Simulation 3D: Projet Mathématique

Introduction

Le but du projet étant de démontrer notre connaissance sur les transformations géométriques dans un espace à l’aide de vecteurs, j’ai décidé de réaliser une simple simulation 3D. La simulation permet de visualiser les opérations sur les vecteurs (Addition, soustraction, produit scalaire, mise à l’échelle, rotations) et d’appliquer des forces sur des objets (Cube) pour pouvoir visualiser comment les vecteurs peuvent affecter les objets dans un jeu vidéo.

Pour réaliser cette situation j’ai utilisé le C++, pour la simple raison de ma familiarité avec le langage. Puisque ce projet est conçu pour Windows seulement, le choix d’API graphique est limité. La simulation utilise DirectX 11 pour la même raison que le C++ a été choisi. Pour finir les choix techniques, pour le GUI, j’ai choisi ImGui (https://github.com/ocornut/imgui) qui est une librairie de GUI immédiat qui très différent des librairies dites évènementielles. La raison du choix est d’une part la familiarité avec celle-ci, mais aussi l’intégration facile avec le projet dû à son architecture.

En ce qui en est de la structure du code, voici les fichiers principaux contenu dans le projet :

**A screenshot of a computer

AI-generated content may be incorrect.**

De haut en bas :

-assets : Les fichiers qui définissent les modèles des entités utilisés pour la simulation.

-imgui: Les fichiers nécessaires au bon fonctionnement de ImGui.

-shaders : Les fichiers qui définissent les shaders (petit programme qui est exécuté sur le GPU) qui sont utilisés pour le bien de la simulation.

-src : Les principaux scripts du projet, la totalité du code que j’ai écris est contenu dans ce fichier.

-x64 : Le fichier .exe est contenu dans ce ficher.

-imgui.ini : Un autre fichier spécifique à ImGui. Ne pas toucher.

-Trois fichiers spécifiques à Visual Studio. Ne pas toucher.

Pour continuer sur la structure du code, j’utilise une forme de compilation particulière pour ce projet qui est un mixte entre un « Unity Build » (Rien à voir avec le moteur) et une compilation plus classique. Le « Unity Build » consiste à compiler qu’un seul des fichiers et à l’intérieur de ce fichier on inclut les autres fichiers .cpp. Par exemple :

**A screen shot of a computer code

AI-generated content may be incorrect.**

Dans le fichier principale (win32\_main.cpp) j’inclus les autres fichiers afin qu’il soit compilé eux aussi. Cependant, ImGui requiert certains fichiers d’être aussi compilé et que j’ai n’ai pas réussi à l’aide du même schéma que mes propres fichiers je dois aussi compiler ces scripts :

A screenshot of a computer program

AI-generated content may be incorrect.

Cette information est importante si seulement vous décidez d’ouvrir les fichiers dans un autre IDE ou éditeur de texte.

Implémentation de la simulation

Pour aller plus en profondeur sur la structure du code, voyons voir ce qui est contenu dans le fichier src :

A screenshot of a computer

AI-generated content may be incorrect.

Je vais expliquer brièvement la structure des fichiers et dossiers qui ne sont pas les plus importants pour vous.

-directx :

A number of days and months

AI-generated content may be incorrect.

Simplement trois fichiers qui sont l’implémentation du « Renderer » de la simulation.

-dx11\_camera.cpp contient le code qui est utilisé pour manipuler la caméra puisqu’on est dans une simulation 3D.

-dx11\_shaders.cpp contient le code qui est utilisé pour créer et manipuler les shaders qui sont utilisés dans la simulation.

-dx11\_main.cpp le fichier principal qui s’occupe de tous ce qui est nécessaire pour le bon fonctionnement de la simulation visuelle.

-imgui :

A screenshot of a computer

AI-generated content may be incorrect.

Ce sont simplement les fichiers internes de la librairie.

-utility

A white background with black text

AI-generated content may be incorrect.

-allocators.h est utilisé pour gérer la mémoire.

-inputs.hpp est utilisé pours les « inputs » de l’utilisateur (comme bouger la caméra).

-string.h est un simple fichier qui m’aide à manipuler les chaines de charactères.

-types.h défini des types de base utilisé dans tous le programme.

-UI

A number of time and date

AI-generated content may be incorrect.

L’implémentation de l’interface utilisateur. Le code peut être assez difficile à lire, si vous n’êtes pas habitué au GUI immédiat.

-asset\_table.cpp

Puisque le projet est de petite taille, j’ai décidé de garder les informations des différents types d’entités (vecteurs, cube, etc..) en mémoire.

-space.cpp

Simplement une implémentation d’une « grid » pour pouvoir mieux se localiser dans l’espace.

-win32\_main.cpp

Le fichier principal qui s’occupe entre autres de l’initialisation générale du programme.

Maintenant, parlons des fichiers les plus importants en commençant par math/vector.hpp.

A computer screen shot of a program code

AI-generated content may be incorrect.

