

Document de design

Sommaire	2
Interactivité	3
Technologie	4
Compilation	5
Architecture	7
Fonctionnalités	9
Ressources	23
Présentation	24

Sommaire

Lors du cours d'infographie IFT-3100 de l'hiver 2018, notre équipe a réalisé une application visant à simuler une version amateur d'un moteur graphique comme Unreal Engin ou Unity.

L'objectif principal du projet de session étant de développer une application qui permet de construire, éditer et rendre des scènes, nous avons choisi de prendre comme ligne directrice une approche orientée vers le jeu vidéo.

Tout au cours du projet, nous avons dû choisir des thèmes à développer parmi 50 objectifs fonctionnels, soit 25 par livrable.

Chaque tranche de 5 objectifs fonctionnels cible un des sujets visités lors du cours.

Lors de l'évaluation des objectifs, ceux choisis seront évalués avec un système d'étoile ayant cette légende:

*** au-delà des attentes

** au niveau des attentes

* en bas des attentes

Les points du projet sont accordés en conséquence, soit un point par étoile pour un cumulatif maximum de 40 points.

Afin de profiter au maximum des connaissances théoriques du cours, nous avons choisi des technologies et designs nous permettant d'étendre au maximum nos apprentissages sans pour autant se couper les ailes.

Interactivité

Dans le logiciel conçu, il existe plusieurs formes d'interactivité afin de mettre l'utilisateur à l'aise.

Afin de permettre à l'utilisateur d'admirer son monde dans toute sa splendeur, nous avons implémenté une vision par caméra en première personne. La caméra est accessible à l'aide d'un clic droit soutenu et le mouvement de la souris sur la fenêtre.

Pour accentuer la caméra, nous permettons aussi le mouvement de la caméra accessible à l'aide d'un clic droit soutenu et l'utilisation des touches flèches.

L'importation de modèles 3D de plusieurs types est possible directement dans le programme à l'aide de l'interface graphique utilisateur.

Il est possible d'effectuer une capture d'écran avec l'interface utilisateur, ainsi qu'à l'aide de la touche F11 du clavier, ce qui fournit un raccourci intéressant pour accélérer la manoeuvre.

Il est aussi possible de faire l'importation d'une texture dans le monde directement sur une forme de type quadrilatère.

Afin de bien gérer les objets du monde, il existe une liste représentant le graphe de scène interne à l'application. Elle permet ainsi de voir tous les objets entourant la caméra, leurs regroupements et des outils pour les modifier. Voici quelques modifications possibles grâce aux outils fournis:

- translation, la rotation et l'échelle d'un objet quelconque.
- choix de couleur gérée par un sélecteur de couleur.
- possibilité de dessin de primitives vectorielles non affectées par le monde 3D.
- une case peut être cochée afin d'activer le *skybox*.
- possibilité de changer le curseur pour mieux plaire à l'oeil de l'utilisateur.
- etc.

Technologie

Les outils utilisés ont été choisis afin de permettre une certaine liberté dans la conception en offrant l'accès direct à OpenGL.

Logiciel système (*Framework*): Nous avons décidé d'utiliser SDL et SDL_Image, un framework qui semble bien aimé des professionnels afin de gérer les événements et les fenêtres. Toutefois son utilisation dans le projet se limite sensiblement à ceci, car nous souhaitons utiliser l'essence même du OpenGL et non passer par des fonctionnalités préconçues comme celles de Openframeworks. Heureusement, SDL offre la possibilité d'utiliser directement un contexte OpenGL dans les fenêtres.

Gestion de l'interface graphique utilisateur: Pour le GUI, nous avons décidé d'utiliser un projet disponible sur github créé par Omar Cornut nommé ImGui. Une fois installé, cet outil offre l'accès à plusieurs fichiers c++ qui contiennent parfois des défauts, mais de par leur disponibilité même permettent la correction de ces problèmes.

Activation des fonctionnalités du OpenGL moderne: Pour l'activation de ces fonctionnalités sur Windows, nous avons utilisé glew de nigel-com. Bref, l'utilisation de cette librairie s'effectue en deux lignes, soit activer le mode expérimental et initialiser glew.

Mathématiques: Pour la convivialité du côté des mathématiques nécessaires à l'infographie, nous avons utilisé la librairie OpenGL Mathematics (GLM) qui permet l'utilisation de vecteur et matrices rapidement compatibles avec les shaders d'OpenGL.

Gestion de la compilation: Pour la gestion des liens, de comment construire le projet et aider l'installation sur les ordinateurs à l'externe, nous avons choisi d'adopter la solution CMAKE qui est compatible avec les différents systèmes d'exploitation, ce qui permet alors le développement facile pour tous.

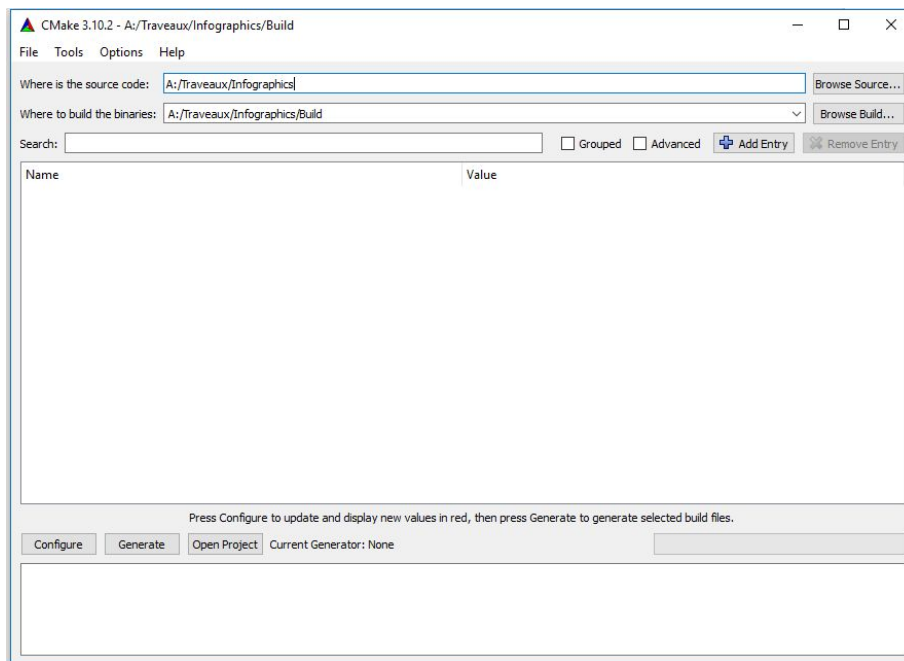
Gestion des versions: Pour la gestion des versions, nous avons profité de la gratuité de github pour les étudiants afin de nous créer un dépôt. Ceci permet donc de gérer la fusion automatique de différentes parties du projet réalisées par des étudiants différents. Github assure aussi la sécurité du projet en fournissant une façon de récupérer une version précédente en cas de catastrophe.

