UNIVERSITÉ LAVAL

IFT-3100: Infographie

Projet de session (TP#1)

présenté à

Philippe Voyer

matricule	*Nom*
111 010 662	Thomas Dixon
910 101 463	Alexandre Gagnon
111 147 051	Samuel Parent

Université Laval

11 mars 2018

Sommaire

L'objectif du projet de session est de développer une application qui permet de construire, éditer et rendre des scènes visuelles. Le projet de session a été divisé en deux parties. La première partie du projet porte sur les 5 catégories suivantes : l'image, les dessins vectoriels, les transformations, la géométrie et les textures. Chaque catégorie a 5 sous-catégories possibles et nous devons choisir d'en implémenter au moins 3 pour chaque catégorie. Pour notre projet, nous avons décidé d'implémenter les sous-catégories de manière rudimentaire pour commencer. Au fur et à mesure que le projet avance, nous pouvons ajuster les aspects de l'application pour être plus concentrés sur une idée précise. Avec cette méthode, nous nous laissons la porte ouverte à beaucoup plus d'implémentation possible des catégories, sans avoir à nous inquiéter de la manière qu'ils vont intégrer notre application. Une fois que plusieurs aspects vont être complétés, nous allons pouvoir décider où nous voulons aller avec les aspects suivant pour créer une application avec une idée et un but clairs.

Pour la première partie du projet, les aspects que nous avons décidé d'implémenter donnent beaucoup de liberté à l'utilisateur. Nous donnons beaucoup de contrôle à l'utilisateur pour qu'il soit capable de créer ce qu'il veut dans la scène. Avec les aspects que nous avons choisi de prendre, l'utilisateur peut dessiner à peu près ce qu'il veut dans la scène, il peut inclure des photos ou des modèles dans la scène et les modifier et il peut ensuite sauvegarder une image de son chef d'œuvre.

Nous voulions créer une application libre, qui donnait beaucoup de choix à l'utilisateur dans ce qu'il pouvait faire, et nous croyons qu'avec les éléments que nous avons inclus dans la première partie de notre projet, nous sommes sur la bonne piste.

Interactivité

Il y a un bon nombre d'interactions possibles dans notre projet étant donné que nous l'avons créé comme une sorte de version de « Paint ». L'utilisateur a presque le contrôle complet de la scène. Il peut ajouter des éléments, éliminer des éléments, changer la nature des éléments qu'il a déjà ajoutés, etc. Bref, l'utilisateur peut faire ce qu'il veut avec les outils que nous lui offrons.

Les entrées qui permettent à l'utilisateur d'envoyer des signaux à l'application sont nombreuses. Il y a les boutons de base qui donne l'option a l'utilisateur de dessiner des formes géométriques, écrire et ajouter du texte dans la scène, ajouter une image dans la scène, prendre et sauvegarder une capture d'écran et ajouter un modelé dans la scène. Il y a aussi les boutons qui apparaissent une fois que l'utilisateur a utilisé sa souris pour sélectionner une composante de la scène. Des commandes pour changer la couleur des lignes, changer la couleur de remplissage, changer la grosseur des lignes, changer la dimension des modèles, changer la position des modèles, changer la rotation des modèles, supprimer un élément, « undo » ou « redo », sont toutes des commandes possibles une fois la composante sélectionnée.

Pour les sorties de l'application, toute forme de dessin peut être considérée comme une sortie, alors la plupart des actions d'entrée qui ont été mentionnées ont une sortie que l'utilisateur voit en conséquence. Il y a le cas de la capture d'écran qui est différente. Cette action ne va pas créer un objet que l'utilisateur peut voir sur l'écran, mais elle va créer un fichier en dehors de l'application où l'utilisateur peut aller voir la capture d'écran qui a été prise.

Technologie

Puisque tous les membres de l'équipe travaillent sous Windows, on a utilisé Visual Studio 2015 afin de développer et compiler notre projet. Cette version est disponible gratuitement (*Visual Studio Community 2015*). Nous utilisons aussi openFrameworks, qui est une boîte à outils dite *open source*. Cette librairie est pour nous un très bon point de départ puisqu'elle est stable et nous fournit une bonne base pour créer une application utilisant OpenGL. Il est relativement facile de trouver réponse à ses questions sur les forums, et plusieurs *addons* sont disponibles avec la librairie, notamment pour charger et afficher des modèles géométriques, ou encore pour faire des interfaces utilisateur.

Afin de gérer le code source, on a utilisé GitHub. C'est une solution gratuite qui permet un partage facile des fichiers entre les membres de l'équipe. Les fichiers sont accessibles par tous en tout temps, soit à partir du logiclel GitHub, soit par le site internet de l'entreprise. Cette technologie nous permet aussi de travailler à plusieurs en même temps sur le projet puisqu'elle intègre un historique des modifications et la possibilité de faire des « branches » afin qu'on puisse travailler sans impacter l'arbre de code principal.

