. En el codi bàsic, els objectes de l'escena virtual només té els valors de material **Lambertian**. Realitza les primeres proves amb aquest tipus de material.

Durant aquesta fase caldrà que completis la classe **Material** amb propietats addicionals, si cal, que es proporciona en el codi base. A més a més, afegiràs diferents tipus de materials perquè en el codi bàsic només es dona la implementació del material de tipus **Lambertian**.

A l'inici de la pràctica la classe *Material* té les components difosa, ambient, especular, el coeficient de reflexió especular o *shininess* i la seva opacitat. Si necessites més atributs en algun moment del desenvolupament, afegeix-los a la classe tot tenint en compte que cal que es llegeixin des fitxer json, ja siguin des de la classe material, o des de la nova classe que implementa el nou material.

DES DE LES DADES DE MÓN REAL COM ES FA EL MAPEIG A MATERIALS ? ÚS DE LES PALETES (o COLORMAPS)

A la classe *SceneFactoryData*, trobaràs la crida *materialMaps()* que s'usa des del mètode *visualMaps()*. Allà es crea un material que té la component difosa corresponent al color que li toca de la paleta definida en el mateix fitxer de dades. Fixa't que les paletes definides en la classe *ColorMapStatic* tenen 256 colors i els teus valors de propietats poden anar entre un valor mínim i un valor màxim. En el mètode *mapeigMaterial* s'està fent la conversió del valor de les dades a l'índex corresponent.