Document de design

Présenté à M. Philippe Voyer

Université Laval
IFT-3100 Infographie

Équipe 04:
David D'Anjou
Johny Savard
Alexis Roberge
Vincent Laverdure

7 mars 2025

Table des matières

Sommaire
Interactivité
Technologie
Compilation
    Télécharger et installer openFrameworks
Architecture
Fonctionnalités
    Module 1: Image:
        1.1 Importation d’image
        1.2 Exportation d’image
        1.4 Espace de couleur
        1.5 Histogramme:
    Module 2: Dessin Vectoriel
        2.2 Outil de dessin
        2.3 Primitives vectorielles
        2.5 Interface
    Module 3:Transformation
        3.1 Graphe de scène
        3.2 Sélection multiple
        3.3 Transformations interactives
        3.4 Historique de transformation
    Module 4: Géométrie
        4.1
        4.2
    Module 5: Caméra
        5.1 Caméra interactive
        5.2 Mode de projection
Ressources
Présentation
Déclaration globale de l’usage de l’intelligence artificielle:

Sommaire

Ce projet a pour objectif de développer une application interactive permettant à l'utilisateur de créer, manipuler et modifier divers éléments graphiques en 2D et en 3D, incluant des primitives, des formes vectorielles et des images. L'application repose sur openFrameworks en C++, une bibliothèque adaptée aux applications graphiques et interactives.
L'idée initiale était de concevoir un outil de test de performances pour les cartes graphiques, mais le projet a évolué vers un éditeur graphique, permettant de travailler sur des scènes combinant des éléments 2D et 3D.

L'application est divisée en deux volets :

- Mode 2D
    - Dessin de primitives vectorielles (ligne, rectangle, ellipse, etc.).
    - Importation et modification d'images (ajustement des couleurs en RGB et HSB).
    - Sélection et modification des objets vectoriels dans la scène.
- Mode 3D
    - Affichage d'une scène 3D avec contrôle de la caméra.
    - Importation d'objets 3D.
    - Création et manipulation d'éléments géométriques en 3D

Interactivité

Pour l'import d'images, il suffit simplement de prendre un fichier png externe à l'application et le glisser dans la fenêtre. Il est ensuite possible de bouger l'image à l'intérieur de la fenêtre pour la positionner, simplement en cliquant sur l'image et en la glissant à la position voulue. Lors de l'import d'une image, l'application ouvre automatiquement une seconde fenêtre pour représenter l'histogramme de l'image. Pour cesser de le visualiser, il suffit de fermer la fenêtre.

Pour l'export, il suffit de prendre un imprime écran(screenshot) avec la touche g sur le clavier pour qu'une copie de la fenêtre actuelle soit exportée sous png dans le fichier data de l'application. Pour l'export d'une image sélectionnée, simplement sélectionner l'image et cliquer sur le bouton export.
Pour modifier les attributs RGB et HSB d'une image importée, il suffit de sélectionner l'image avec le clic gauche de la souris, puis de modifier les valeurs dans l'interface en haut à gauche.

Pour le dessin 2D, nous utilisons l'interface pour sélectionner la primitive vectorielle voulue (dans le menu en haut à gauche de la fenêtre) puis nous dessinons avec le clic gauche de la souris pour former la primitive voulue. En cours de dessin, il est possible d'annuler en cliquant sur le bouton droit de la souris. Pour que l'utilisateur voit facilement ce qu'il dessine, le curseur prend la forme de la primitive en cours. L'utilisateur peut aussi modifier les paramètres voulus de sa primitive dans à l'aide de clic de souris dans l'interface, soit en choisissant la couleur de l'intérieur et du contour de sa primitive, en modifiant l'épaisseur de la ligne de contour ou en choisissant d'activer ou désactiver le remplissage ou le contour de sa primitive.

Pour notre graphe de scène, nous avons pris la décision de séparer le 2D du 3D pour faciliter le dessins en mode 2D. Pour passer d'un mode à l'autre il est possible de le faire avec en appuyant avec un clic gauche de la souris sur le bouton intitulé 2D sous la section Mode 2D/3D. Ensuite

une fois des dessins effectués il est possible de les sélectionner en cochant le “toggle” des dessins voulu sous la section “Liste d’objets”. Ils passeront ainsi au rouge pour indiquer qu’ils sont sélectionnés. Il est ensuite possible de supprimer les dessins sélectionnées en appuyant sur le bouton “Delete all Selected”. En sélectionnant un “toggle” d’un dessin il est aussi possible de le déplacer en modifiant les champs: “Pos 1 X”, “Pos 1 Y”, “Pos 2 X” et “Pos 2 Y” et finalement appuyer sur le button “Apply Transform”, le tout se situant dans la section “Transform selected 2D”.

Dans le mode 3D de l’application, il est possible de changer le mode de projection entre perspective et orthogonale en cliquant sur le bouton “change projection mode”. De plus, on peut bouger la caméra à l’aide du clic gauche et du clic de molette. La caméra bouge différemment si l’on clique à l’intérieur de la zone de délimitation ou à l’extérieur. Pour ajouter des formes dans la scène 3D, on commence par sélectionner une forme avec les boutons de gauche. Une fois la forme choisie, on peut faire un clic droit et l’objet sera dessiné à l’endroit du clic. Il y a 3 boutons pour sélectionner un import, le nom de ce bouton changera pour le nom du fichier import. Pour importer un fichier, on glisse le fichier dans la fenêtre. Les fichiers avec les extensions .glb et .obj sont acceptés. Il y a aussi les boutons “undo” qui recule dans les actions, puis “redo” qui avance dans les actions qu’on a reculées. Cette fonctionnalité fonctionne aussi en mode 2D. On peut sélectionner les instances créées dans la scène, elles seront alors dessinées en rouge. Le bouton “Delete all selected” permet de supprimer toutes les instances sélectionnées. Le bouton “toggle wireframe” nous permet de changer l’état de visibilité des boîtes de délimitation.

Technologie

L’application est développée en C++17 en utilisant openFrameworks version 0.12.0 qui est la dernière version disponible à ce jour le le site officiel d’openFramworks: [download | openFrameworks](). Ce framework est spécialisé dans la modélisation graphique et basé sur

OpenGL. Si ce projet était à recommencé nous aurions peut-être utiliser DirectX à la place, car beaucoup de notre temps était de chercher de la documentation désuète ou tout simplement inexistante.

Pour la gestion des interfaces utilisateur et l'affichage des formes en 2D et 3D, nous avons intégré deux bibliothèques essentielles : ofxGui, qui facilite la création d'interfaces graphiques interactives, et ofxAssimpModelLoader, permettant le chargement et le rendu de modèles 3D. Le projet s'appuie également sur la Standard Template Library (STL) de C++, notamment pour le stockage et la manipulation des formes graphiques et des données d'histogramme.

IDE:
Le développement a été réalisé sous Visual Studio 2022, car il est un des logiciels recommandé lors de l'utilisation d'openFramworks et que plusieurs d'entre nous l'avons déjà utilisé au moins une fois dans le passé. Avec du recule nous aurions peut-être opté pour Clion, car plusieurs d'entre nous avons éprouvé des difficultés lors des "pull" dans GitLab. Il semblerait que les dossiers .vs et obj créait beaucoup de problème et devions alors les supprimer à presque chaque modification d'un autre membre de l'équipe.

