Evaluation Graphismes interactifs

ESIEA 4A – 2025/2026

Projet pour validation du module

Le projet consiste à reprendre le code de base fourni et complété par vos soins pendant le module, code portant sur la réalisation d'un « raytracer/pathtracer » basique.

A implémenter

Le projet consistera à ajouter des fonctionnalités à ce code, une imposée (notée [OBLIGATOIRE]), et 2 autres au moins – notées [OPTIONNELLE] - à sélectionner parmi une liste fournie (2 au minimum pour valider le module, mais vous pouvez, si vous le souhaitez, en implémenter plus). La difficulté des fonctionnalités est indiquée par des étoiles (de [*] facile à [***] difficile).

NB : La difficulté sera prise en compte lors de la notation finale.

[OBLIGATOIRE] Ajout d'**objets 3D complexes** (formés de triangles) en utilisant et en adaptant la classe CMesh développée dans le code de base pour réaliser un rendu 3D temps réel. La classe une fois adaptée devra dériver de Entity3D et Collider (à la manière de la classe Sphere). En tant que Collider il faudra donc implémenter une méthode raycast (qui bouclera sur tous les triangles et testera la collision rayon/triangle en vous basant sur un des algorithmes que vous pourrez trouver à cette adresse : https://www.realtimerendering.com/intersections.html).

NB : aucune optimisation n'est demandée (il faut « juste » que le raytracer fonctionne même si le rendu est lent)

[OPTIONNELLE][*] Ajout de **Glow** post-process. Le principe est exposé dans la page suivante : https://learnopengl.com/Advanced-Lighting/Bloom . Attention c'est un shader avec OpenGL il faudra comprendre le concept et ensuite l'implémenter dans le raytracer, en faisant travailler le CPU pour boucler sur tous les pixels et appliquer le flou nécessaire, en 2 passes pour accélérer le rendu du flou, une passe horizontale puis une passe verticale.

[OPTIONNELLE][*] Ajout de **profondeur de champ** (DoF = Depth of Field). La focale a déjà été implémentée dans le code de base, il ne vous reste qu'à pouvoir modifier l'ouverture de la caméra pour simuler une profondeur de champ. Plus l'ouverture sera grande, plus la profondeur de champ sera petite (zone vue nette). Le plus simple pour implémenter cela est de tirer aléatoirement une nouvelle position pour la caméra pour chaque lancer de rayon (raytrace). Cette nouvelle position devra être dans un disque autour de la position de la caméra (position + glm::diskRand(rayonOuverture)).

[OPTIONNELLE][*] Utilisation des **cœurs du CPU** pour accélérer le rendu (open MP). Des informations peuvent être trouvées ici :

https://learn.microsoft.com/fr-fr/cpp/build/reference/openmp-enable-openmp-2-0-support?view=msvc-170

https://learn.microsoft.com/fr-fr/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170

[OPTIONNELLE][*] Ajout de **tonemapping** simple. On implémentera les plus simples : Reinhard / Extended Reinhard. Toutes les informations et formules nécessaires peuvent être trouvées sur la page suivant : https://64.github.io/tonemapping/

[OPTIONNELLE][**] Ajout de **motion blur** pour des animations simples (animation sur une trajectoire linéaire à vitesse constante). L'animation peut être ajoutée dans la classe Entity3D. Il faudra ajouter la « date » à laquelle l'animation commence, la durée de l'animation et les 2 positions (début/fin) de l'objet. Il faudra aussi ajouter une méthode pour évaluer la position en fonction d'un paramètre de temps. Lors du lancer d'un rayon il faudra générer un temps aléatoirement sur la période souhaitée pour le motion blur et passer ce temps lors des rebonds pour que ces rebonds puissent « taper » sur les autres objets évaluer au même moment (il faut « figer » la scène à l'instant tiré aléatoirement pour le rayon initial).

[OPTIONNELLE][**] Ajout de **textures** sur les objets. Il faut être capable d'aller chercher les UVs du point touché par le rayon pour récupérer une couleur dans une texture. Pour générer des UVs sur une sphère, rien de plus simple, il suffit d'utiliser la trigonométrie : vous calculez les angles (vertical entre -90° et 90° et horizontal entre 0 et 360°) et vous convertissez les valeurs entre -1 et 1. Vous serez alors en mesure d'appliquer une texture de type « planisphère » en l'appliquant à la sphère. Il vous faudra ensuite charger une texture (SDL_LoadBMP, cf. la classe texture du projet LittleEngine) pour être capable avec les UVs d'aller chercher le pixel correspondant. Dans l'idéal il faudrait « lisser » le pixel de texture en calculant la moyenne des pixels les plus proches (si le cœur vous en dit vous pouvez implémenter le lissage mais ce n'est pas obligatoire).

[OPTIONNELLE][***] Ajout de **réflexion ET de réfraction** avec un comportement cohérent (réflexion/réfraction/couleur du matériau « mixées » en fonction d'un coefficient de Fresnel). On supposera qu'aucun objet ne sera inclus ou en intersection avec un autre (pour faciliter les calculs et le passage entre les milieux afin de déterminer aisément l'indice courant). Des informations à retirer de cette page ? https://blog.demofox.org/2017/01/09/raytracing-reflection-refraction-fresnel-total-internal-reflection-and-beers-law/

NB : toutes les fonctionnalités devront être « mises en commun » pour avoir un rendu correct et un seul projet final. Ne négligez pas la mise en commun de votre code et la discussion pour réfléchir à la manière d'implémenter les différentes fonctionnalités.

Modalités

Le projet sera réalisé **par groupe** de 3 et devra être **rendu avant le 2 février 2026 minuit par mail à l'adresse** contact@foks-lab.fr . **Le mail devra avoir pour objet « ESIEA_GI2026 ». La composition des groupes devra être envoyée avant le 13 janvier 2026 minuit à cette même adresse avec le même objet.**

Le livrable devra contenir :

Le **code source du projet** (pas le projet compilé) qui devra être portable (il suffira de l'ouvrir avec Visual Studio 2022 sur une autre machine pour pouvoir le compiler puis l'exécuter en mode Release ou Debug)

Un **document PDF** listant les fonctionnalités implémentées et expliquant la façon dont cela a été réalisé (1 page maximum par fonctionnalité).

Le tout devra être rendu sous la forme d'**un fichier ZIP** (pas RAR, 7ZIP ou autre) nommé **GI2025_NOM1_NOM2_... . ZIP** ou NOM1/NOM2/... sont les noms des étudiants ayant participé au projet. Le ZIP contiendra **2 répertoires** :

SOURCES : dans lequel on retrouvera les sources du projet prêtes à être compilées sous Visual Studio (**contenant donc les fichiers solution et projet de Visual Studio 2022**). *Pour vérifier si le projet est valide, copier/coller le projet sur une autre machine que la vôtre et tester d'ouvrir puis compiler/exécuter* : **le projet DOIT être portable !**

RAPPORT : dans lequel on retrouvera le fichier **GI_2025_NOM1_NOM2_... . PDF** . Ce fichier devra lister sur sa page de garde les étudiants ayant réalisé le projet. Un sommaire indiquera chacune des fonctionnalités implémentées et ensuite chaque page détaillera une fonctionnalité.

NB : Un projet rendu hors délai se verra attribuer une note de 0, respectez la date qui vous sera donnée SVP.

NB : Le respect des instructions données sur cette page sera aussi pris en compte lors de la notation (ce sont des points facilement gagnés ... ou perdus si vous ne faites pas l'effort de respecter la consigne !).