Chargement des modèles 3D externes: Pour le chargement des modèles, nous avons adopté une librairie cross-platform dénommée open asset import library (Assimp). Comme le nom le dit, elle sert à lire les vertices et indices de texture d'un modèle particulier. Toutefois, nous devons quand même gérer l'ajout de ces éléments dans des tableaux OpenGL et les envoyer au shader.

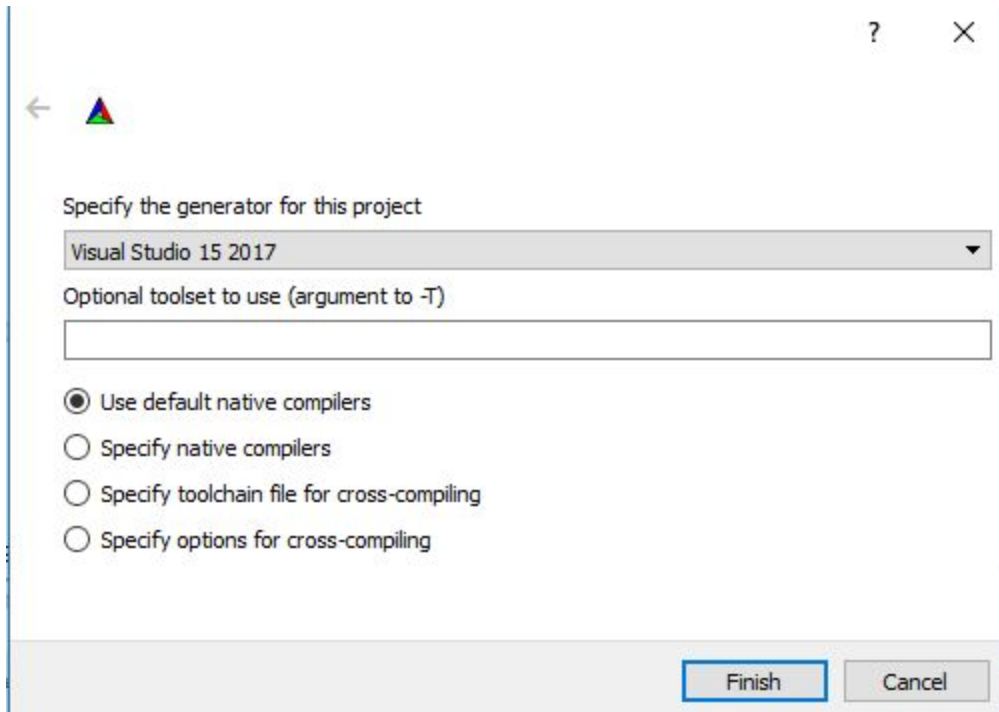
Compilation

Voici le processus pour avoir accès à l'outil de compilation sur Windows:

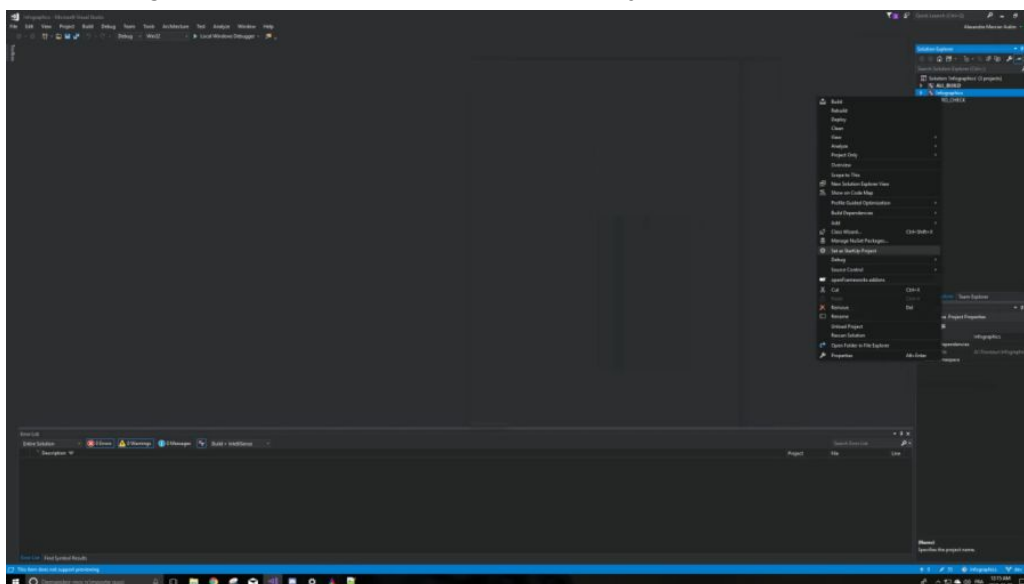
1. Installez Cmake selon l'architecture de l'ordinateur au lien suivant: <https://cmake.org/download/>
2. Dans le champ source code, entrer le chemin vers le dossier Infographics et dans le champ binaries, entrez le même chemin en ajoutant /Build. Voici ce à quoi la fenêtre devrait ressembler:



3. Cliquez sur configure et choisir l'éditeur que vous utilisez, pour les étapes, nous assumons que vous utiliserez Visual Studio 15. Voici ce à quoi la fenêtre devrait ressembler:



4. Cliquez sur Finish
5. Une fois la configuration terminée, appuyez sur Generate.
6. Une fois la génération terminée, appuyez sur Open Project.
7. Dans Visual Studio, aller dans l'explorateur de solution, faire un clic droit sur le projet Infographics et choisir set as StartUp Project.