Les vecteurs sont définis comme ceci. Pour comprendre le mot clé union, on peut prendre un vec\_3 V. On peut accéder aux membres de V de deux façons grâce à l’union :

1) V.x OU V.y OU V.z (OU V.w dans le cas du vec\_4)

2) V.AsArray qui est une façon d’accéder au composant du vecteur sous forme de tableaux ce qui peut simplifier quelques opérations parfois.

À noter que le tableau et les composants (x, y, z, w) occupent le même endroit dans la mémoire et que f32 représente un float (4 bytes).

Pour comprendre comment les opérations de base sur les vecteurs sont implémentés, il faut comprendre comment on peut changer la fonctionnalité des opérateurs dans c++.

Pour commencer, l’addition :

A computer screen shot of a black screen

AI-generated content may be incorrect.

On change la fonctionnalité de l’opérateur +. En arguments, on prend les deux vecteurs vl et vr. Ce qu’il faut comprendre c’est que si dans le code il est écrit : MonVecteur1 + MonVecteur2, les deux vecteurs vont être passés en arguments à la fonction qui va elle effectuer le calcul. On pourrait aussi faire la même chose sans changer la fonctionnalité de l’opérateur. Par exemple :

A computer screen with text

AI-generated content may be incorrect.^

Ces deux fonctions sont équivalentes.

Pour en revenir à l’implémentation de l’addition de vecteur c’est très simple, on prend tous les composants du vecteur et on les additionne. Rien de plus.

Pour la soustraction :

A computer screen with white text

AI-generated content may be incorrect.

Comme l’addition, on agit sur deux vecteurs, cette fois-ci en soustrayant les composants des vecteurs.

Pour la mise à l’échelle :

A computer screen shot of a black background

AI-generated content may be incorrect.

On utilise seulement un vecteur, et on multiplie tous ses composants par le scalaire, encore une fois très simple.

Pour le produit scalaire :

A screen shot of a computer code

AI-generated content may be incorrect.  
On accumule le résultat de la multiplication des composants des deux vecteurs utilisés.

Pour le produit vectoriel :

A computer screen with text

AI-generated content may be incorrect.

On utilise simplement une représentation dérivée de la formule avec les déterminants.

On fait aussi une simplification en inversant les deux opérations qui définissent le composant en Y et en supprimant le signe négatif.

Pour la longueur d’un vecteur :

A screen shot of a computer program

AI-generated content may be incorrect.

On utilise simplement le théorème de Pythagore qui stipule que a² + b² + c² = d² (En 3D).

SumSquared représente a² + b² + c². On vérifie que la somme est positive ou égale à zéro, puisque donné à la fonction sqrtf une valeur négative n’est pas défini et peut causer toute sorte de problème. Pour finir, on prend la racine carrée du résultat des composants au carré avec sqrtf(SquaredSum).

Pour la normalisation d’un vecteur :

A screen shot of a computer program

AI-generated content may be incorrect.

La normalisation est définie comme : Vecteur/||Vecteur||. Donc, on commence par aller chercher la longueur du vecteur, et on divise simplement tous ses composants par sa longueur.

Pour trouver l’angle entre deux vecteurs :

A screen shot of a computer program

AI-generated content may be incorrect.

Donc, ce qu’on sait sur la représentation du produit scalaire de façon géométrique.

Dot (vl, vr) = Length(vl) \* Length(vr) \* cos(theta), si vl est le vecteur de gauche et vr le vecteur de droite.

En réarrangeant l’expression on obtient : cos(theta) = (Dot (vl, vr)) / (Length(vl) \* Length(vr))

C’est exactement ce que le code fait : On trouve les deux longueurs et on les multiplie. On vérifie que le dénominateur n’est pas égale à 0, puisque diviser par 0 est une erreur fatale.

On prend ensuite le produit scalaire des deux vecteurs, on le divise par le dénominateur. On finit par prendre l’inverse de cos, défini par acos() et on retourne le résultat.

Pour la projection d’un vecteur sur un autre vecteur :

A screen shot of a computer program

AI-generated content may be incorrect.

La projection d’un vecteur est définie par : proj sur u de v = (v\*u / ||u||^2) scale u

La fonction calcule la longueur du vecteur qui est la cible de la projection en prenant le produit scalaire, car : dot(a, a) = ||A||\*||A||\*cos(theta) et on sait que cos(theta) d’un vecteur identique == 1, car aucun angle ne les sépare. Donc, dot(a, a) = ||A||^2 et on évite de calculer une racine carrée.

Ensuite, on vérifie que le dénominateur n’est pas 0, car, encore une fois, c’est une erreur fatale. On calcule le numérateur qui représente la longueur de la projection sur le vecteur cible et on la divise par l’échelle de la cible de la projection ce qui nous donne le facteur de mise à l’échelle qui nous permet de retrouver le point de la projection. On finit par l’appliquer au vecteur cible.

Pour la projection d’un vecteur sur un plan :

A screenshot of a computer program

AI-generated content may be incorrect.

