Pràctica 1
Raytracing: Fase 1

GiVD - curs 2022-23

FASE 1: Nous tipus d'objectes i la seva visualització bàsica

ÍNDEX

1. INTRODUCCIÓ

Al final d'aquesta fase hauràs construït la teva escena virtual amb objectes de tipus diferents que provindran directament de fitxers d'escenes virtuals (DATA_TYPES::VIRTUALWORLD) o bé de fitxers de dades (DATA_TYPES::REALDATA).

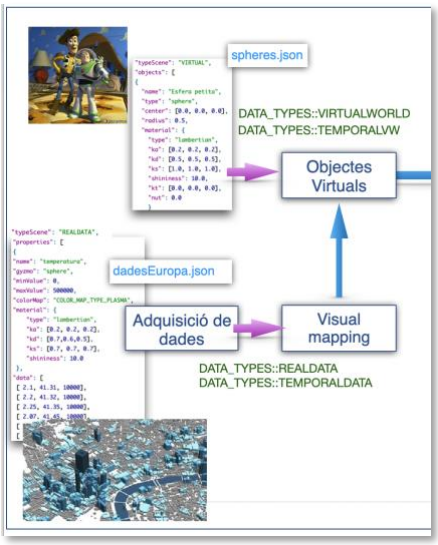
Els objectes continguts a la teva escena a visualitzar (classe Scene) seran objectes paramètrics - esferes, plans, capses, triangles i malles de triangles (o Mesh) -. A la pràctica base les classes que representen esferes (Sphere) i plans (Plane) ja estan implementades. La classe que representa malles de triangles (Mesh) està parcialment implementada, ja que permet la construcció d'una malla a partir d'un fitxer .obj però en canvi no implementa el mètode hit.

A la pràctica base també es dona el codi per carregar escenes des de fitxers .json. A la Figura 1 tens un resum del seu format en ambdós casos:

- DATA_TYPES::VIRTUALWORLD: En el cas que les dades provinguin d'un programa de construcció de mons virtuals com blender o 3DStudioMax, es suposarà que l'escena està ben formada i els objectes ja estan posicionats correctament en l'escena virtual (veure fitxer spheres.json de la figura 1).
- DATA_TYPES::REALDATA En el cas que les dades provinguin de mesures geolocalitzades, el fitxer json conté la informació de com es volen mapejar aquestes dades al món virtual (veure fitxer data0.json de la figura 1) .

Teniu diferents exemples dels fitxers .json en el projecte base. S'ha optat pel format json donat que és molt descriptiu i es pot entendre bé el significat de cada dada.

En el codi, trobareu que cada classe que permet crear les seves dades des de fitxer implementa la classe Serializable que s'encarrega de llegir la part del fitxer corresponent a les seves dades. Per exemple, la classe Sphere deriva de la interfície Serializable implementant els mètodes de read, write i print. Al codi base ja estan implementats aquests mètodes en els objectes de tipus esfera, pla i malla triangular.




```
{
"scene": "OneSphere",
"typeScene": "VIRTUAL",
"objects": [
{
  "name": "Esfera",
  "type": "sphere",
  "center": [0.0, 0.0, 0.0],
  "radius": 0.75,
  "material": {
    "type": "lambertian",
    "ka": [0.2, 0.2, 0.2],
    "kd": [0.8, 0.4, 1.0],
    "ks": [1.0, 1.0, 1.0],
    "shininess": 10.0,
    "kt": [0.0, 0.0, 0.0],
    "nut": 0.0,
    "opacity": 1.0
    }
}
]
}
```

Exemple d'escena virtual: oneSphere.json

```
{
"scene": "Data0",
"typeScene": "REALDATA",
"Real": {
  "minPoint": [-5, -5],
  "maxPoint": [5, 5]
},
"Virtual": {
  "minPoint": [-5,  0, -5],
  "maxPoint": [ 5,  2.0,  5]
},
"attributes": [
{
"name": "temperatura",
"gyzmo": "sphere",
"minValue": 0,
"maxValue": 2,
"colorMap": "COLOR_MAP_TYPE_INFERNO",
"material": {
    "type": "lambertian",
    "ka": [0.2, 0.2, 0.2],
    "kd": [0.7,0.6,0.5],
    "ks": [0.7, 0.7, 0.7],
    "shininess": 10.0
},
"data": [
[ -2, -1, 0.5]
]}]}
```

Exemple d'una escena de dades: data0.json

Figura 1. Exemples de fitxers d'entrada .json

2. PASSOS A SEGUIR

Les activitats es poden repartir en dues parts diferenciades:

1.1. les que es refereixen al tractament de nous tipus d'objectes i les seves interseccions a partir d'una escena VIRTUALWORLD

1.2. les que creen l'escena corresponent a les dades geolocalitzades a partir d'un fitxer de dades (REALDATA).