8. Une fois que le projet est choisi comme projet de démarrage, cliquer sur build.
9. Rouler le programme et apprécier la vue.

Architecture

L'architecture du projet se divise en plusieurs parties comme suit:

Main: Se charge uniquement de préparer l'application, de partir la boucle principale et de fermer l'application.

Application: Se charge principalement de l'initialisation de la fenêtre utilisée par Renderer, la boucle principale qui permet la capture des événements système et de l'appel de la fonction de dessin du Renderer.

Renderer: Se charge de la gestion de l'interface graphique et des dessins au bon moment des objets graphiques de Scene. Il est aussi responsable des dessins de l'interface graphique utilisateur supportée par ImGui. Elle délègue les événements en lien avec la scène à son instance de l'objet Scene.

Scene: Cette classe contient tous les objets d'une scène dans un GroupObject et comporte l'essentiel à leur rendu, soit des matrices de vue et de perception. Cette classe gère la caméra et l'interaction avec les objets du monde virtuel.

AbstractObject: Classe polymorphe qui permet de classer les objets du monde. Cette classe comporte plusieurs fonctions utiles comme uniformColor qui prépare une couleur uniforme ou les fonctions permettant de faire des matrices de translation, rotation et échelle.

GroupObject: Enfant de AbstractObject qui contient une liste d'AbstractObject afin de permettre un classement des objets monde sous forme d'arborescence.

QuadObject: Enfant de AbstractObject permettant de dessiner un quadrilatère dans le monde 3D

CubeObject: Enfant de AbstractObject permettant de dessiner un cube

SkyboxObject: Enfant de AbstractObject permettant de dessiner un skybox.

PrimitiveObject: Enfant de AbstractObject permettant de dessiner les différents types de primitives vectorielles. Elle est aussi utilisée pour faire le dessin du curseur de la souris.

ModelObject: Enfant de AbstractObject qui permet de dessiner un objet préalablement importé par la classe Model.

Nous avons aussi un diagramme de classe rapide qui représente bien l'application, il est disponible en meilleure qualité dans le vpp du dossier doc (Ouvrable avec *Visual Paradigm*).



Fonctionnalités

1.1 Importation d'images

Pour importer une image, il suffit d'entrer le chemin de l'image à importer dans le champ *Fichier* de la fenêtre *Importer / Exporter*, puis cliquer sur le bouton *Importer image*, tel qu'illustré sur l'image suivante, suivie du résultat de l'importation:



Ce qui se produit ici dans le code est qu'un objet de type *QuadObject* est ajouté à la scène, en utilisant l'image importée comme texture.

Note 1: Quelques images sont disponibles dans le dossier *Resources* du projet pour simplifier les tests.

Note 2: Les images au format JPEG (ou JPG) ne sont pas supportées.

1.2 Exportation d'images

L'exportation d'images est l'équivalent d'une capture d'écran de la fenêtre de l'application. L'image est cherchée dans le buffer de dessin et ensuite convertie en *SDL_Surface* pour ensuite être enregistrée en BMP grâce à *SDL_SaveBMP*. La raison de la nécessité de l'inversion en y semblerait être que sous Windows, les fichiers BMP sont inversés nativement. Pour expérimenter la fonction, il suffit d'appuyer sur F11 pour prendre la capture ou de cliquer sur le bouton *Capture d'écran* dans la fenêtre *Importer / Exporter*. Lorsqu'on utilise le bouton de capture, on peut également sélectionner l'endroit où l'image sera enregistrée en écrivant le chemin dans le champ *Fichier* un peu plus haut.

Voici la partie principale du code:

```
void Renderer::screenShot(int x, int y, int w, int h, const char * filename)
{
    unsigned int size = w * h * 4;
    unsigned char *pixels = new unsigned char[size]; // 4 bytes for RGBA
    glReadBuffer(GL_FRONT);
    glReadPixels(x, y, w, h, GL_BGRA, GL_UNSIGNED_BYTE, pixels);

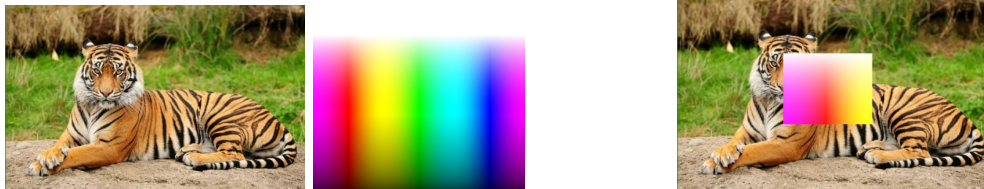
    //vertical flip cause glRead goes backwards for some reason
    unsigned char *flipPixels=new unsigned char[size];
    for (int i = 0; i < w; ++i) {
        for (int j = 0; j < h; ++j) {
            for (int k = 0; k < 4; ++k) {
                flipPixels[(i + j * w) * 4 + k] = pixels[(i + (h - 1 - j) * w) * 4 + k];
            }
        }
    }

    SDL_Surface * surf = SDL_CreateRGBSurfaceFrom(flipPixels, w, h, 8 * 4, w * 4, 0, 0, 0, 0);
    SDL_SaveBMP(surf, filename);

    SDL_FreeSurface(surf);
    delete[] pixels;
    delete[] flipPixels;
}
```

1.3 Échantillonnage d'image

Pour l'échantillonnage d'image, il est possible de générer une image à partir d'échantillons de pixels en provenance de deux images. Il suffit simplement de sélectionner une image de base et une image qui va servir d'échantillon de pixels, puis de choisir le pourcentage de cette image qui va être échantillonnée et de sélectionner où nous voulons faire afficher cet échantillon sur l'image de base pour finalement générer une nouvelle image. Tout ceci est géré à l'aide de l'interface graphique dans la fenêtre d'échantillonnage d'image. Voici un exemple où les deux images de gauche ensemble deviennent l'image de droite.