Gestionnaire de code:
La gestion du code source, des versions et des collaborations en équipe a été assurée via Git et GitLab qui est la seconde plateforme la plus utilisée pour le travail collaboratif. Comme cette dernière a été utilisée pour le cours GLO-2004 pour plusieurs d'entre nous, il semblait être le choix le plus logique pour ce projet.

Système d'exploitation:
Nous sommes tous sous plateforme x64 avec une combinaison de Windows 10 et Windows 11. Un des membres de l'équipe à changer de Mac OS à Windows 11 en cours de projet, car plusieurs problèmes de compatibilité se faisaient ressentir avec les nouvelles puces M4 d'Apple.

Compilation

Télécharger et installer openFrameworks

1. Rendez-vous sur le site officiel : https://openframeworks.cc/download/
2. Téléchargez la version correspondant à votre système d'exploitation (Windows, macOS ou Linux).
3. Extrayez l'archive dans un répertoire de travail (ex: C:/openFrameworks/ sous Windows).

Télécharger la dernière version de visual studio à cet endroit: https://visualstudio.microsoft.com/downloads/. Une fois installé, ouvrir un nouveau projet en utilisant l'option de cloner un référentiel en utilisant l'adresse suivante: https://gitlab.com/JohnySavard/infographie.git. Il est important de cloner le projet dans le fichier d'installation d'openframeworks à cet endroit: \openframeworks\apps\myApps/.

Si vous n'utilisez pas le référentiel, simplement télécharger le projet fourni et le déposer au même endroit que mentionné précédemment.

Par la suite, ouvrez le project generator d'openframeworks: \openframeworks\projectGenerator/projectGenerator.exe. Il suffit de créer un nouveau projet en sélectionnant 2 addons supplémentaires: ofxAssimpModelLoader et ofxGui.En cliquant sur generate, le projet est lancé dans visual studio, ou il faudra supprimer les fichiers ofapp.h et ofapp.cpp. (il peut être nécessaire de supprimer le ficher .vs qui est caché dans le répertoire, selon votre configuration). Finalement il suffit d'exécuter l'application dans visual studio!

Architecture

Notre application repose sur une structure de base, largement inspirée des laboratoires du cours, et adopte une forme simplifiée du modèle MVC (Modèle-Vue-Contrôleur). L'organisation du projet repose sur plusieurs classes ayant chacune un rôle distinct dans le fonctionnement global du programme.

La classe Application joue le rôle de contrôleur principal, en assurant la gestion des événements, des interactions utilisateur et de la logique générale du programme. Elle orchestre le comportement des autres modules et leur permet de communiquer entre eux.
La classe Renderer est responsable du rendu graphique, traitant à la fois l'affichage des formes vectorielles et des images importées. Elle permet d'appliquer différentes transformations et ajustements aux objets affichés à l'écran.

La classe HistogramWindow est dédiée à l'analyse des images. Elle génère un histogramme des couleurs d'une image sélectionnée et le met à jour en fonction des modifications effectuées par l'utilisateur. Cet histogramme offre une représentation visuelle de la répartition des couleurs et permet une meilleure compréhension des ajustements appliqués.

La classe PortalWindow est chargée de gérer un portail offrant une vue alternative sur la scène. Elle permet d'afficher un point de la scène sous une perspective différente en utilisant une seconde caméra, offrant ainsi une fonctionnalité supplémentaire pour l'exploration et la manipulation d'objets dans l'espace de travail.

Par manque d'expérience avec openFrameworks, l'architecture du projet n'a pas été découpée de manière optimale. Les classes restent volumineuses et regroupent plusieurs responsabilités, ce qui pourrait être amélioré en affinant la séparation des modules et en suivant une approche plus

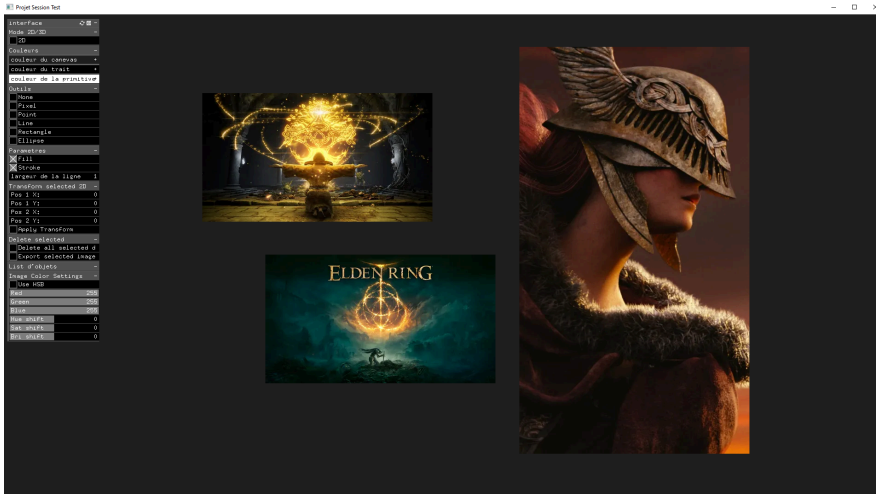
rigoureuse du génie logiciel. Un découpage plus structuré permettrait de rendre le projet plus maintenable et plus flexible pour la seconde partie du travail.

Fonctionnalités

Module 1: Image:

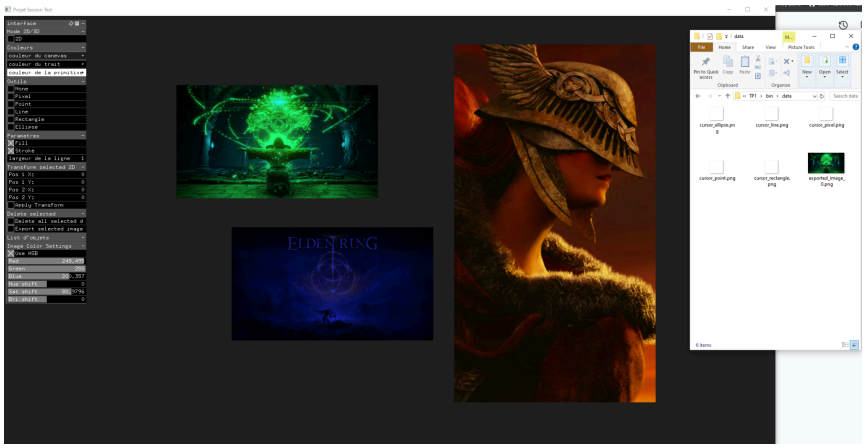
1.1 Importation d'image

Il est possible d'importer une ou plusieurs images, et de les placer en arrière plan de notre scène. Une fois l'image importée, il est possible de la glisser partout dans la scène. Il est possible d'importer une seule ou plusieurs images à mettre dans la scène 2D et de les placer comme bon nous semble dans la scène:



1.2 Exportation d’image

Il est possible d’exporter une image de la scène entière (imprime-écran) par la touche g. Il est aussi possible d’exporter une image en particulier dont le rendu a été modifié (RGB et HSB) pour l’exporter dans le fichier bin/data par exemple:

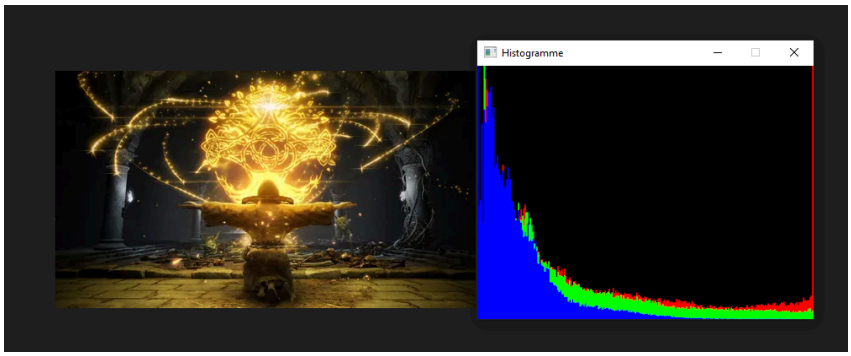


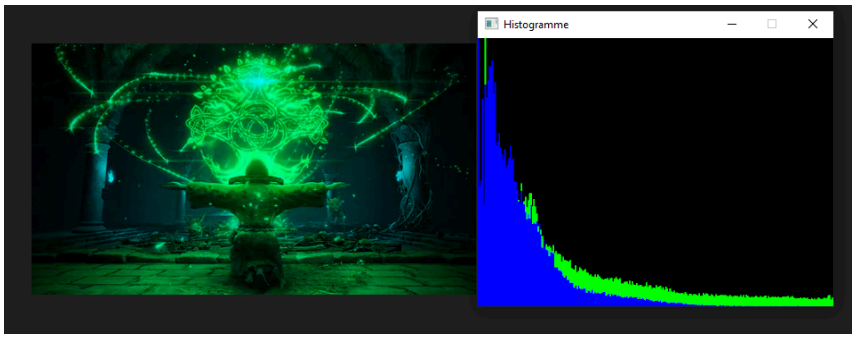
1.4 Espace de couleur

Il est possible de modifier les niveaux de RGB et HSB des images importées en utilisant le slider de RGB er de HSB de l'image, situé dans le menu de gauche. Le bouton HSB doit être sélectionné. Cela met à jour l'histogramme automatiquement et il est possible d'exporter cette image. l'image de la section précédente (1.2) présente des images dont le contenu a été modifié en comparaison avec la section 1.1.

1.5 Histogramme:

Lors de l'import d'une image, il est possible de visualiser son histogramme de couleurs dans une fenêtre distincte. L'histogramme est mis à jour pour refléter la dernière image importée ou dont les attributs ont été modifiés en dernier. Il est possible de simplement fermer l'histogramme si on ne désire plus le voir. Voici 2 exemples montrant un histogramme différent pour la même image selon la modification de ses attributs:



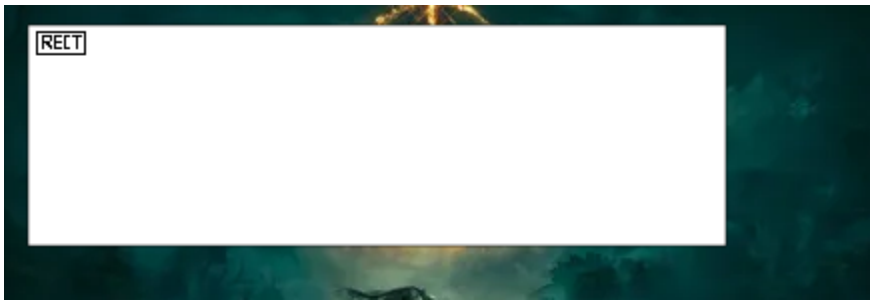


L'IA notamment CHAT gpt a été utilisée pour debugger, pour comprendre les erreurs d'ouverture de fenêtre et comprendre le fonctionnement global d'un histogramme.

Module 2: Dessin Vectoriel

2.1 Curseur dynamique

Lorsque nous sommes en mode de dessin 2D, le curseur prends la forme de la primitive vectorielle actuellement sélectionnée. Par exemple un cercle graphique avec une abréviation pour que l'usager voit dans quel mode de dessin il se trouve. Nous avons nous-même dessiné les images du curseur à l'aide du logiciel:https://www.piskelapp.com/p/create/sprite. En voici un exemple:



2.2 Outil de dessin

On peut modifier l'outil de dessin à l'aide de l'interface fournie dans notre application. On peut modifier l'épaisseur de la ligne de contour de chaque primitive (d'une épaisseur de 1 à 10), la couleur de l'intérieur de la primitive ainsi que la couleur de la ligne de contour de la primitive. Il est aussi possible de choisir d'activer ou désactiver le remplissage et le contour de l'outil de dessin pour permettre à l'utilisateur de choisir si la primitive qu'il veut dessiner doit avoir une ligne de contour ou doit être remplie.

2.3 Primitives vectorielles

On donne le choix parmi 5 types de primitives vectorielles que l'utilisateur peut dessiner selon la largeur et hauteur qu'il désire : on peut dessiner un polygone (hexagone), une ligne simple, un point (avec un rayon fixe) , un rectangle et une ellipse. Chaque type de primitive possède sa propre couleur et épaisseur qui est modifiable avant de dessiner celle-ci. Lors du dessin d'une primitive autre que le point, une zone correspondant au type de primitive s'affiche pour montrer à l'utilisateur un aperçu de la taille de la primitive qu'il s'apprête à dessiner.

2.5 Interface

On utilise une interface qui permet de modifier les valeurs de plusieurs attributs, ainsi que de voir tous les objets de notre scène. Chaque mode (2D et 3D) possède sa propre interface qui se met à jour en temps réel lorsque l'utilisateur effectue une sélection ou modification de sa scène. Dans l'interface 2D, on retrouve l'outil de dessin et ses paramètres ainsi qu'un groupe qui permet d'effectuer des transformations sur nos dessins. Le mode 3D possède un menu qui permet de choisir l'objet 3D à ajouter dans la scène. Les 2 interfaces possèdent aussi deux boutons pour le undo/redo, ainsi que la liste de tous les objets dans leur scène respective et un bouton pour les sélectionner et les supprimer.