2.1. INCLOURE NOUS TIPUS D'OBJECTES: VIRTUALWORLD

En aquests passos crearàs nous objectes i les seves interseccions.

Et proposem que quan vagis a implementar cada intersecció amb un objecte nou, primer implementis i testegis el mètode hit comprovant que retorna el punt d'intersecció del raig amb l'objecte més proper a l'origen del raig amb un objecte creat usant un menú similar al menú File->New Sphere de la interfície. Després, la idea és que el llegeixis des del fitxer .json on es codifica l'escena virtual.

Per a calcular la intersecció del raig amb qualsevol d'aquest tipus d'objectes, heu d'implementar els mètodes en les classes corresponents sobrecarregant el mètode hit de la classe Object (derivada de la classe Hitable). Els mètodes hit de la classe Sphere i de la classe Plane ja estan implementats.

Les dades que són ja de per sí mons virtuals estan contingudes en fitxers de dades (per exemple oneSphere.json de la carpeta resources). En aquest cas, el fitxer pot contenir diferents tipus d'objectes superficials, com has vist a la fase 0. En el codi inicial només es llegien objectes de tipus esfera (mira el mètode void SceneFactoryVirtual::*read*(const QJsonObject &json). Fixa't que delega la lectura de cada

objecte a la classe corresponent. Així, quan afegeixis un nou tipus d'objecte, tingues present d'implementar el mètode *read* per tal de llegir les seves dades de fitxer.

En general, en els fitxers de dades d'objectes virtuals es poden incloure tants objectes com es vulguin, dels tipus que es vulgui. És a dir, en un mateix fitxer es poden incloure esferes, plans, capses 3D, malles de triangles i cilindres.

PAS 1: Box

1.a. Creació d'una nova classe Box per definir una capsa 3D definida pels seus vèrtexs extrems, suposant que la capsa estarà alineada als eixos coordinats. Pots agafar de mostra la classe Sphere per guiar la teva implementació. Com la classe Box derivarà de la classe Object, implementa-hi el seu mètode hit.

1.b. Modifica la classe ObjectFactory per a crear capses amb les seves cares paral·leles als eixos coordenats.

1.c. Fes el nou Menú a File-> New Box que creï una escena virtual amb només una capsa. Mira com es connecten els menús des de la classe MainWindow a la classe Builder. Testeja el test d'intersecció implementat usant el Raytracing.

Per veure si les vostres interseccions funcionen correctament, en el mètode RayPixel de la classe RayTracer useu el mètode de ColorShading que retorni un color concret quan es trobi que el raig interseca amb un objecte, com fèieu en la Fase 0. També podeu usar la t de la intersecció trobada per veure si seguiu fent correctament l'ordenació de les interseccions que heu programat a la Fase 0.

1.d. Un cop funcionin les interseccions, implementar el mètode read a la classe Box per a que es llegeixi des del fitxer json. En aquest cas es vol que una capsa es llegeixi com:

```
"name": "capsa",
"type":  "BOX",
 "punt_min": [-1, -1, -1],
 "punt_max": [1, 1, 1]
```

Per fer les proves de lectura copia el fitxer spheres.json en un nou fitxer box.json per a definir-te un fitxer que permeti carregar capses.

PAS 2: Triangle

2.a. Segueix els mateixos passos que has fet en la creació de la Box. En aquest cas, fes una nova classe *Triangle* que defineixi un triangle a partir de tres vèrtexs. La normal associada al triangle és la del pla que conté els tres punts, amb la direcció resultant de l'ordenació dels tres punts en sentit antihorari. Implementa el seu mètode hit. Modifica la classe ObjectFactory per a crear triangles. Crida la construcció de l'escena des d'un nou menú File-> New Triangle de la interfície i prova el hit que has implementat.

Per a la intersecció raig-triangle podeu basar-vos en el mètode de càlcul d'intersecció descrits a [1], secció 10.3, i en el codi ja donat de intersecció raig-pla.

2.b. Copia el fitxer spheres.json en un nou fitxer triangles.json per a definir-te un fitxer que permeti carregar triangles, implementant el mètode read a la classe Triangle. Fixa't que ara actualment no està contemplada la creació de triangles ni a la lectura del fitxer ni a l'escena.