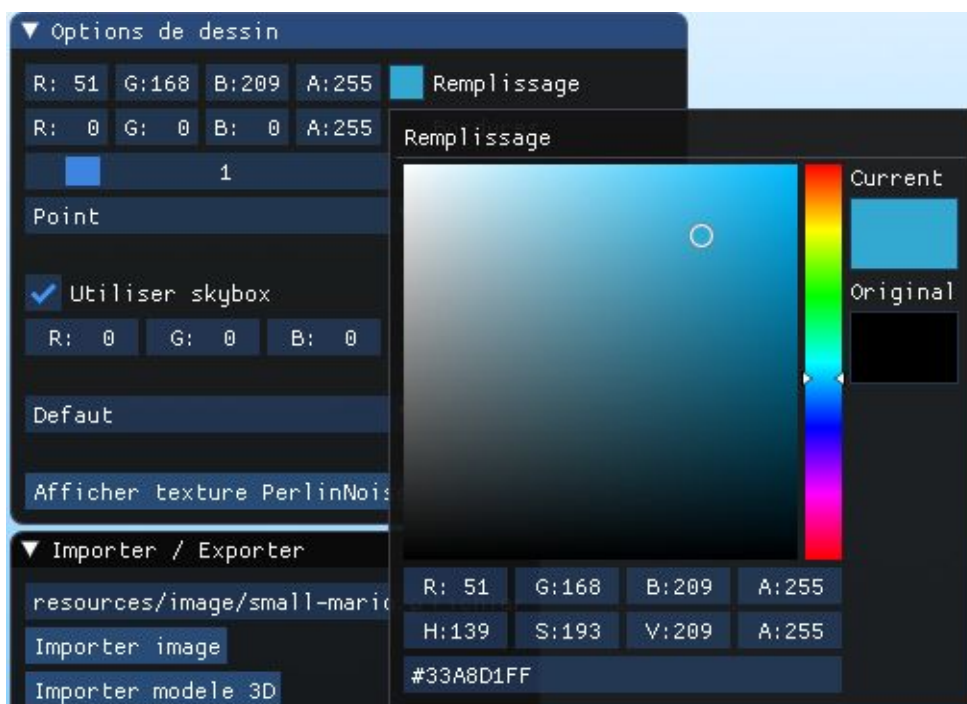
1.4 Sélecteur de couleur

Des sélecteurs de couleur sont disponibles à quatre emplacements différents dans l'application.

1. Dans la fenêtre *Options de dessin*, on peut sélectionner la couleur de remplissage utilisée pour le prochain dessin de primitive.
2. Dans la même fenêtre, on peut sélectionner la couleur des bordures pour le prochain dessin de primitive.
3. Toujours dans la fenêtre *Options de dessin*, il est possible de sélectionner la couleur de l'arrière-plan de la scène lorsque le skybox est désactivé. Ce sélecteur n'inclut pas de composante alpha, contrairement aux trois autres, puisque ce serait inutile dans le contexte d'une couleur d'arrière-plan.
4. Dans la fenêtre *Transformations* (qui apparaît lorsqu'un élément est sélectionné dans le graphe de scène), il est possible d'aller modifier la couleur des éléments sélectionnés.

Ces sélecteurs de couleur sont entièrement gérés par *ImGui*. Il y a trois façons d'éditer la couleur à l'aide de ces sélecteurs. On peut soit:

- Double-cliquer sur l'une des composantes de couleur et modifier manuellement sa valeur à l'aide du clavier.
- Cliquer sur l'une des composantes puis, tout en maintenant le bouton enfoncé, glisser vers la droite pour augmenter la valeur ou glisser vers la gauche pour la réduire.
- Cliquer sur le carré d'aperçu de couleur à droite des champs d'édition pour ouvrir une petite fenêtre supplémentaire permettant de sélectionner la couleur intuitivement, puis cliquer à côté pour fermer la fenêtre.



1.5 Espace de couleur

Les différents espaces de couleurs sont gérés par exemple lors de l'importation des textures.

Le tout fonctionne grâce à une petite fonction qui permet de classer quel est le type d'image selon ses caractéristiques, soit le nombre d'octets par pixel et son résultat de l'application d'un masque de rouge. Autrement dit, nous gérons RGB, BGR, RGBA et BGRA.

Voici cette fonction:

```
void Model::getImageProperties(SDL_Surface *image, GLint &nOfColors, GLenum &texture_format)
{
    if (image == NULL)
    {
        cout << "can't get properties of a null image" << endl;
        return;
    }

    nOfColors = image->format->BytesPerPixel;
    if (nOfColors == 4) // contains an alpha channel
    {
        if (image->format->Rmask == 0x000000ff) texture_format = GL_RGBA;
        else texture_format = GL_BGRA;
    }
    else if (nOfColors == 3) // no alpha channel
    {
        if (image->format->Rmask == 0x000000ff) texture_format = GL_RGB;
        else texture_format = GL_BGR;
    }
    else
    {
        cout << "weird texture detected" << endl;
    }
}
```

De plus, les sélecteurs de couleur de *ImGui* permettent si on le souhaite de sélectionner la couleur en format HSV lorsqu'on ouvre l'éditeur de l'un des sélecteurs (se référer à l'image de la fonctionnalité 1.4).

2.1 Curseur dynamique

L'application offre la possibilité de modifier le curseur de la souris pour cinq différents dessins vectoriels au choix. Pour sélectionner le curseur souhaité, il suffit de sélectionner son préféré dans la liste déroulante *Curseur* de la fenêtre *Options de dessin*. Voici à quoi ressemble ces différents curseurs:



L'essentiel du code qui gère les curseurs se trouve dans `Renderer::updateCursor()` et `Renderer::drawCursor()`. Le code n'est pas inclus ici car il est un peu trop long.

2.2 Outils de dessin

La fenêtre *Options de dessin* offre la possibilité à l'utilisateur de modifier la valeur des différents outils de dessin tels que:





- La couleur de remplissage de la prochaine forme qui sera dessinée
- La couleur des bordures de la prochaine forme qui sera dessinée
- L'épaisseur des bordures de la prochaine forme qui sera dessinée (si la valeur est 0, la forme n'aura pas de bordures)
- La couleur de l'arrière-plan (prend seulement effet lorsque le skybox est désactivé)



Ces valeurs sont stockées dans des variables membres de la classe *Renderer* et utilisées lors du dessin.