Module 3:Transformation

3.1 Graphe de scène

Pour notre graphe de scène nous avons opté pour 2 scènes séparées pour le 2D et le 3D pour faciliter le dessins en mode 2D et ainsi pouvoir faire des plus belle œuvre d'art dans ce mode. Le 2D à une structure de données VectorPrimitive qui est dans un vecteur, intitulé shapes dans le fichier renderer.h, ce qui nous permet d'ajouter, supprimer et même la sélection de ses dernières. Ceci est similaire pour le graphe de scène 3D avec une structure GeometricPrimitive et le vecteur shapes3D. Encore une fois cette structure nous permet l'ajout, la suppression et la sélection

d’éléments.

```
enum class VectorPrimitiveType { none, polygon, point, line, rectangle, ellipse };

enum class ActionType { add, remove, transform };

struct VectorPrimitive
{
    VectorPrimitiveType type;            // 1 * 4 = 4  octets
    float               position1[2];    // 2 * 4 = 8  octets
    float               position2[2];    // 2 * 4 = 8  octets
    float               stroke_width;    // 1 * 4 = 4  octets
    unsigned char       stroke_color[4]; // 4 * 1 = 4  octets
    unsigned char       fill_color[4];   // 4 * 1 = 4  octets
    bool                  filled = NULL;
    bool                  stroke = NULL;
};

enum class GeometricPrimitiveType { none, cube, pyramid, imported };

struct GeometricPrimitive
{
    GeometricPrimitiveType type;
    string name;
    bool selected;
    float               position[3];
    float               size;
    unsigned char       color[4];
    bool                show_wireframe;
    ofMesh mesh;
    int                 ordre;
};
```

```
VectorPrimitiveType draw_mode;
VectorPrimitive* shapes;

GeometricPrimitiveType draw_mode3D;
vector<GeometricPrimitive> shapes3D;
```

3.2 Sélection multiple

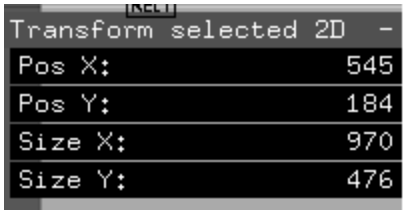
Il est possible de sélectionner plusieurs éléments en même temps de notre projet actuellement. Ceci peut alors permettre, entre autres, de faire la suppression de plusieurs dessins en même temps au besoin. Ceci à été réalisé en valider à un temps d'intervalle régulier (0.5sec) tous les dessins "toggle", ensuite on change la couleur de tous les éléments sélectionnés et ainsi faire une sélection multiple. Voici une partie du code qui permet de mieux visualiser:

```
if (is2DModeActive && nextTimeCheck < ofGetElapsedTimef() + intervalCheckTime) {
    nextTimeCheck = ofGetElapsedTimef() + intervalCheckTime;
    setColorSelected();
}
```

```
void Application::setColorSelected() {
    for (int i = 0; i < toggles.size(); i++) {
        if (toggles[i].get()->getParameter().toString() == "1") {
            renderer.shapes[i].fill_color[0] = 255;
            renderer.shapes[i].fill_color[1] = 0;
            renderer.shapes[i].fill_color[2] = 0;
            lastSelectedDrawing2D = i;
        }
        else {
            if (colors.size() > 0) {
                renderer.shapes[i].fill_color[0] = colors[i].get()->r;
                renderer.shapes[i].fill_color[1] = colors[i].get()->g;
                renderer.shapes[i].fill_color[2] = colors[i].get()->b;
            }
        }
    }
    renderer.draw();
}
```

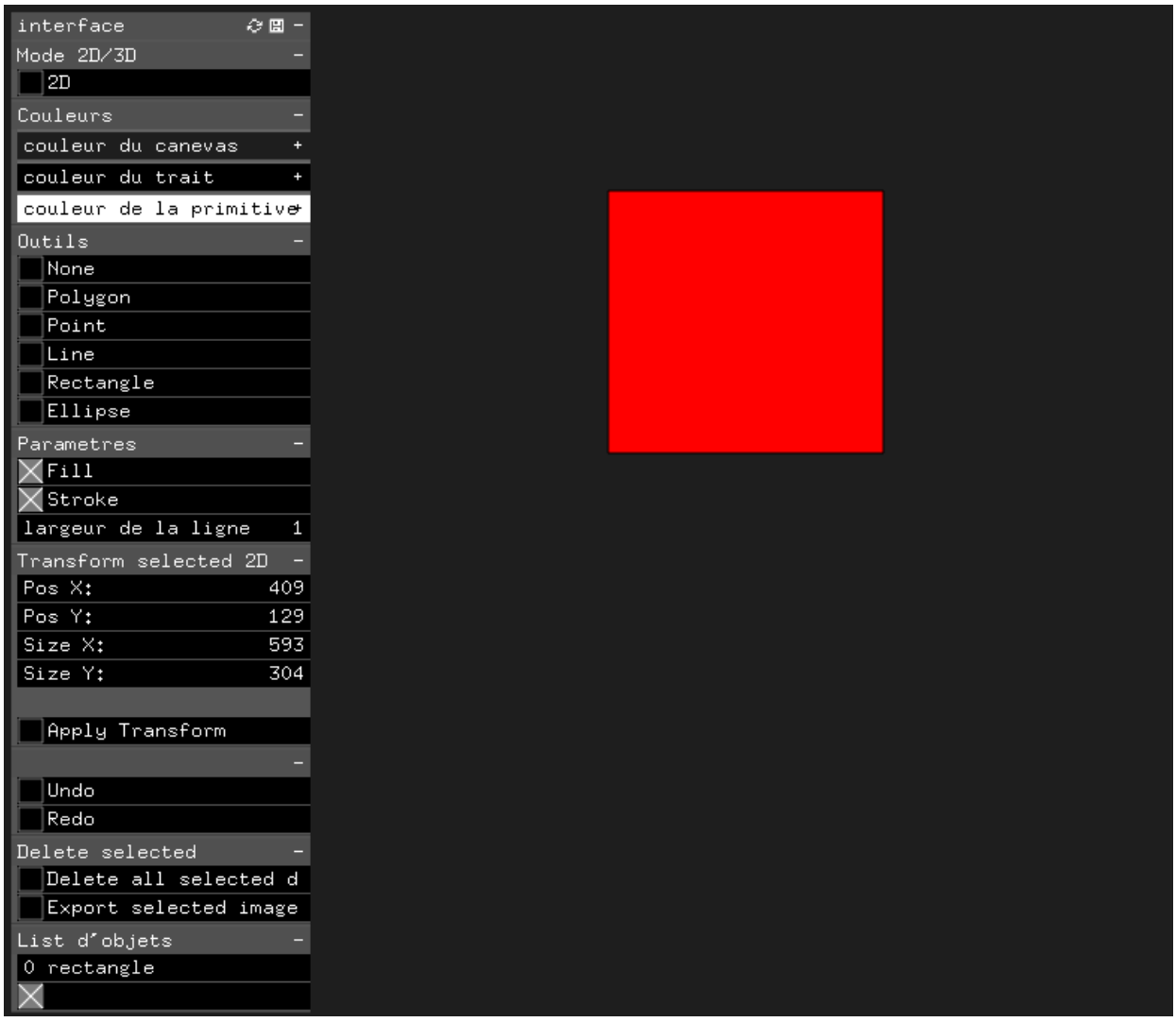
3.3 Transformations interactives

Il est également possible de faire plusieurs actions sur nos objets dans notre scènes tel que le déplacements. Effectivement, il est possible de déplacer nos dessins dans la scène avec la section suivante:

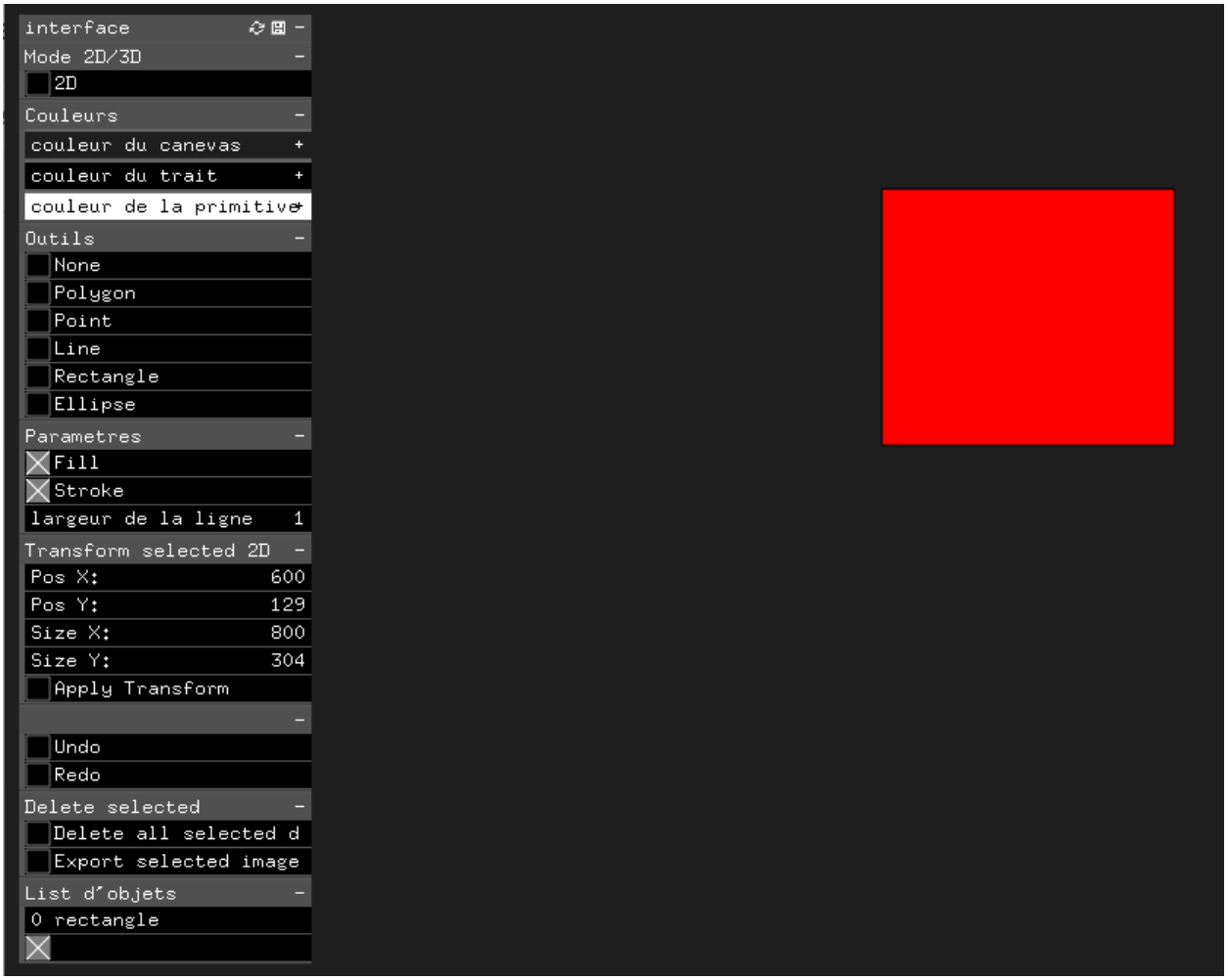


Ceci permet de déplacer les 2 points en X et Y utilisés pour la création du dessin. Il est alors possible de le déplacer.

Avant:



Après:



3.4 Historique de transformation

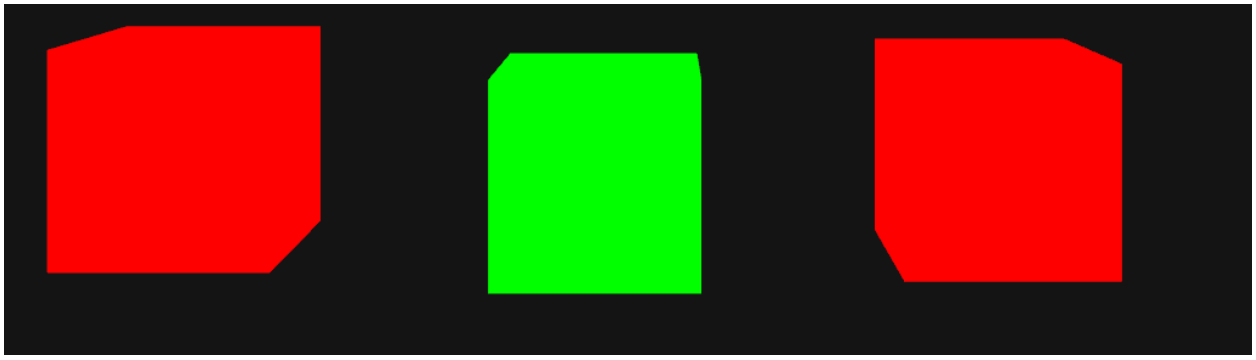
Il est possible d'annuler et de rétablir les transformations faites dans notre application (annuler l'ajout d'un objet va alors le supprimer). Lorsqu'une transformation est faite par l'utilisateur, celle-ci est enregistrée dans la pile 'undo', et lorsque ce bouton est cliqué dans l'interface, on annule la plus récente transformation, et on la place dans la pile 'redo'. Si l'utilisateur veut rétablir une transformation qu'il vient d'annuler, il peut cliquer sur le bouton redo pour faire cela.

Module 4: Géométrie

4.1 Boîte de délimitation

Un bouton de l'interface nous permet d'activer le mode boîte de délimitation qui nous permet de voir les délimitations de chaque objet 3D ajouté par l'utilisateur dans la scène. Pour ce faire, on utilise les boutons de sélection des objets 3D. Une fois la sélection terminée, on clique sur le bouton "Toggle wireframe" qui va altérer l'état visuel des boîtes de délimitations pour chacun des objets. Si la délimitation était active, elle serait désactivée et vice versa de manière indépendante entre chaque objet. Ceci marche avec n'importe quel mesh, incluant les objets importés.