El format d'un triangle en el .json serà: identificat per tres punts. Per exemple,

```
"name": "un triangle",
"type": "TRIANGLE",
"p1": [0.0, 0.0, 0.0],
"p2": [1.0, 0.0, 0.0],
```

```
"p3": [0.5, 1.0, 0.0]
```

PAS 3: Mesh

Una Mesh o malla triangular és un objecte format només per la superfície de l'objecte representada com a conjunt de cares. Aquesta classe Mesh utilitza la classe Face per a indexar els vèrtexs. Aquests models es codifiquen usant la representació indexada de vèrtexs en el format .obj. El codi per llegir el format .obj es proporciona a la pràctica base (mira la constructora de la classe Mesh.cpp que crida al mètode *load* amb el nom de fitxer .obj que vulguis carregar). Les malles triangulars tenen el següent format .json dins d'una escena:

```
"name": "Br Objecte",
"type": "MESH",
"objFileName": "://resources/cube.obj"
```

Fixa't que el fitxer cube.obj cal que es trobi als recursos del projecte per a poder-se llegir de forma relativa. No només cal tenir-lo a la carpeta resources sinó que també cal tenir-lo donat d'alta al resources.qrc del projecte de QtCreator.

3.a. Implementa el mètode hit que permeti calcular-ne la seva intersecció amb el raig. Fixa't que considerem només malles de triangles i per tant, pots usar la intersecció amb triangles que has fet en el punt anterior. Recordeu que en el cas de trobar varies interseccions amb l'objecte, es té en compte sempre la intersecció més propera a l'observador.

Modifica la classe ObjectFactory per a carregar i crear malles poligonals en forma de Mesh. En aquest cas, el read a la classe Mesh ja està implementat. Quan penses que és millor crear els triangles de la malla?

3.b. Per tal d'accelerar el càlcul de les interseccions amb les malles triangulars, es vol crear la seva capsa contenidora i en fer el hit del raig, es vol comprovar primer si interseca amb la capsa que conté la malla (*bounding box*) o amb l'esfera que conté la malla (*bounding sphere*). Si ja no fa hit amb aquests objectes, es pot descartar la intersecció. Implementa les dues possibilitats, la de la *bounding box* i la de la *bounding sphere*. Quina creus que teòricament és millor? Com funcionen en la realitat? Prova a carregar alguns dels fitxers .obj que trobaràs en el campus amb malles més complicades, per veure amb quina estratègia et va més ràpid l'algorisme de raytracing. Pots imprimir per consola la capsa contenidora per veure on situar la càmera.

PAS 4: **[OPCIONAL]** Cylinder

4.a. Fes una nova classe Cylinder amb el mètode hit. El cilindre és un objecte paramètric identificat pel seu centre de la base, el radi, l'alçada i la direcció del seu eix. Pots suposar que per defecte el seu eix sempre serà el de les Y's positives.

Modifica la classe ObjectFactory per a carregar i crear cilindres. Recorda implementar el mètode read a la classe Cylinder per a que es llegeixi des del fitxer json seguint el següent format:

```
"name": "un cilindre",
"type": "CYLINDER",
"radious": 2.0,
"h": 4.0,
"center": [0, 0, 0],
"axis": [0, 1, 0]
```

2.2. INCLOURE DADES GEOLOCALITZADES: REALDATA

PAS 5: Creació de l'escena a partir de dades geolocalitzades

5.a. **Explora el codi**: Mira la classe SceneFactoryData. Fixa't que en el mètode createScene(), en el mètode load() es llegeixen totes les dades i al final es crida al mètode visualMaps(). Dins de mètode load(), es crida al mètode read() i allà cal crear l'objecte base on es col·locaran tots els objectes. El mètode visualMaps() serà l'encarregat de crear un objecte per cada valor de les dades i li assignarà un material. Mira les explicacions de l'Apèndix d'aquest document per a veure el format d'un fitxer de tipus REALDATA.