On définit un plan par sa normale (Un vecteur qui pointe parallèle à sa surface, pour le plan XZ (0, 1, 0) est sa normale par exemple)

On prend donc les deux vecteurs, v (le vecteur qu’on projette) et la normale du plan.

La projection sur la normale du plan représente le composant du vecteur qu’on veut supprimer. Dot(v, pn) est utilisé pour savoir à quel point le vecteur v pointe en direction de la normale. Et simplement on le supprime vu vecteur de base par v – ProjectionOnNormal pour avoir la projection finale.

C’est tout pour les opérations utilisées sur les vecteurs. Quelques matrices et opérations sur les matrices sont utilisés pour la simulation des vecteurs.

Pour commencer, la structure qui représente la matrice :

A computer screen shot of a program code

AI-generated content may be incorrect.

Ensuite, la multiplication :

A computer screen shot of a program

AI-generated content may be incorrect.

La multiplication de matrice s’effectue en multipliant les rangées de la matrice qui transforme par les colonnes de la matrice de base. On remplit la matrice résultante par rangée. C’est-à-dire que la première valeur de la première rangée est égale à la première rangée de la matrice de transformation multiplier par la première colonne de la matrice de base. La deuxième valeur est égale à la première rangée de la matrice de transformation multiplier par la DEUXIEME colonne de la matrice de base, etc…

Pour en revenir à la fonction, on va chercher les quatre colonnes de la matrice de base. On boucle ensuite sur les quatre rangées de la matrice résultante en sauvegardant les résultats dans chaque composant de la rangée ce qui nous permet de remplir une rangée à chaque itération.

Le résultat est calculé en utilisant la méthode du produit scalaire, puisque les calculs nécessaires sont les mêmes.

Une autre opération qui est utiliser par la simulation mathématique est la rotation selon les angles.

A computer screen shot of a program

AI-generated content may be incorrect.

J’ai choisi d’utiliser les angles d’Euler même si elles ne sont pas les plus robuste (<https://en.wikipedia.org/wiki/Gimbal_lock>), On prend simplement les trois matrices de rotation autours des axes X, Y et Z, puis on les multiplie entre elle pour connaitre la rotation combine.

La visualisation

La visualisation est assez simple, premièrement, on a la liste des vecteurs définis par l’utilisateur sur la gauche.

A screenshot of a computer

AI-generated content may be incorrect.

On peut cliquer sur les vecteurs pour vois leurs informations. On peut changer la couleur, l’origine, la direction et les angles de rotations. A savoir que quand un vecteur est créé il est impossible de modifier son nom.

A screenshot of a computer

AI-generated content may be incorrect.

On peut aussi ajouter de nouveaux vecteurs.

A screenshot of a computer

AI-generated content may be incorrect.

On peut choisir les mêmes paramètres en plus du nom.

Maintenant la partie de droite contient plusieurs composants, à commencer par la calculatrice de vecteur.

A screenshot of a computer

AI-generated content may be incorrect.

Il y a plusieurs types de calculatrice vectorielle, mais elle fonctionne tous de la même façon. On a les deux composants de l’opération, le bouton pour effectuer l’opération et les résultats. A savoir qu’on peut donner un nom au vecteur résultant et aussi changer sa couleur. L’origine et sa direction sont inchangeables, car ils sont les résultats de l’opération.

Un autre exemple de calculatrice est la mise à l’échelle.

A screenshot of a computer

AI-generated content may be incorrect.

Pour changer le type d’Operations on utilise :

A screenshot of a computer

AI-generated content may be incorrect.

Et pour finir, la simulation du cube se présente comme ceci :

A screenshot of a computer

AI-generated content may be incorrect.

On peut modifier certaine propriété de la force qu’on veut appliquer au cube a l’écran. Il faut au minimum glisser un vecteur de force et choisir une puissance plus grande que zéro pour que la simulation fonctionne. A noter aussi que la force de la gravite a été réduite pour que le cube disparaisse de l’écran moins rapidement.

Améliorations

Je pense que le UI pourrait être un peu plus interactif, par exemple, le vecteur choisi pourrait avoir un contour noir ou blanc dépendamment de sa couleur. Il pourrait être aussi beaucoup plus esthétique.

La structure du code laisse parfois à désirer surtout dans les fichiers concernant le « Renderer » et les entités. Surtout que ce n’est pas une approche classique (OOP) et que ce n’est pas complément un projet personnel, j’aurais dû plus réfléchir à la structure générale.

La simulation physique est très simpliste, peut être plus d’options pour celle-ci même si ce n’était pas le but principal du projet.

Je pourrais aussi améliorer l’aspect visuel des vecteurs, ça peut être difficile de comprendre vers quelle direction ceux-ci pointent si on n’est pas familier avec les mathématiques.

Aucuns tests en profondeur n’ont été fait pour ce programme, donc des erreurs sont possibles.

Lien GitHub : https://github.com/GGRoy03/projet\_math