Les cubes rouges sont sélectionnés.
Avant avoir appuyé sur "Toggle wireframe"



Après avoir appuyé sur "Toggle wireframe"

4.2 Primitive géométriques

L'application a déjà deux primitives géométriques créées par programmation, soit des cubes et des pyramides à base carré. Des points sont générés en se basant sur une taille fournie dans un objet GeometricPrimitive puis des triangles sont créés en reliant ces derniers. Les normales sont elles aussi générées. Aucune donnée externe au programme n'est utilisée.

```
ofMesh newMesh;
newMesh.clear();

// Add vertices to mesh
for (const auto& v : vertices) {
    newMesh.addVertex(v);
}

// Add indices to mesh to build triangles
for (const auto& i : indices) {
    newMesh.addIndex(i);
}

// Approximate normals
for (const auto& v : primitive3D.mesh.getVertices()) {
    newMesh.addNormal(normalize(v));
}

return newMesh;
```

Voici ce que GeometricPrimitive contient

```cpp
enum class GeometricPrimitiveType { none, cube, pyramid, imported };

struct GeometricPrimitive
{
    GeometricPrimitiveType type;
    string name;
    bool selected;
    float               position[3];
    float               size;
    unsigned char       color[4];
    bool                show_wireframe;
    ofMesh mesh;
    int                 ordre;
};
```

Voici comment sont créé les points des primitives

```cpp
case GeometricPrimitiveType::cube:
    vertices.push_back(glm::vec3(-size, -size, size));
    vertices.push_back(glm::vec3(size, -size, size));
    vertices.push_back(glm::vec3(size, size, size));
    vertices.push_back(glm::vec3(-size, size, size));
    vertices.push_back(glm::vec3(-size, -size, -size));
    vertices.push_back(glm::vec3(size, -size, -size));
    vertices.push_back(glm::vec3(+size, +size, -size));
    vertices.push_back(glm::vec3(-size, +size, -size));

    indices.insert(indices.end(), {
        // Front
        0, 1, 2, 2, 3, 0,
        // Right
        1, 5, 6, 6, 2, 1,
        // Back
        5, 4, 7, 7, 6, 5,
        // Left
        4, 0, 3, 3, 7, 4,
        // Top
        3, 2, 6, 6, 7, 3,
        // Bottom
        4, 5, 1, 1, 0, 4
        });
    break;
```

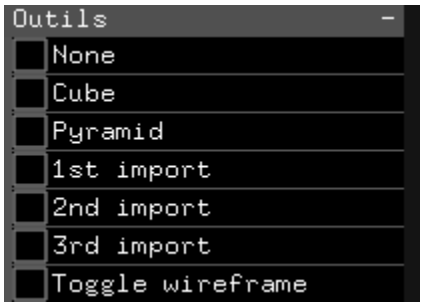
```cpp
case GeometricPrimitiveType::pyramid:
    vertices.push_back(glm::vec3(-size, -size, size));
    vertices.push_back(glm::vec3(size, -size, size));
    vertices.push_back(glm::vec3(-size, -size, -size));
    vertices.push_back(glm::vec3(size, -size, -size));
    vertices.push_back(glm::vec3(0, size, 0));

    indices.insert(indices.end(), {
        // Front
        0, 1, 4,
        // Right
        1, 3, 4,
        // Back
        3, 2, 4,
        // Left
        2, 0, 4,
        // Bottom
        2, 3, 1,
        1, 0, 2
        });
    break;
```

4.3 Modèles 3D

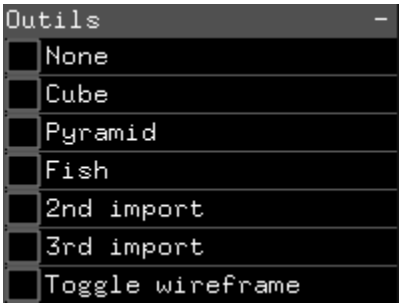
Il est possible d'importer 3 types de modèles 3D simultanément. Il suffit de glisser le fichier de ce modèle dans l'application. Si le modèle est supporté, alors un des emplacements de sélection

d’import aura son nom modifié pour le nom du fichier importé. Lorsqu’un 4e import est fait, il est conservé à la place du 1er, puisqu’il y a une limite d’import de 3.

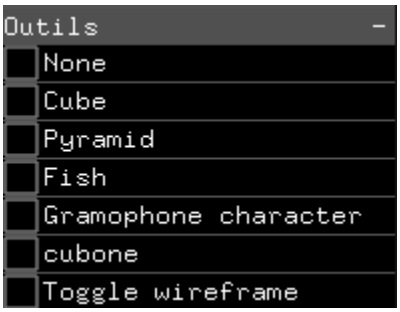
Avant l’import



Après l’import du premier

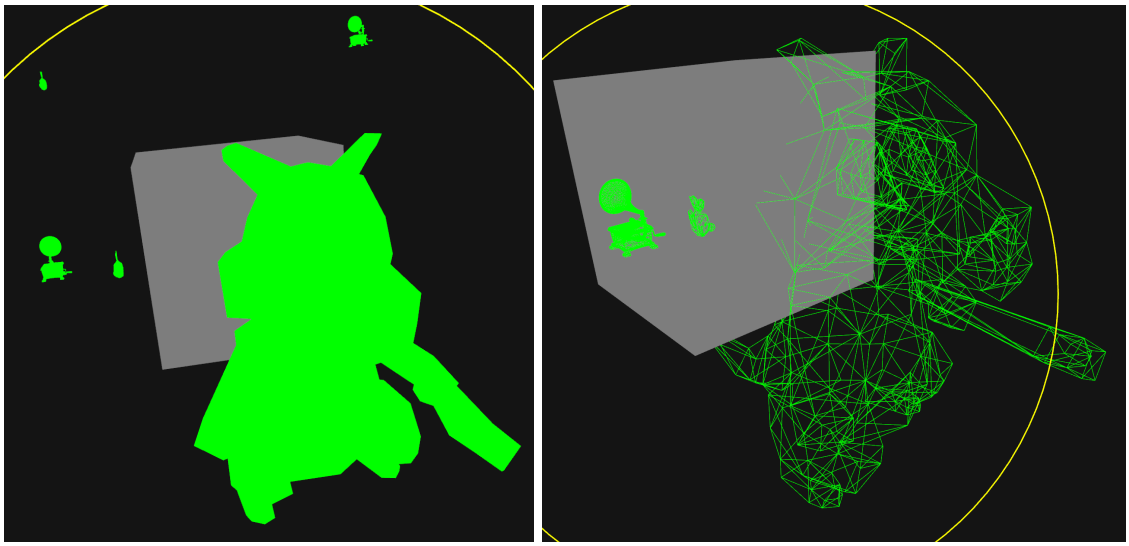


Après 3 imports



L’import utilise la grosseur originelle du modèle 3D, il se peut donc que certains modèles ont l'air petits dans la scène.

Voici un exemple d’une scène contenant 3 modèles 3D importés.