2.3 Primitives vectorielles

L'utilisateur peut dessiner cinq types de primitives vectorielles. Pour ce faire, on commence par sélectionner la forme voulue dans la liste déroulante *Forme à dessiner* de la fenêtre *Options de dessin*. Ensuite, on ajuste les différents paramètres décrits dans la fonctionnalité 2.2. Finalement, on clique à des endroits libres de la fenêtre où on veut placer un des sommets de la primitive à dessiner. Tous les exemples suivants sont faits avec remplissage jaune, bordure noire, bordure de taille 3:

- Lorsque *Point* est sélectionné, on clique à un emplacement et un point y sera dessiné. Sa taille et sa couleur sont déterminées par la couleur et l'épaisseur des bordures. Dessiné à l'aide de GL_POINTS. 
- Lorsque *Ligne* est sélectionné, on clique à deux emplacements pour dessiner une ligne entre ces deux points. Les lignes ne sont pas affectées par la couleur de bordure. Dessiné à l'aide de GL_LINES 
- Lorsque *Triangle* est sélectionné, on clique à trois emplacements pour indiquer les emplacements des trois sommets. Dessiné à l'aide de GL_TRIANGLES. 
- Lorsque *Rectangle* est sélectionné, on clique à deux emplacements pour indiquer deux sommets opposés du rectangle. Les deux autres sommets sont extrapolés. Dessiné à l'aide de GL_TRIANGLE_FAN. 

- Lorsque *Quad* est sélectionné, on clique à quatre emplacements, en sens horaire ou anti-horaire (au choix) pour indiquer les quatre sommets du quadrilatère à dessiner. Dessiné à l'aide de `GL_TRIANGLE_FAN`.



L'essentiel du code servant à dessiner ces primitives peut être consulté dans les méthodes *Application::mainLoop()* (pour la réception des événements de clic) et *Renderer::ajouterPtDessin()*.

2.4 Formes vectorielles

L'application permet à l'utilisateur de dessiner deux formes vectorielles différentes: le bonhomme sourire (*Smiley*) et l'astérisque (*Étoile*). La marche à suivre pour les dessiner est la même que pour les primitives vectorielles. Les exemples suivants utilisent les mêmes paramètres de dessin que ceux de la fonctionnalité précédente (remplissage jaune, bordures, noires, bordures de taille 3):

- Lorsque *Smiley* est sélectionné, on clique à deux emplacements pour indiquer deux sommets opposés du visage du bonhomme. Les yeux et la bouche sont de la couleur des bordures. Chacune des quatre primitives composant cette forme (tête, oeil gauche, oeil droit, bouche) sont dessinées à l'aide de `GL_TRIANGLE_FAN`.
- Lorsque *Étoile* est sélectionné, on clique à deux emplacements pour donner les coordonnées de l'une des deux lignes diagonales. Les coordonnées des autres lignes sont extrapolés. La couleur de remplissage n'a aucun effet. Chacune des quatre lignes sont dessinées à l'aide de `GL_LINES`.



L'essentiel du code de dessin de ces formes se retrouve dans *Renderer::ajouterSmiley()* et *Renderer::ajouterEtoile()*, en plus des deux mêmes méthodes que celles utilisées pour le dessin de primitives.

2.5 Interface

L'interface graphique gérée par ImGui est utilisée pour accéder à presque toutes les autres fonctionnalités de l'application. Elle est composée de cinq fenêtres flottantes ancrées aux bordures de la fenêtre principale de l'application. De plus, chacune des fenêtres peut être réduite en cliquant sur la petite flèche en haut à gauche de la fenêtre en question. Ces cinq fenêtres sont:

- *Options de dessin* qui permet de régler les paramètres de dessin vectoriel, de modifier le curseur à utiliser, de choisir si on veut utiliser le skybox ou une couleur fixe d'arrière-plan, ainsi qu'un bouton permettant d'accéder à la fonction de génération de texture procédurale.
- *Importer / Exporter* qui permet d'importer des images et des modèles, exporter des captures d'écran, ainsi qu'accéder à la fonction de composition d'image.
- *Échantillonnage d'image* qui permet d'accéder à la fonction du même nom.

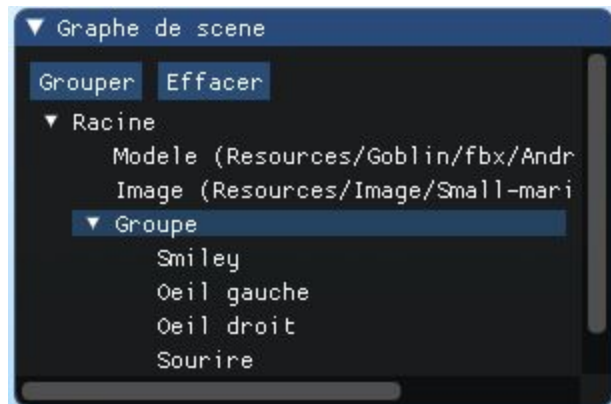
- *Graphe de scène* qui affiche les différents éléments présents dans la scène en se moment, sous forme d'arborescence.
- *Transformations* (seulement visible lorsqu'un élément est sélectionné dans le graphe de scène) qui permet d'appliquer dynamiquement des transformations aux différents éléments de la scène qui sont sélectionnés dans le graphe de scène.

Les captures d'écran sont ici intentionnellement omises, puisque celles-ci sont déjà disponibles dans les descriptions des autres fonctionnalités. L'essentiel du code de l'affichage du GUI est consultable dans *Renderer::drawGUI()*.

3.1 Graphe de scène

Le graphe de scène côté architecture est composé d'un *GroupObject* qui permet de conserver des pointeurs intelligents d'*AbstractObject*. Cette classe étant elle-même un *AbstractObject*, elle peut contenir des références vers d'autres *GroupObjects* afin de former une structure de type arbre n-aire. Lors de l'appel de la fonction *draw*, l'évènement est délégué à tous les objets de sa liste interne. Il est aussi possible d'effacer, ajouter ou obtenir un objet du tableau avec les fonctions implémentées dans cette classe.