5.b. Crea una classe nova (FittedPlane) que codifiqui un pla afitat com a base de referència o "terra" per situar tots els objectes. Aquest pla es situarà en el món virtual acotat en els límits del món virtual definits en el fitxer. Primer prova a llegir aquest pla i instanciar-lo a l'escena des de la classe SceneFactoryData. Fixa't en el mètode read. Després, defineix el mètode hit a la nova classe FittedPlane per tenir en compte la seva intersecció del raig dins del límits del món virtual. Ajuda't del hit de la classe Plane. El format del pla afitat del fitxer .json és el següent:

```
"base" : {
  "type": "FITTEDPLANE",
  "normal": [0.0, 1.0, 0.0],
  "point": [0.0, 0.5, 0.0],
  "pmin": [-5.0, -5.0],
  "pmax": [5.0, 5.0],
  "material": {
     "type": "MaterialTextura",
    "ka": [0.2, 0.2, 0.2],
    "kd": [0.7,0.6,0.5],
    "ks": [0.7, 0.7, 0.7],
    "shininess": 10.0,
    "textureFile": "://resources/map.png"
  }},
```

Modifica el mètode hit de la classe Scene per a que tingui en compte el pla base o el "terra". Testeja el teu codi visualitzant el pla terra de l'escena quan es carregui un fitxer .json de tipus REALDATA.

En general deixarem el pla base amb la normal (0, 1,0), però també el "terra" dels objectes podria ser una esfera i llavors es definiria amb els paràmetres propis de l'esfera (centre i radi).

5.c. Calcula les **transformacions geomètriques** que s'han d'aplicar a tot objecte per a situar-lo en el món virtual a escala (ajuda't de l'exercici que has fet a la Fase 6 de la Pràctica 0). Implementa el codi necessari per a crear un objecte per cada dada en la seva posició correcta i a escala al mètode objectMaps() de la classe SceneFactoryData seguint les indicacions següents:

- A cada dada, instancia un objecte geomètric unitari corresponent al seu "gyzmo" centrat en el punt (0, 0, 0) de món i a escala 1.
- Escala i trasllada l'objecte de forma correcta. Fixa't que a la classe Objecte es defineix el mètode abstracte aplicaTG que ha de ser definit a cada objecte. Mira la seva implementació inicial a la classe Sphere. Com calcularàs el centre del teu objecte en l'escena virtual? Com calcularàs la seva escala?
- Comença fent "gyzmos" de tipus Sphere. En aquest cas, cal situar el seus centres sempre en el pla Y = Y_plaBase. Comprova que es carreguen bé les dades posant diferents línies de dades en el fitxer json o agafant el fitxer dadesBCN.json on hi ha una propietat definida i varies línies de dades. Per ara només has de veure esferes.
- Implementa el tipus Box com a "gyzmo". Escala només les Y's per tenir ben representades les dades. Deixa les escales X i Z a valors constants. Recorda implementar el seu mètode aplicaTG.

5.d. Fes proves amb el fitxer data10.json. Situeu inicialment la càmera a la posició (13,2,3) i apunta al VRP (0,0,0) amb un vector de verticalitat (0,1,0).

5.e. **[OPCIONAL]**: Definir cilindres com a “gymos”. En relació als cilindres, es suposarà que tot cilindre està definit vertical en l'eix Y. La seva base estarà centrada en el pla Y=Y_plaBase i el seu radi serà fixe (pots escollir tu el valor). La seva alçada serà funció del valor de la dada que representi. Utilitza el fitxer data0.json per a situar i escalar el teu cilindre i comprovar el teu nou mètode.

EXTENSIONS OPCIONALS

- Prova a posar dues propietats en el fitxer amb dos *gizmos* diferents.
- Prova a calcular interseccions del raig amb objectes distribuïts en un arbre CSG. Així podràs fer operacions d’intersecció, negació, unió entre objectes. Només cal que guardis tots els *hits* del raig i després els processis segons l’arbre CSG.
- Afegir nous objectes paramètrics com torus o d’altres superfícies implícites.
- Mapejar les dades en una esfera i no en un pla.

APÈNDIX: FORMAT DELS FITXERS DE DADES GEOLOCALITZADES REALDATA ("typeScene": "REALDATA")

Els fitxers que tenen dades geolocalitzades contenen un conjunt de propietats a representar. Mira l'exemple data10.json del projecte. A continuació explorem diferents camps que el composen.

En el fitxer es dóna la informació de com fer el mapeig les dades en el món virtual (*Visual mapping*). Les dades "Real" de mapeig corresponen a les dimensions del món real (en x-z) i les dades "Virtual" són les mesures on es volen portar les dades a l'escena virtual. Així doncs, es definirà:

```
"Real": {
  "minPoint": [2.0259, 41.3013],
  "maxPoint": [2.2955,  41.4903]
},
"Virtual": {
  "minPoint": [-10,  0.5, -10],
  "maxPoint": [ 10,  0.5,  10]
},
```

En relació a les dades geolocalitzades procedents del món real, es defineixen els límits en x i en z, minPoint i maxPoint sempre suposant que els objectes que representaran les dades es situaran en un pla perpendicular a l'eix Y.
El sistema de referència virtual definit per minPoint i maxPoint defineix els límits de l'escena on s'han de mapejar totes les dades procedents dels fitxers de dades.

