C++ - Synthèse d'images - Mathématiques

IMAC 2

— 2019-2020 —

World IMaker

Ce projet consiste en la réalisation d'un éditeur-visualiseur de terrain et de scène en 3D. Un certain nombre de fonctionnalités devront être implémentées afin de permettre à l'utilisateur de créer un monde à partir de blocs cubiques et de naviguer dedans.

Il vous permettra d'aborder des notions clé de la programmation en C++, du rendu 3D et de l'interpolation numérique.

I - Gestion de projet

Le projet devra être réalisé en **binôme** et remis au plus tard le **7 janvier 2020**. Un rapport, dont le contenu attendu est décrit plus bas, devra être fourni.

Vous devrez utiliser le gestionnaire de version **Git** et héberger votre dépôt sur une plateforme en ligne, telle que GitHub, GitLab ou Bitbucket. Nous vous demanderons de nous envoyer le lien vers ce dépôt rapidement, afin que nous ayons accès à l'historique de vos commits.

Vous pourrez trouver sur Internet de nombreux tutoriels et conseils pour prendre en main Git. Vous trouverez également ci-joint une brève notice décrivant les commandes les plus utilisées et la marche à suivre en cas de conflits (c'est normal et ça ne nécessite vraiment pas de paniquer 😉).

II - Cahier des charges

Un jour pas comme les autres, vous retrouvez votre meilleur ami Toto, artiste à ses heures perdues, dans le bar du coin. Autour d'une bonne pinte, celui-ci vous explique qu'il aimerait bien créer un joli monde en 3D dans lequel il pourrait naviguer, afin de pouvoir le présenter lors de son entretien de recrutement pour une boîte de dessin-animés. Le problème, c'est que Toto, c'est vraiment pas un as de l'ordinateur. Faire des beaux dessins, il sait faire, mais utiliser Maya ou Photoshop, ça il ne sait vraiment pas. A vrai dire, il ne sait même pas ouvrir ses mails, et son téléphone, c'est un vieux Nokia 3310. Du coup, il aimerait bien savoir si vous ne connaîtriez pas un logiciel vraiment facile à utiliser, qui lui permettrait de faire cela. Vous n'en connaissez pas, et après une rapide (peut-être trop rapide) recherche sur le web, vous êtes forcés de lui dire qu'hélas, ça n'a pas l'air d'exister… Or Toto, cette petite boîte de dessin-animés, c'est un peu son rêve, et l'entretien approche à grands pas ! En effet, celui-ci devrait avoir lieu début janvier, et Toto n'aura jamais le temps d'apprendre à maîtriser Blender d'ici là.

Heureusement pour lui, vous êtes ses deux meilleurs amis, et vous êtes tous les deux des pros de la prog' (ou en tout cas, vous aimeriez bien). Vous le rassurez donc en lui promettant que vous allez lui créer l'éditeur de scène le plus simpliste de la terre, tellement simple que même sa grand-mère pourrait faire des trucs avec ! Bon, par contre, qu'il ne se réjouisse pas trop non plus, il ne reste que deux mois, donc il ne pourra mettre que des cubes dedans. Mais comme Toto est super doué, il devrait être capable de faire une scène stylée, même s'il n'y a que des cubes.

A. L'application à produire

Après lui avoir longuement tiré les vers du nez, l'application que souhaite Toto consiste donc en un éditeur de terrains 3D, où le paysage est représenté uniquement par des cubes. La scène est donc un énorme pavé représentant le monde constitué de WxLxH cubes unitaires dont certains sont "vides" et d'autres sont des cubes de différentes couleurs. L'utilisateur devra non seulement pouvoir se déplacer dans la scène, mais également la modifier. Pour ce faire, vous serez amené à représenter correctement ce monde 3D constitué de cubes en mémoire, et donc à choisir une structure de données adaptée pour le représenter.

Le monde est donc constitué d'espaces vides et de cubes de différentes couleurs. Chaque cube de couleur différente peut être vu comme un type différent de cube. Il peut être donc intéressant de travailler sur la notion de type de cube comme vous le verrez par la suite.