Du côté GUI, le graphe de scène est dessiné dans la fenêtre *Graphe de scène* à l'aide de la méthode récursive *Renderer::drawTreeRecursive()*. Au départ, les groupes sont fermés. Pour afficher leur contenu, il suffit de cliquer sur la petite flèche à gauche du nom du groupe. Lorsqu'on clique sur l'un des éléments, il devient sélectionné. Lorsqu'un ou plusieurs éléments sont sélectionnés, on peut appuyer sur la touche Suppr. du clavier ou le bouton *Effacer* du GUI pour retirer ces éléments de la scène. On peut également mettre plusieurs éléments sélectionnés dans un même groupe avec le bouton *Grouper*. Finalement, pour ajouter des nouveaux éléments, il faut procéder avec les diverses façon de créer des nouveaux objets (par exemple, importer une image, importer un modèle ou dessiner une primitive vectorielle.

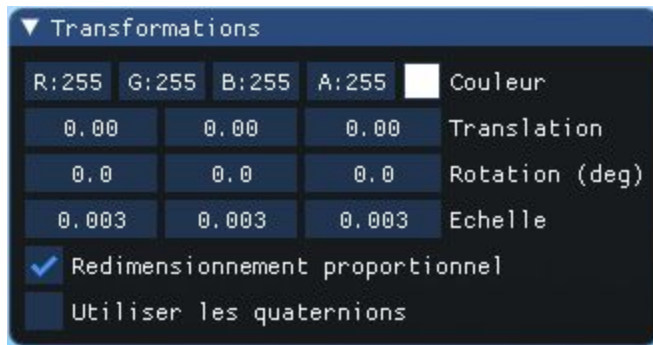


3.2 Sélection multiple

Dans le graphe de scène, il est possible de sélectionner plusieurs éléments simultanément en maintenant appuyée la touche Ctrl du clavier avant de cliquer sur les éléments à sélectionner. Cette opération peut servir à effacer plusieurs éléments en même temps ou encore à les grouper. De plus, lorsque plusieurs éléments sont sélectionnés, les transformations interactives (décrites dans la prochaine fonctionnalité) sont appliquées à chacun des éléments sélectionnés.

3.3 Transformations interactives

Les transformations interactives sont effectuées à l'aide de l'interface graphique, plus précisément, la fenêtre *Transformations*. Afin d'accéder à cette fenêtre, assurez-vous qu'au moins un élément est sélectionné dans le graphe de scène. Une fois qu'un objet est sélectionné, il suffit de modifier ses paramètres. Pour ce faire, on peut double cliquer sur un champ d'édition et entrer manuellement la valeur souhaitée au clavier, ou encore cliquer sur un des champs et, tout en maintenant le bouton appuyé, déplacer la souris vers la droite pour augmenter la valeur ou vers la gauche pour la réduire.



Il est aussi possible d'appliquer les transformations à plusieurs objets en utilisant la sélection multiple ou en sélectionnant un groupe (tous les éléments du groupe seront alors affectés par les transformations). Dans le code, cette partie est implémentée de façon très simple en profitant des propriétés du polymorphisme. Un seul appel aux méthodes de *AbstractObject* *setPosition()*

(ou *addPosition()*), *setRotation()* (ou *addRotation()*) et *setScale()* (ou *addScale()*) permettent de modifier les transformations des objets dans la scène.

De plus, deux cases à cocher sont disponibles. L'une sert à sélectionner si on veut que les modifications à l'échelle de l'objet soient appliquées de façon proportionnelle en xyz ou individuellement. L'autre sert à activer le mode d'édition de la rotation par quaternions.

3.4 Historique de transformation

Non implémenté.

3.5 Système de coordonnées

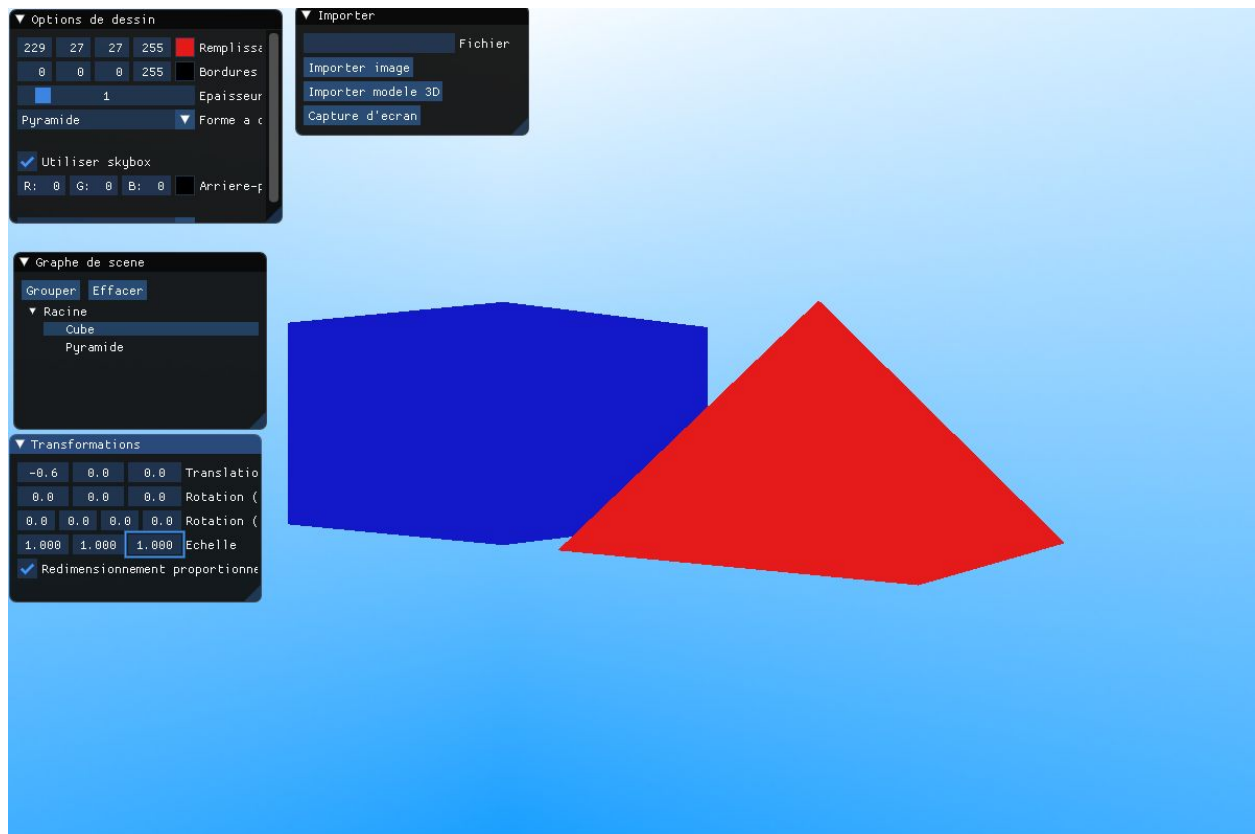
Non implémenté.