4.5 Instanciation

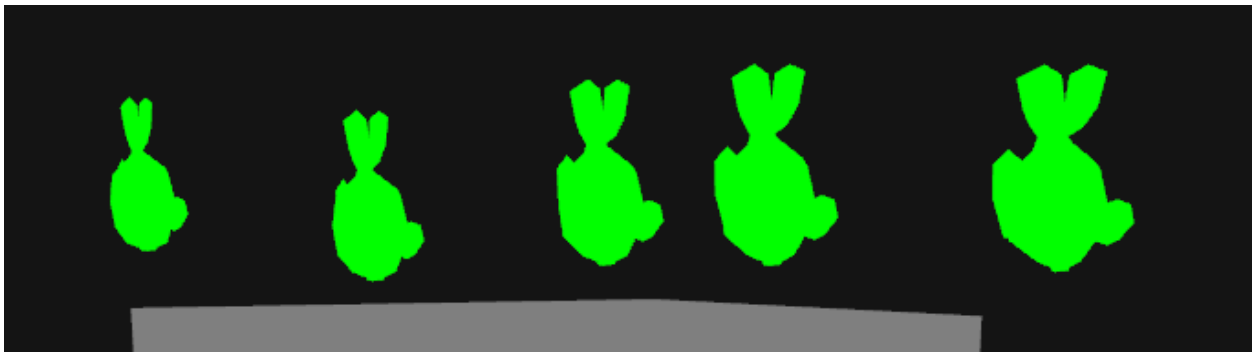
Comme démontré dans 4.5, il est possible d'importer différents objets dans la scène. Une fois les objets importés, une copie est conservée dans un tableau de “importedModel3d”.

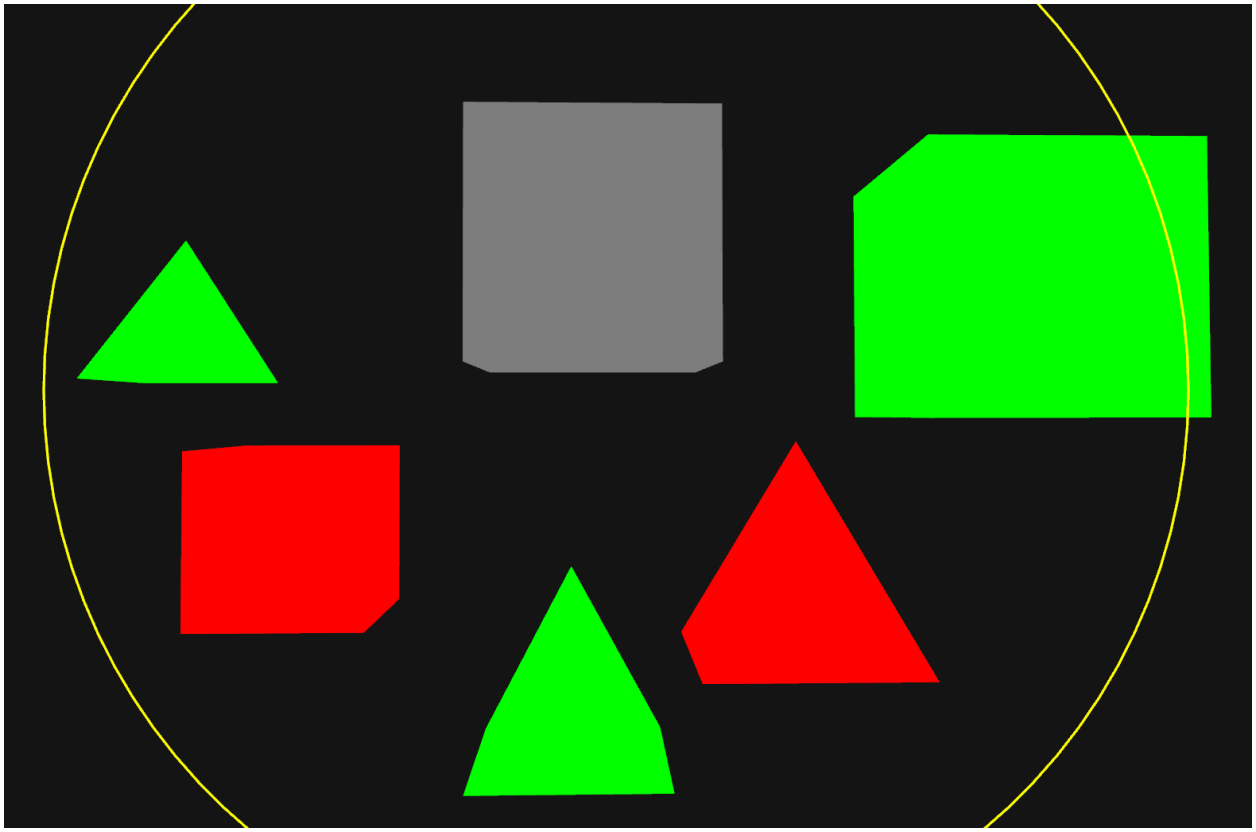
```
struct ImportedModel3D
{
    bool isEmpty;
    string name;
    ofMesh mesh;
};
```

On conserve donc une copie du mesh original. Lorsqu’on place une instance de ce dernier, il est converti en “GeometricPrimitive” de type “imported”. Il est actuellement seulement possible de changer la couleur lorsque le mesh est sélectionné, mais chacun des mesh des instances est indépendant. La grosseur du mesh dépend de l’état du bufferHead au moment de l’ajout.

Pour le démontrer, voici quelques exemples:

L’élément à gauche est mis lorsque le buffer est à 1.





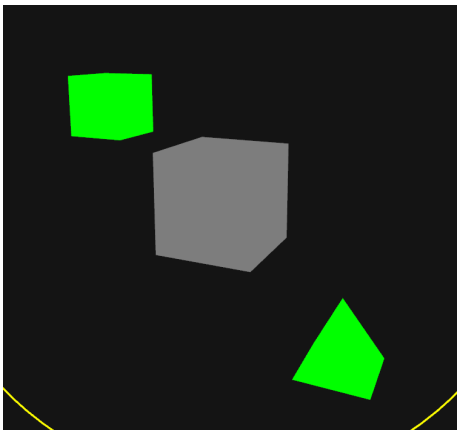
Module 5: Caméra

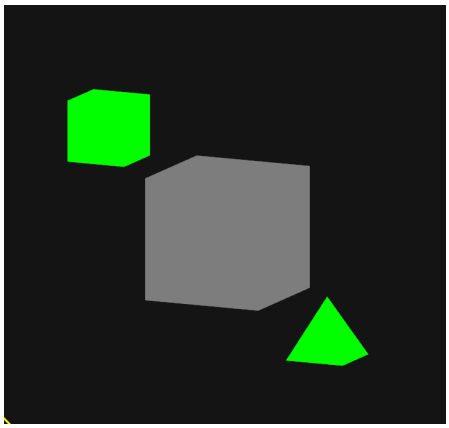
5.1 Caméra interactive

Dans le mode 3D de notre application, nous avons une caméra avec laquelle l'utilisateur peut interagir, ainsi qu'une zone de délimitation qui change le type de mouvement que la caméra effectue si l'on clique à l'intérieur ou l'extérieur de celle-ci. À l'intérieur de la zone, le clic gauche permet une rotation libre par rapport à l'origine, alors qu'à l'extérieur de la zone on effectue une rotation autour de l'axe Z. Le clic de la molette permet une translation en Y ou Z dans la scène.