Enfin, pour permettre l'édition du monde, et notamment de la couleur des cubes, vous devrez fournir à l'utilisateur des outils d'édition. Vous pourrez pour cela utiliser la librairie ImGui (voir les exemples d'utilisation ici: https://github.com/ocornut/imgui), qui vous permettra de développer très aisément différents types de menus.

B. Les fonctionnalités requises

Afin que Toto puisse réaliser son projet, un certain nombre de fonctionnalités devront être présentes. Si elles ne sont pas là, Toto ne pourra jamais réussir son entretien. Il faudra donc finaliser ces fonctionnalités avant de passer à la suite.

a) Affichage d'une scène avec des cubes

La première chose à faire est de pouvoir afficher une scène avec des cubes. L'état initial du monde que l'on souhaite modifier est une couche de 3 cubes de la même couleur à partir du bas (ou du fond) du monde. Il vous faudra pour y parvenir créer un moteur de rendu et les shaders permettant d'afficher des cubes de couleurs unies. Ca vous rappelle quelque chose, non ? Ah bah oui, vos TPs. Et la caméra pour se déplacer ? Ah et bien, c'est dans les TPs aussi.

b) Edition des cubes

C'est bien, c'est beau, on a plein de cubes. Mais c'est toujours les mêmes, et ils sont toujours au même endroit. Toto est perplexe, et il pense que ce n'est pas comme ça qu'il pourra créer sa scène merveilleuse. Qu'il se rassure, vous allez lui faire un outil qui lui permet de **sélectionner les cubes** pour en changer la couleur.

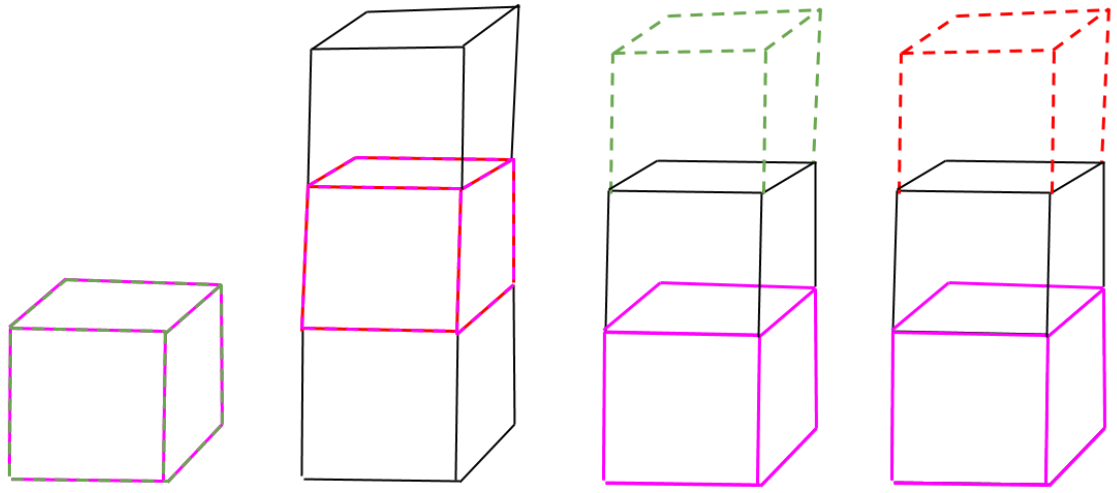
Commencez par ajouter au code la notion de "curseur", qui servira de pointeur vers la case sélectionnée par l'utilisateur. Cette case peut aussi bien contenir un cube qu'être vide. Assignez au curseur une valeur arbitraire au début du programme (il pourrait s'agir de la case au centre du pavé initial), et arrangez-vous pour que Toto puisse **déplacer ce curseur dans la scène grâce au clavier**. Il devra être capable de bouger dans toutes les directions de l'espace 3D : haut, bas, gauche, droite, avant et arrière. Comme vous n'avez pour le moment aucun retour visuel de la case sélectionnée, utilisez la sortie standard pour afficher dans la console ses coordonnées pour vérifier que le code fonctionne.