4.1 Boîte de délimitation

Non implémenté.

4.2 Primitives géométriques

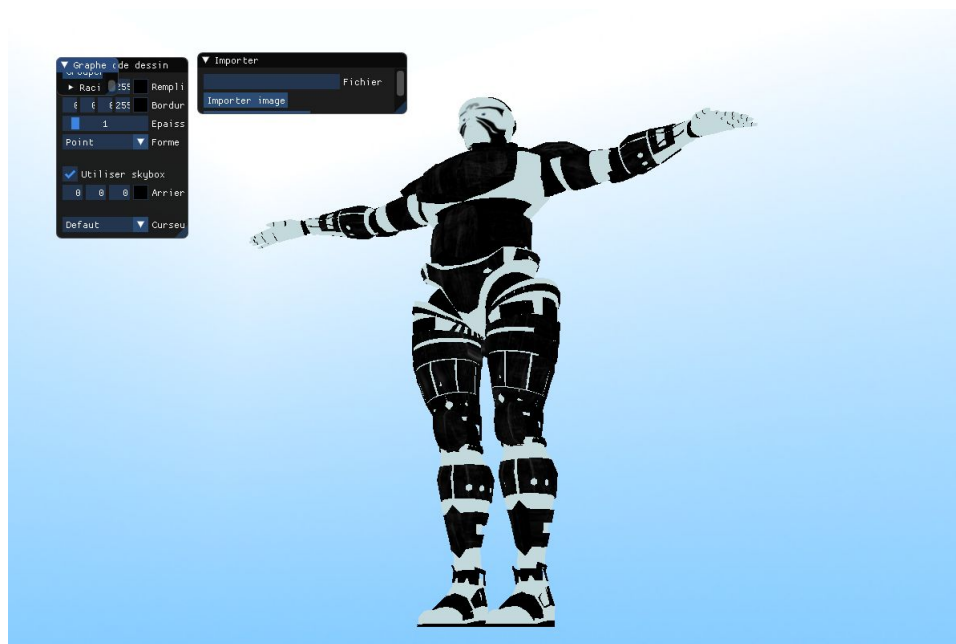
Pour les deux primitives géométriques demandées, nous avons choisi d'offrir le cube et la pyramide à base carrée à l'utilisateur. Comme pour les primitives vectorielles, il est possible de choisir la couleur de la forme avec le sélecteur de couleur de remplissage. On clique ensuite à un emplacement libre de l'écran pour que la primitive soit générée aux coordonnées (0,0,0). Ces formes géométriques peuvent être déplacées, tournées et modifiées en taille avec le menu de transformation. Le code permettant d'effectuer le dessin de ces formes se situe respectivement dans les classes CubeObject et SBPyramidObject. Le principe derrière leur affichage est de dessiner des triangles en fonction des points de coordonnées des sommets.



4.3 Modèle 3D

Pour importer un modèle, il suffit d'entrer le chemin du fichier dans le champ *Fichier* de la fenêtre *Importer / Exporter*. Puis, dans la même fenêtre, on appuie sur le bouton *Importer modèle 3D*.

Côté code, le chargement de modèles 3D est fait à partir d'Assimp. Le code de cette fonctionnalité se situe principalement dans *Model/Object*, *Model* et *Mesh*. *Model/Object* représente l'objet du monde virtuel, cette classe contient des informations telles que l'échelle, la rotation et la position de l'objet. *Model* se charge de la lecture des informations du fichier modèle et de transférer ces informations à *Mesh*. *Mesh* permet le rendu de cet objet grâce aux pointeurs qu'il initialise lors de l'appel de *setupMesh()* qui lui permettront de passer les vertices, normales et textures au shaders lors de l'appel de *Draw()*. Les modèles sont dessinés avec des GL_TRIANGLES.



4.4 Animation

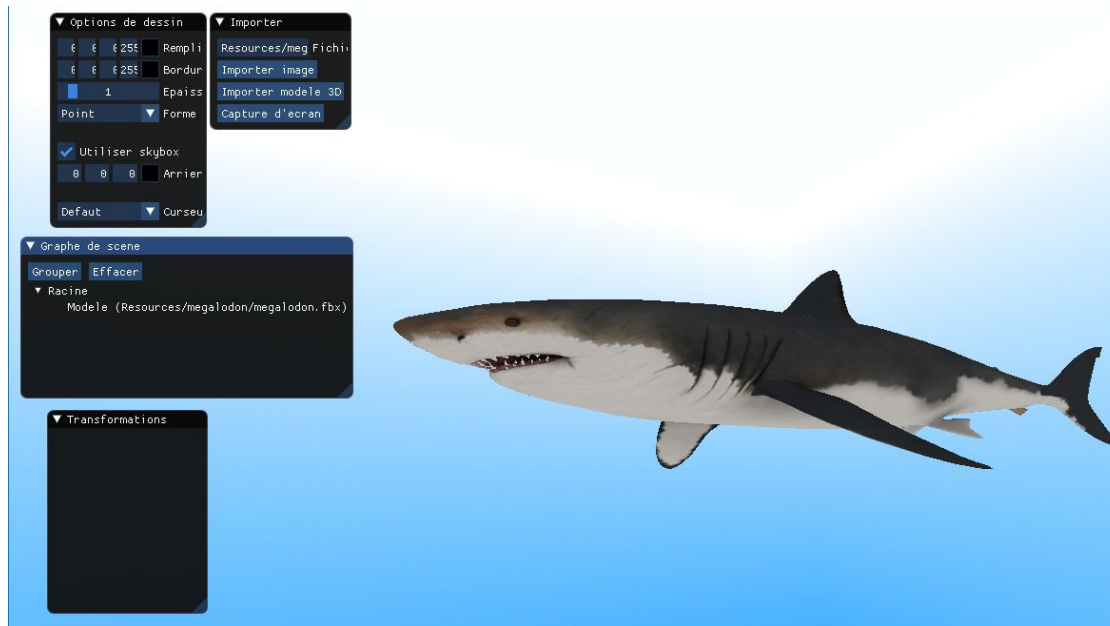
Non implémenté.

