TP de Visualisation (M2 I3D CMI IIRVIJ)
RayTracing part I

Installation:
Unzip TP_RT.zip
Créer un répertoire de compilation et utiliser cmake pour compiler
L'exécutable se trouve dans le repertoire RaytracingGPU et s'appelle RaytracingGPU

Code fourni:
L'affichage, les BVH et toutes les fonctions d'intersections sont fournies. Le programme lance un rayon depuis la caméra à travers chaque pixel de l'écran et exécute la fonction:

vec3 raytrace(in vec3 Dir, in vec3 Orig)

qui doit calculer la couleur finale du pixel à afficher.
Dir et Orig définissant la direction et l'origine du rayon

Le code de raytrace doit être écrit (**en GLSL**) dans le fichier TP_RT/tp/raytrace_tp1.frag
Le nom du fichier est modifiable au début du fichier RaytracingGPU/rtgpu_bvh.cpp

Les fonctions disponibles utilisable dans le code du shader (en plus des fonctions de GLSL) :

void just_hit_bvh(in vec3 O, in vec3 D)
Effectue l'intersection BVH Rayon(O: origine, D: direction) ne permet pas de récupérer les infos de l'intersection

void traverse_all_bvh(in vec3 O, in vec3 D)
Effectue l'intersection BVH Rayon(O: origine, D: direction) permet de récupérer les infos de l'intersection en appelant les 3 fonctions ci-après

bool hit()
Renvoit vrai si le rayon a touché un objet.

vec4 intersection_color_info()
Récupère les infos de couleur de l'intersection (R G B A)

vec4 intersection_mat_info()
Récupère les infos de materiau de l'intersection: (shininess, roughness, emissivity, area emissive)

void intersection_info(out vec3 N, out vec3 Pg)
Récupère les infos géométrique de l'intersection (N normale à la surface, Pg: position du point de l'intersection)

Doc GLSL: https://www.shaderific.com/glsl-functions

Dans le code C++: fichier RaytracingGPU/rtgpu_bvh.cpp

Comment ajouter un widget à l'interface et faire passer la valeur au shader ?

- Ajouter la variable dans la classe *RTViewer* (ne pas oublier de l'initialiser dans le constructeur)
- Ajouter le widget imGUI dans *void RTViewer::interface_ogl()*
- Ajouter l'uniform au début du shader en utilisant un indice de "location" libre >20
- Ajouter l'appel à set_uniform_value(loc,val) dans *void RTViewer::draw_ogl()* ligne 311

Comment modifier la scène: RTViewer::scene1()
bvh_gpu_scene_ contient les méthodes: add_sphere, add_cube, add_cylinder, add_orientedQuad.
Les paramètres sont une matrice de transformation 4x4 et un *Material*.

A faire:

1. Afficher si le rayon touche un objet de la scène (Rouge) ou pas (VERT)

2. Afficher la couleur

3. Afficher la couleur pondérée par le coefficient lambertien (N.L)
   Passer la position de la lumière en uniform (*set_uniform_value(21, GLVec3(10,10,100))*
   On pourra déplacer la lumière avec l'interface (1/2/3 sliders)

4. Afficher un rendu de phong. utiliser roughness [0,1] pour déterminer l'exposant du speculaire

5. Ajouter les reflets (indice utiliser la fonction GLSL reflect, comme pour le speculaire de phong)

6. Ajouter les ombres (lancer un rayon vers la lumière)

7. Mettre plusieurs lumières.

8. Faire une version phong + transparence
   On utilisera la fonction GLSL refract avec un ratio de 1.05 (modifiable)

9. Comment faire pour afficher les objets transparents à travers un objet transparent ?
   Problème pas de récursivité possible en GLSL !
   Indice: Fausse récursivité codée en "dur"

10. Mixer transparence, et reflets

11. Utiliser Fresnel (lois de Snell-Descartes) pour améliorer le rendu