Une fois cette première étape terminée, vous allez devoir faire en sorte que Toto puisse voir la case qu'il sélectionne actuellement. Arrangez-vous pour **dessiner les contours** de cette case d'une couleur particulière, et faites en sorte qu'ils soient **toujours visibles**, même si la case est cachée derrière d'autres cubes.

Proposez finalement une option dans les menus pour que Toto puisse **modifier la couleur (ou le type)** du cube sélectionné.

c) Sculpture du terrain

Maintenant que Toto peut changer les couleurs, il veut aussi pouvoir changer la topologie du terrain. Proposez lui une manière d'**ajouter un bloc** dans la case sélectionnée si celle-ci est vide, ou bien de **supprimer le bloc** qu'elle contient si elle ne l'est pas. Faites également en sorte que Toto puisse extruder ou creuser le terrain, c'est-à-dire **d'ajouter ou de supprimer des cubes du haut de la colonne** dans laquelle se trouve le curseur.



d) Génération procédurale

Toto a beaucoup travaillé et il est fatigué. Vous aimeriez bien le remotiver en lui montrant des scènes dont il pourrait s'inspirer, mais hélas, vous êtes des programmeurs, et donc par définition feignants. Du coup, plutôt que de réaliser les scènes à la main, vous décidez d'implémenter un **système de génération procédurale** à l'aide de *radial basis function*.

=> voir section III - Mathématiques pour bien comprendre ce qui est demandé.

e) Ajout de lumières

Toto a une toute dernière requête concernant la création de sa scène. Il aimerait pouvoir réaliser deux ambiances : une de jour et une de nuit. Pour cela, il aimerait placer une lumière directionnelle pour représenter le Soleil, et un ou plusieurs points de lumière afin d'allumer des "lampadaires" (oui oui, avec une capacité d'abstraction suffisante, on peut imaginer plein de choses en voyant une tour de cubes).

Mettez en place le code permettant la gestion d'au moins une **lumière directionnelle** et un **point de lumière**. Ajoutez les outils nécessaires pour que Toto puisse placer ces objets dans la scène et les en retirer.

C. Les fonctionnalités additionnelles

Grâce à vous, Toto peut désormais créer sa scène et a une chance de réaliser son rêve. C'est bien pour lui, mais c'est bien pour vous aussi, car entre-temps, vous avez partagé votre logiciel sur Internet, et il a fait un carton ! Vous êtes maintenant super célèbres, et des dizaines d'utilisateurs vous envoient chaque jour des lettres d'amours, et aussi de temps en temps, des demandes d'amélioration.

Comme vous êtes sympas, et qu'il reste un peu de temps avant l'entretien de Toto, vous décidez d'implémenter deux d'entre elles afin qu'il puisse aussi en profiter.

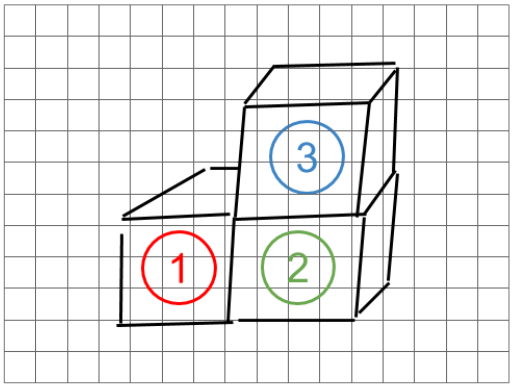
a) Amélioration de la sélection

Les utilisateurs de votre logiciel trouve cela assez fastidieux de devoir déplacer le curseur avec les flèches directionnelles. Toto vous l'avait également rapporté, mais il n'avait pas insister plus que ça, après avoir vu les cernes ancrées sur votre visage… Cependant, vous êtes bien obligé de le reconnaître, on est loin de l'expérience utilisateur intuitive que vous lui aviez promis.