4.5 Instanciation

Non implémenté.

5.1 Mapping

Le mapping autre que les quadrilatères et le skybox est principalement fait sur les modèles 3D importés. La partie qui lie la texture à un buffer est située dans la classe *Model*, tandis que le pointeur vers les coordonnées de textures est situé dans *Mesh*. Lors d'un appel à la fonction de dessin, *Model* ne fait que transférer l'appel au *Mesh*. Un bel exemple représentant le mapping est le requin disponible dans les ressources. Comme *Assimp* se charge de lire les indices, il nous sauve énormément de temps pour cet objectif fonctionnel.



5.2 Composition

Cet objectif fonctionnel est représenté par la composition d'une image choisie par l'utilisateur et un bruit de perlin. Celle-ci est instanciée sur un quadrilatère dans la scène lorsque que l'on appuie sur le bouton *Importer image pour composition* dans la section importer. Pour ce qui est du code, l'algorithme multiplie chacun des pixels de l'image par un bruit de Perlin pour donner une nouvelle texture. Voici un petit exemple où l'image de gauche devient celle de droite lorsque composé avec un algorithme de bruit de Perlin:



5.3 Traitement

Non implémenté.

5.4 Texture procédurale

La texture procédurale est représentée par un bruit de Perlin. Celle-ci est instanciée sur un quadrilatère dans la scène lorsque l'on appuie sur le bouton *Afficher texture PerlinNoise* dans la fenêtre *Options de dessin*. La texture est définie sur une surface où l'algorithme va modifier chacun des pixels de cette surface par un algorithme de bruit de Perlin multiplié par une couleur définie.



5.5 Cubemap

Le cubemap est représenté par le skybox présent dans la scène, celui-ci est instancié à partir d'images dégradées faites à la main. Il est entièrement possible de le désactiver à l'aide de la case à cocher *Utiliser skybox* dans la fenêtre *Options de dessin*. Pour le code, nous avons utilisé `GL_TEXTURE_CUBEMAP` avec 6 images que nous envoyons dans la texture (en réalité c'est 3, car tous les côtés sont pareils). Pour le reste, la majorité du code ressemble au dessin d'un cube normal. Voir à la page suivante pour le code.

```

GLuint SkyboxObject::loadCubemap(std::vector<char*> faces, GLint wrapS, GLint wrapT, GLint minFilter, GLint magFilter)
{
    GLuint textureID;
    SDL_Surface* image;
    GLint channels;
    GLenum type;

    //Générer la texture
    glGenTextures(1, &textureID);

    //Lier la texture à la cible texture 2D
    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);

    //wrapping
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
    //filtering
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

    for (GLuint i = 0; i < faces.size(); i++)
    {
        image = Model::loadImage(string(faces[i]));
        Model::getImageProperties(image, channels, type);

        //link the texture
        glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, channels, image->w, image->h, 0, type, GL_UNSIGNED_BYTE, image->pixels);
        //Free image
        SDL_FreeSurface(image);
    }

    glBindTexture(GL_TEXTURE_CUBE_MAP, 0); //Délie la texture pour éviter d'être corrompue avec une autre

    return textureID; //retourne l'objet texture créé
}

```

Ressources

La règle générale est que tous les fichiers du dossier Src sont **faits par nous** sauf pour ceux provenant de ImGui.

Voici quand même une liste non exhaustive:

- Main
- Application
- Renderer
- Scene
- AbstractObject
- GroupObject
- QuadObject
- CubeObject
- SkyboxObject
- PrimitiveObject
- ModelObject
- AbstractShader
- ShaderLoader
- Shaders du fichier Src
- CMakeList.txt
- PerlinNoise

Fait par nous, mais **grandement inspiré** de: <https://learnopengl.com/>

- Model
- Mesh

La matrice de perspective a été trouvée ici :

<http://www.geeks3d.com/20090729/howto-perspective-projection-matrix-in-opengl/>

Ressources externes:

ImGui : <https://github.com/ocornut/imgui>

SDL: <https://www.libsdl.org/>

Glew: <https://github.com/nigels-com/glew>

Cmake : <https://cmake.org/>

Assimp: <https://github.com/assimp/assimp>

GLM: <https://glm.g-truc.net/0.9.8/index.html>

Présentation

Mathieu Dubreuil fait un bac en informatique. Il adore l'intelligence artificielle et aimerait éventuellement aller faire de la recherche dans ce domaine. Il aime bien apprendre de nouvelles choses. Prêt à tester les éventuels livrables qui lui sont proposés par son équipe, il s'adapte assez rapidement au domaine assez complexe qu'est l'infographie sans expérience préalable dans le milieu.

Samuel Baillargeon est à sa 2e session du bac en informatique (IFT). Il a suivi le parcours DEC-BAC offert en collaboration entre le Cégep de Lévis-Lauzon et l'Université Laval. Au Cégep, il a eu l'occasion de faire beaucoup de C++ et a également suivi un cours d'infographie utilisant C++ et OpenGL durant son DEC en informatique industrielle. Il a eu la chance de travailler chez Creaform (scanneurs 3D) lors d'un de ses stages et apprécie grandement le domaine, ainsi que les mondes virtuels. En accomplissant ce cours, il espère poursuivre avec le cours programmation de jeux vidéo, puisque c'est un domaine qui le passionne et dans lequel il aimerait un jour travailler.

Alexandre Mercier-Aubin est diplômé du DEC technique en informatique industrielle et poursuit son parcours scolaire à l'Université en IFT. Il souhaite principalement bâtir sur les connaissances techniques du cégep afin de lui donner une base théorique forte pour le préparer au marché du travail. Il aura la chance de faire un stage chez Activision dans le département Engine l'été prochain et considère le cours comme une bonne préparation aux éventuels défis du domaine.