5.2 Mode de projection

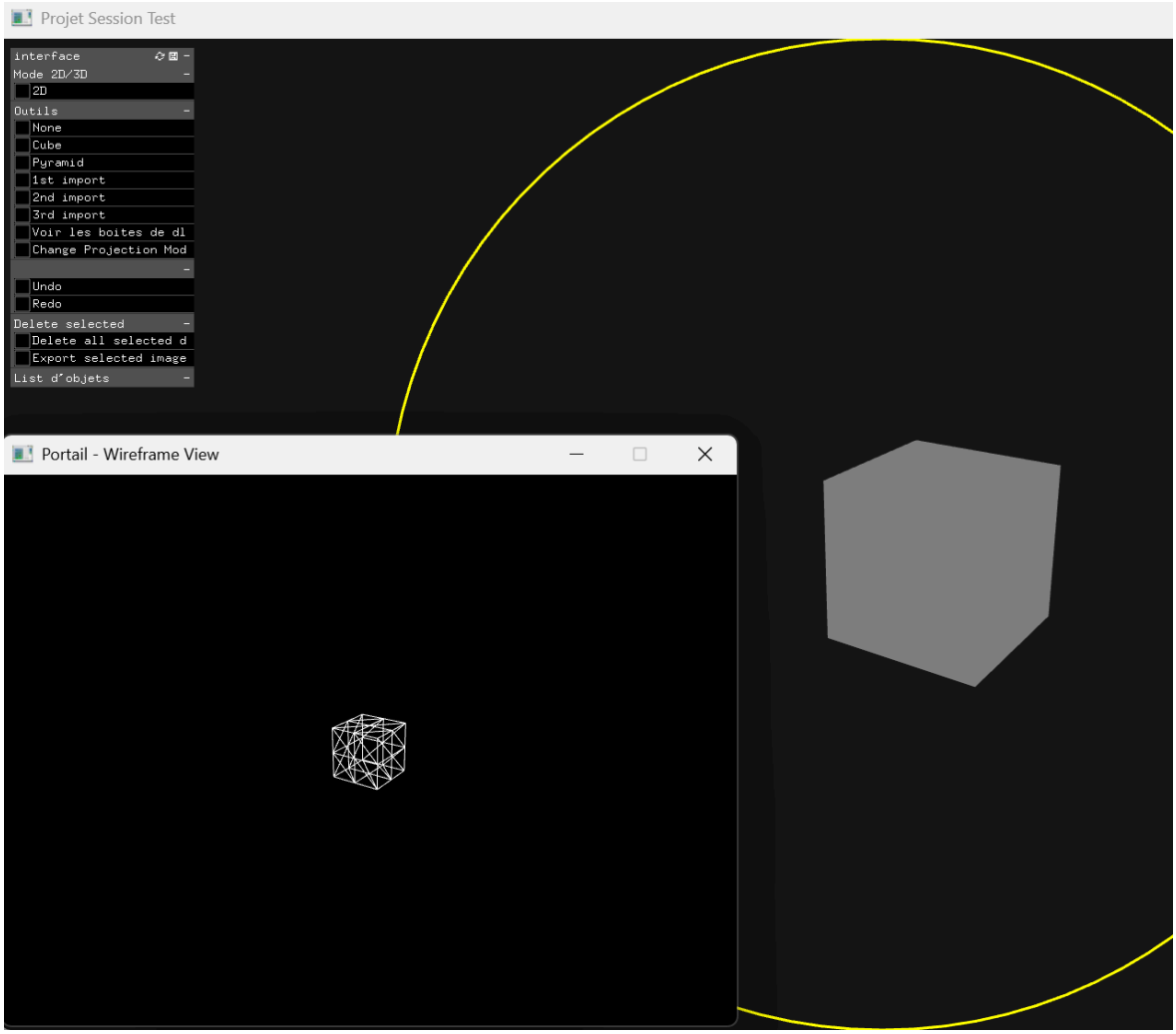
On peut modifier le type de projection lorsqu'on est dans le mode 3D. Par défaut, le mode de projection est la projection en perspective, mais on peut changer en projection orthogonale en cliquant sur le bouton de l'interface "Change projection mode". Ci-dessous un exemple d'une scène dans les 2 modes de projection (perspective à gauche et orthogonale à droite) :





5.5 Portail

Nous avons réalisé dans une seconde fenêtre, afin de ne pas encombrer la scène 3D, une seconde caméra qui prend l'image de la scène actuelle mais en "wireframe". Ceci peut faciliter le travail des objets 3D sans avoir besoin de passer la première caméra d'un mode à l'autre voici une image qui montre le résultat:



Ressources

Les images importées peuvent être n'importe quelles images, mais voici celles que nous avons utilisées pour le présent document: https://alphacoders.com/elden-ring-wallpapers .
Pour nos curseurs, nous avons utilisé le site suivant pour les dessiner par nous même:https://www.piskelapp.com/p/create/sprite.
Pour la prise de vidéo, nous avons utilisés l'outil OBS (open broadcaster software) disponible ici: https://obsproject.com/welcome

Modèles 3D
Gramophone: https://www.turbosquid.com/3d-models/gramophone-character-vintage-2317851
Cubone: https://poly.pizza/m/cc7gCdKaQYU
*Dans les captures d'écran et la vidéo, une version grossi du modèle cubone à été utilisée pour mieux montrer les boites de délimitations.
Poisson: https://poly.pizza/m/52s3JpUSjmX

Présentation

Tous ont participé de manière équivalente au projet, ont respecté les échéances et ont participé aux réunions. Voici les membres de l'équipe:

- David D'Anjou
- Johny Savard
- Alexis Roberge
- Vincent Laverdure

Bien que tous ont participés à toutes les sections, que ce soit en conseils ou en rétro-action, voici un compte-rendu global des efforts principaux de chacun:

1.1	Importation d'images	David
1.2	Exportation d'images	David
1.4	Espace de couleur	David
1.5	Histogramme	David
2.1	Curseur dynamique	David
2.2	Outils de dessin	Alexis
2.3	Primitives vectorielles	Alexis
2.5	Interface	Alexis
3.1	Graphe de scène	Johny
3.2	Sélection multiple	Johny
3.3	Transformations interactives	Johny
3.4	Historique de transformation	Alexis
4.1	Boîte de délimitation	Vincent
4.2	Primitives géométriques	Vincent
4.3	Modèles 3D	Vincent
4.5	Instanciation	Vincent
5.1	Caméra interactive	Alexis
5.2	Modes de projection	Alexis
5.5	Portail	Johny
	Document Design:	David
	Vidéo:	Tous
	Editing:	Johny

Déclaration globale de l’usage de l’intelligence artificielle:

L’intelligence artificielle a été utilisée principalement comme support pour le débogage, notamment pour interpréter les codes d’erreur et proposer des solutions lorsque nous étions bloqués. Elle a également servi de ressource pour orienter l’implémentation de certaines fonctionnalités, comme mentionné dans la section correspondante.

L’intégration de Copilot à Visual Studio 2022 a permis d’accélérer certaines parties du développement en suggérant automatiquement du code, en particulier pour les structures répétitives et les tâches redondantes. Toutefois, son utilisation est restée complémentaire aux outils classiques de l’IDE, sans remplacer notre travail d’analyse et d’implémentation manuelle des fonctionnalités spécifiques au projet.

Lien vers la vidéo:

https://youtu.be/tgWzb9-fsGc