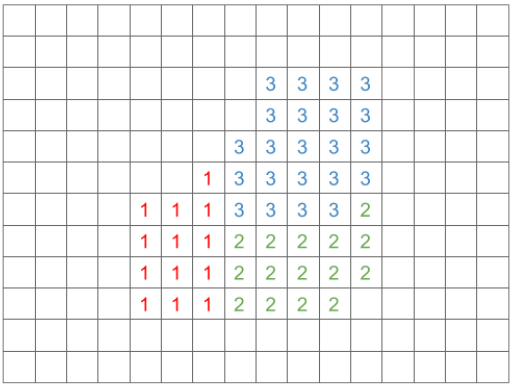
Afin d'améliorer cela, vous allez modifier le code de votre curseur pour calculer sa position à partir de celle du curseur de la souris. Comme vous êtes malins, vous décidez d'utiliser un FBO.
=> voir http://www.opengl-tutorial.org/fr/intermediate-tutorials/tutorial-14-render-to-texture/

Vous allez donc devoir créer un nouveau set de shaders, qui attend en paramètre une variable permettant d'identifier chaque case de la scène, et l'écrit dans la texture associée au framebuffer en utilisant la même projection que celle utilisée pour faire le rendu de la scène à l'écran. Le rendu devra être effectuée sur toutes les cases "sélectionnables", c'est-à-dire les cases contenant un bloc, mais aussi les premières cases de chaque colonne vide (autrement, il ne sera jamais possible de positionner le curseur sur une colonne dont toutes les cases ont été supprimées).

Une fois la texture rendue, vous pourrez utiliser glReadPixels pour récupérer la valeur identifiant la case placée en dessous du curseur de la souris.



Ecran



Framebuffer

b) Outils de painting

Prérequis: Amélioration de la sélection

Toto trouve ça bien de colorier des cubes, mais quand il y en a 300 et qu'il doit changer leur couleur individuellement, c'est moins bien. Il aimerait donc pouvoir modifier la couleur de plusieurs cubes à la fois, comme s'il peignait un tableau.

Faites en sorte que Toto puisse colorier rapidement les cubes placés en dessous de son curseur lorsqu'ils les balaient de la souris (en maintenant une touche du clavier ou de la souris enfoncée par exemple). Améliorez ensuite ce système de manière à pouvoir choisir un diamètre délimitant la zone à l'intérieur de laquelle la coloration s'applique.

c) Sculpting ++

Prérequis: Amélioration de la sélection

De la même manière que pour la coloration, Toto aimerait bien pouvoir sculpter son terrain de façon plus rapide et plus intuitive. Pour cela, commencez par afficher la projection d'un disque horizontal sur le sol, ayant pour centre la position du curseur. Ensuite, appliquez à l'ensemble des colonnes placées à l'intérieur de ce disque une fonction gaussienne permettant de calculer leur nouvelle hauteur (c'est-à-dire le nombre de cubes les composant). Faites ainsi en sorte que plus les colonnes sont proches du centre du disque, plus elles soient impactées par le changement de hauteur.

d) Sauvegarde / Chargement de la scène

Pour l'instant, Toto est obligé de recréer sa scène de zéro dès qu'il relance le programme. Afin de lui éviter cette perte de temps, vous allez lui fournir un système de sauvegarde. Vous devrez pour cela être capable d'**écrire un fichier sur le disque contenant les informations permettant de recréer la scène** qu'il a construite. Vous pouvez formater ce fichier de la manière que vous souhaitez. Il vous faudra également être capable de **lire ce fichier** et de **régénérer les éléments correspondant dans la scène**. Enfin, il vous faudra ajouter **les options de sauvegarde et de chargement dans les menus**. Dans un premier temps, vous pourrez utiliser un chemin statique pour le fichier. Ensuite, il faudra que Toto puisse spécifier l'endroit où il souhaite enregistrer son fichier.