Per evitar que els objectes o gizmos que corresponen a les dades no surtin flotant a l'escena virtual, s'ha pensat en posar un pla base on es situaran els gizmos. Aquest pla base bé definit per:

```
"base" : {
  "type": "FITTEDPLANE",
  "normal": [0.0, 1.0, 0.0],
  "point": [0.0, 0.5, 0.0],
  "pmin": [-5.0, -5.0],
  "pmax": [5.0, 5.0],
  "material": {
     "type": "MaterialTextura",
     "ka": [0.2, 0.2, 0.2],
     "kd": [0.7,0.6,0.5],
     "ks": [0.7, 0.7, 0.7],
     "shininess": 10.0,
     "textureFile": "://resources/map.png"
   }},
```

Per a cada propietat o atribut que representi en el fitxer de dades, es definirà el tipus d'objecte que representarà els valors de la propietat (gizmo), els valors mínims i màxims de les dades mostrejades, la paleta de colors que es farà servir per representar els diferents valors i el material base dels gizmos. En el següent exemple es mostra l'atribut "temperatura".

```
"attributes": [
{
        "name": "temperatura",
        "gyzmo": "sphere",
        "minValue": 0,
        "maxValue": 40,
        "colorMap": "COLOR_MAP_TYPE_PLASMA",
        "material": {
           "type": "lambertian",
           "ka": [0.2, 0.2, 0.2],
           "kd": [0.7,0.6,0.5],
```

```
                "ks": [0.7, 0.7, 0.7],
                "shininess": 10.0
            }
        }
    ]
```

A més a més a cada atribut, es defineix el conjunt de valors mostrejats a la llista anomenada "data". A cada element llista es defineix terna de <latitud, longitud i valor> de la dada mostrejada. Es suposa que aquestes mesures sempre són escalars.

```
    "data": [
            [ 2.1, 41.31, 23],
            [ 2.2, 41.32, 24],
            [ 2.25, 41.35, 23],
            [ 2.07, 41.45, 25],
            [ 2.15, 41.4, 26],
            [ 2.20, 41.35, 23.1],
            [ 2.29, 41.37, 24],
            [ 2.08, 41.43, 26],
            [ 2.17, 41.42, 30],
            [ 2.12, 41.47, 31]
           ]
```

Sent, per exemple, en el primer punt, els dos primers valors, 2.1, 41.31 són la latitud i la longitud del punt mesurat on s'han captat els 23 graus de temperatura (darrer valor del primer punt). Si es volgués mesurar la humitat, es definiria una altra propietat o atribut en la llista d'attributs amb el seu nom, tipus de gyzmo, el valor mínim i màxim mostrejat, la seu paleta, material i dades mostrejades. I així, per exemple, per a la temperatura es podrien crear esferes i per a la humitat cilindres.

A la classe SceneFactoryData es llegeixen aquests tipus de fitxers i és el mètode SceneFactoryData::objectMaps() l'encarregat de construir l'escena virtual a partir de les dades i la informació de mapeig. Vigila per què aquest mètode està parcialment implementat.

Fixa't que partir de les dades de *visual mapping* i la primitiva geomètrica bàsica (esfera, box, cilindre o *Mesh* definida en la línia etiquetada per gizmo) que representa la propietat a visualitzar, a cada element de la llista de data (x1, z1, valor1), s'haurà d'instanciar l'objecte corresponent, escalat i posicionat a l'escena virtual. Es situarà a l'espai segons la latitud i la longitud mapejades en el món virtual i s'escalarà en Y segons la magnitud del valor mesurat proporcional a la mida de les Y's del món virtual. El seu color difós serà el que correspongui al valor de la dada seguint la paleta de colors definida en el fitxer de *mapping*. El valor de cada dada representarà un color que es podrà extreure de la paleta seleccionada d'entre les paletes definides a la classe ColorMapStatic (veure el mètode SceneFactoryData::materialMaps()).

Temps individual estimat: 3.5 hores presencials + 8-10 hores de feina fora de classe

Material de referència

[1] Capítol de llibre que explica els bàsics de RayTracing i com es deriven alguns test d'intersecció.
[2] Transparències que expliquen com deduir els tests d'intersecció amb cilindres:
https://mrl.nyu.edu/~dzorin/rend05/lecture2.pdf