e) Importer-Exporter de scène

~~Prérequis: Sauvegarde / Chargement de la scène~~

~~*Votre logiciel est tellement génial que des milliers d'artistes trouvent cela plus simple de bâtir leur scène avec lui plutôt qu'avec EarthSculptor. Néanmoins, il y a quand même des choses qu'ils ne peuvent faire qu'avec EarthSculptor et ils aimeraient donc pouvoir ré-ouvrir leur scène dedans.*~~

~~Pas de soucis, comme vous adorez votre communauté et qu'en plus, ça ne vous demandera pas un travail considérable, vous décidez de partager votre format de données en créant un~~

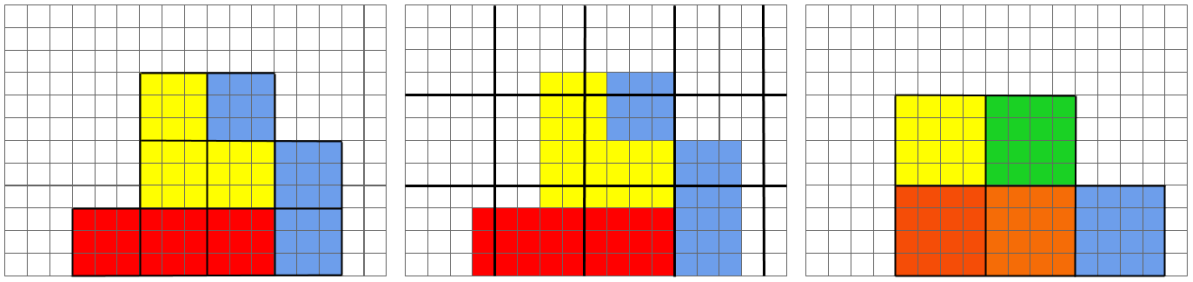
~~importer pour assimp. Comme ça, tous les logiciels utilisant assimp obtiendront le support pour votre format ! Et puis bon, tant qu'à faire l'importer, autant faire l'exporter. Et tant qu'à faire un importer et un exporter, autant refactoriser le code gérant la sauvegarde et le chargement à l'aide de ces deux objets.~~

f) Chargement de modèles 3D

Afin que Toto puisse donner un peu de vie à sa scène, arrangez vous pour qu'il puisse **importer des modèles 3D, ainsi que leurs éventuelles textures**, dans la scène. Vous devriez pouvoir le faire en utilisant la librairie assimp et en adaptant vos shaders.

g) Niveau de discrétisation

Arrangez-vous pour que les utilisateurs puissent modifier dynamiquement la résolution de tous les cubes de leur scène. Afin d'y arriver, vous pouvez dans un premier temps sectionner l'ensemble des cubes de manière à ce qu'ils atteignent une taille unitaire, puis les fusionner entre eux jusqu'à obtenir la taille demandée par l'utilisateur (conservez les blocs composés de 50% de blocs unitaires au moins, supprimez les autres). Faites la moyenne de la couleur des blocs unitaires pour obtenir la couleur du bloc résultant.



h) Blocs texturés

Certains utilisateurs vous ont remonté le côté trop cartoon des scènes générées, et ils souhaiteraient mettre en place un environnement plus réaliste, notamment en assignant des textures aux blocs.

Permettez aux utilisateurs de définir des sets de textures qu'ils pourront assigner aux cubes qu'ils souhaitent (une texture par face, donc six textures par cube).

i) Vos idées

Si aucune des fonctionnalités présentées dans cette partie ne vous inspire, vous pouvez toujours proposer vos propres idées. N'hésitez pas à contacter l'un de vos professeurs pour qu'ils valident leur intérêt dans le cadre de ce projet.

III - Mathématiques

Vous savez combien Toto a souffert de la blague 0+0=.... Pourtant, lui et vous adorez les maths. Par esprit de revanche, vous vous proposez le challenge suivant : être capable de générer procéduralement la géométrie d'une scène en utilisant les ***radial basis functions***.

A. Radial basis functions

Vous vous souvenez très bien de ce que c'est, mais par acquis de conscience, vous regardez quand même ce tuto : http://www-igm.univ-mlv.fr/~vnozick/divers/rbf.pdf.

B. Géométrie générée

Vous l'avez bien compris, les *radial basis functions* permettent d'extrapoler des valeurs scalaires dans un espace vectoriel de dimension *n* à partir de points de contrôle. Dans notre cas, il s'agit d'un espace tridimensionnel, où l'on spécifie pour certains points 3D (points de contrôle) une valeur relative à la possibilité d'être un bloc ou non. On peut extra/interpoler ces valeurs pour n'importe quelle case 3D de l'espace. Si pour une case donnée, la valeur interpolée est supérieure à 0, il faut dessiner un bloc, sinon, la case est laissée vide.

En pratique, l'idée est de générer une scène à partir de quelques points de contrôles plutôt que de générer tous les blocs à la main. Une possibilité consiste à créer un fichier contenant une série de points 3D ainsi qu'une valeur pour chacun de ces points. Concernant les fonctions radiales, vous pourrez en définir quelques une dans votre code et ajouter dans le fichier de configuration le nom de la fonction choisie, ainsi que ses paramètres. Il faut également spécifier des bornes dans l'espace au delà desquelles il n'est plus nécessaire de créer des blocs.

Votre programme devra lire ce fichier de configuration et générer les blocs sur la zone 3D spécifiée. Vous pouvez considérer pour cette partie que le fichier de configuration fourni par l'utilisateur a toujours le format attendu par votre parseur (afin de ne pas perdre de temps à mettre en place la gestion des erreurs de parsing).

C. Fonctions radiales

Toto s'aperçoit rapidement que, selon la fonction radiale qu'il choisit, le résultat de la scène générée est très différent. Pour lui, c'est comme une drogue, il commence à expérimenter plein de fonctions radiales avec différents points de contrôle. Vous sentez bien que si vous ne faites pas de même, il vous en voudrait. Au fil de vos expérimentations, vous vous apercevez que certaines fonctions radiales sont plus efficaces que d'autres pour générer certain types de scènes, comme des montagnes, des îles, etc. Prenez le temps de faire un bilan de vos observations et les faire figurer dans le rapport.

IV - Code (C++)

Ce projet est à réaliser en C++. Il est largement conseillé d'utiliser une version moderne du C++ (11, 14 ou 17).

A. Compilation

Vous devrez fournir les fichiers permettant de générer les artefacts de compilation avec CMake (au moins 3.10). Vous devrez également spécifier les options -W, -Wall et -Werror pour compiler votre projet.

B. Bibliothèques autorisées

Les bibliothèques utilisables sont:

- la bibliothèque standard du C++,
- SDL (version 1.2 ou 2) ou glfm pour la gestion des fenêtre,
- ImGui pour l'interface utilisateur,
- OpenGL et GLEW,
- assimp pour l'import et l'export de modèles 3D et des textures,
- glm et glimac.
- Eigen pour les *radial basis functions*.

Si vous souhaitez utiliser une autre bibliothèque, vous devrez nous en envoyer la demande rapidement afin que nous validions la cohérence de son utilisation dans le cadre de ce projet.

C. Documentation du code

Nous attendons un code lisible, bien organisé et auto-documenté. Cela signifie que votre code doit être compréhensible sans qu'il soit nécessaire d'y ajouter des commentaires : essayez de nommer vos variables et vos fonctions explicitement (utilisez 'cubeCount' au lieu de 'n' par exemple) et n'hésitez pas découper de longs algorithmes en plus petites fonctions.

D. Organisation du code et architecture logicielle

Avant de coder, vous prendrez le temps nécessaire pour réfléchir à la conception de votre programme. Il faudra joindre au rapport les diagrammes de classes et les choix d'architecture de votre logiciel, en justifiant vos choix.

Privilégiez une architecture simple de classes ou de structures décrivant et organisant les différents constituants de l'éditeur-visualiseur (on pourra par exemple retrouver des noms de classes comme Window, Scene, Cube, Toolbox, Map ou Camera).

Essayez de mettre en application différentes bonnes pratiques de la programmation en C++. Pensez par exemple à ne pas réinventer la roue en utilisant la librairie standard (vector, array, map, string, pair, tuple, algorithm), utilisez des smart pointers pour éviter les fuites de mémoire (unique_ptr), initialisez vos classes via la liste d'initialisation du constructeur, passez les paramètres volumineux en mémoire (c'est-à-dire les classes, pas les types primitifs ou les enums) par référence constante lorsque c'est possible, etc.

E. Vérification des acquis

Voici ce qu'il serait bon de rencontrer dans votre projet :

- classes
- classes et fonctions templatées
- héritage
- polymorphisme
- usage massif d'outils de la STL
- fonctions lambdas
- des fonctions et des variables constexpr
- des exceptions pour traiter les erreurs systèmes (version d'opengl incompatibles, plus de ram, etc)
- des messages d'erreurs pour traiter les erreurs utilisateurs (fichiers inexistants, entrées invalides, etc)
- des asserts pour détecter et prévenir les erreurs de programmation (division par zéro, valeur nulle non attendue, etc)

Voici ce qu'il serait regrettable de rencontrer dans votre projet :

- une mauvaise organisation de vos fichiers et de vos classes
- des variables, des fonctions ou des types mal nommés
- des références non constantes alors qu'elles devraient l'être
- des passages par copies non justifiés
- des erreurs de segmentation
- des fuites de mémoires
- des bugs
- du code de debug non retiré
- du code mort (= des portions de code non utilisés par le programme)
- du code dupliqué
- du code illisible d'un point de vue sémantique (des for dans des for dans des if dans des for dans des else par exemple)
- du code illisible d'un point de vue esthétique (indentation irrégulière, mélange camelCase / snake_case, mélange tabs / spaces)
- des mega fonctions de plus de 30 lignes
- des mega fichiers de plus de 500 lignes
- des variables globales non constexpr
- des crashs lorsque l'utilisateur effectue une opération non prévue

V - Evaluation

L'évaluation sera basée sur différents aspects.

A. Fonctionnalités de l'application

- L'application se lance correctement et ne plante pas.
- **Toutes** les fonctionnalités requises ont été implémentées.
- Au moins **deux** fonctionnalités additionnelles ont été implémentées.

B. Coopération et gestion de projet

- Les deux membres du binôme se sont impliqués dans la réalisation du projet et devront pouvoir présenter la manière dont ils y ont contribué. Il n'est pas nécessaire que les deux membres aient travaillé à part égale sur le projet, néanmoins, il est attendu que chacun ait pris part au développement du code des fonctionnalités nécessaires (cela afin d'éviter les traditionnels "j'ai écrit le code et il a écrit le rapport" ou "j'ai écrit le code et il a dessiné les textures").
- Le projet a été réalisé sur une durée raisonnable et n'a pas été démarré au dernier moment.
- Les messages de commits sur le dépôt Git sont clairs et décrivent l'ensemble les changements qu'ils apportent.

C. Rapport

Votre rapport devra comprendre :

- Une synthèse sous forme de tableau récapitulatif, spécifiant pour chaque fonctionnalité si elle a été implémentée intégralement, partiellement ou pas du tout. Dans le cas partiel, vous préciserez quels éléments fonctionnent et lesquels ne fonctionnent pas.
- Pour chacune des fonctionnalités implémentées, la présentation de la manière dont elle a été implémentée (architecture, design patterns, uml, etc).
- Les résultats (captures d'écran par exemple) obtenus par génération procédurale en fonction des différentes fonctions radiales et points de contrôle utilisés, ainsi qu'une explication / analyse des choix entrepris.
- La présentation des difficultés rencontrées (s'il y en a) et des éléments qui vous ont empêché de réaliser ce que vous souhaitiez.
- Une partie individuelle décrivant ce que vous avez apprécié ou non dans la réalisation du projet, et ce que vous pensez avoir appris.

N'oubliez pas que le rapport compte pour une bonne partie de la note, ne le négligez pas.