Compilation

Afin de compiler notre projet, il est nécessaire d'avoir Visual Studio 2015 (ou plus récent), et d'avoir le *Platform* toolset *Visual Studio 2015(v140)*. Puisque nous utilisons GitHub et qu'ils ont certaines restrictions quant à la taille maximale de fichiers, il est nécessaire d'ajouter manuellement la librairie openFrameworks dans l'arborescence de notre code source. Nous utilisons la version 0.9.8, disponible sur http://openframeworks.cc dans la section des téléchargements. Une fois téléchargé, il suffit d'extraire le contenu et le mettre dans le dossier */src/libs/*. Si tout a été fait correctement, le chemin suivant devrait être valide :

src/libs/of_v0.9.8_vs_release/libs/openFrameworksCompiled/project/vs

Une fois cette étape terminée, il faut tout simplement ouvrir la solution Visual Studio située dans le répertoire Workspace. À l'intérieur de l'application, il est très important de vérifier que la compilation se fait avec la stratégie **x86**. Compiler le projet intitulé « ProjetEquipe13 ». Dans la séquence, celui-ci va aussi compiler le projet openFrameworks puisque notre projet a une dépendance directe sur celui-ci.

Architecture

Pour nous, la définition de l'architecture logicielle est de concevoir et structurer un programme d'une telle manière que nous pouvons améliorer :

- La qualité
- La conception
- L'évolution
- L'entretien

Nous avons essayé de créer une architecture qui rend ces aspects faciles à gérer en divisant en plusieurs sections. Le rôle de l'application n'est que de mettre le tout en commun et de relayer certains événements. Ainsi, l'application est dans un dossier séparée et inclus les autres dossiers (core, tools, ui). Nous avons une classe de rendu dont la principale tâche est de rendre le contenu de la scène, en plus de conserver les paramètres de dessin active (taille de ligne, couleur de ligne, couleur de remplissage, etc.). En tout temps, l'application ne possède qu'une seule scène. Celle-ci définit le système de coordonnées et s'occupe de la gestion de tous les objets géométriques qui seront créés par l'utilisateur. On y gère aussi le *undo*, *le redo* ainsi que la sélection active. Chaque modification à un élément doit nécessairement passer par les interfaces de la scène, ce qui nous permet de gérer les changements facilement. Les géométries de la scène sont créées par des outils. Un outil est une classe qui, lorsque activée, va recevoir la majorité des événements de la part de l'utilisateur tels que les clics et les déplacements de souris. Il ne peut y avoir qu'un outil d'édition à la fois. Lorsqu'un outil démarre, l'outil courant est remplacé par le nouvel outil. C'est une architecture qui nous permet facilement d'ajouter des fonctionnalités sans pour autant déstabiliser le reste du code. De plus, puisque les outils sont des sous-classes d'une interface, il n'est pas nécessaire à l'application de connaître le fonctionnement interne de l'outil. Cette approche va aussi nous faciliter la gestion de la caméra lorsque nous allons l'implémenter, puisque nous allons introduire la notion d'outil d'affichage qui pourra momentanément suspendre l'outil actif d'édition afin de modifier les paramètres de la vue.

Fonctionnalités

La première section du projet de session visait à nous faire travailler sur les aspects d'image de notre projet. Nous avons choisi implémenter les aspects suivants :

- Il est possible d'importer des fichiers images et de les afficher dans une scène sous une forme ou une autre (1.1) : Le projet offre deux manières d'importer une image : importer à partir d'un bouton d'importation ou « drag and drop » d'une image dans la scène. Si le bouton d'importation est sélectionné, un fichier de dialogue ouvre et donne l'option à l'utilisateur de choisir son fichier image. Dans tous les cas, c'est l'outil *datPlaceImageTool* qui s'occupera de gérer l'ajout à la scène. Quand une image valide est choisie, l'outil permet à l'utilisateur de modifier les paramètres de taille de l'image, et de la bouger dans la scène avant de cliquer pour confirmer l'insertion. Le *drag and drop* supporte l'insertion multiple d'images, qui seront ajoutées une à la fois. Il suffit de faire un clic droit pour en éliminer une et passer à la suivante.

- Il est possible d'exporter des rendus d'une scène dans des fichiers images (1.2): Il y a un bouton pour commencer l'exportation des images. Quand il est activé, un fichier de dialogue ouvre qui donne l'option à l'utilisateur de choisir où il veut sauvegarder son fichier image. L'utilisateur doit donner un nom au fichier pour qu'il puisse le sauvegarder et il va être sauvegardé avec l'extension « .PNG ». Un objet « ofImage » est créé pour garder l'image prise par la capture d'écran (.grabScreen) et la sauvegarder à l'endroit choisi par l'utilisateur.

- Il est possible de sélectionner une couleur parmi un ensemble de couleurs et de l'assigner à un élément visuel (1.4): Puisqu'il est possible de placer des éléments visuels dans la scène de l'application, nous avons donné l'option à l'utilisateur de choisir la couleur de ces éléments. En utilisant « ofxGUI », nous avons créé un panneau qui contrôle le niveau des couleurs RGB. Si, avant de placer un élément visuel dans la scène, le niveau d'une des couleurs est changé, l'élément va

prendre cette couleur. Tant que la couleur n'est pas modifiée, les objets placés vont continuer d'avoir la couleur qui avait été sélectionnée. Le sélecteur de couleur est à même les outils de dessin.

La deuxième partie du projet était fixé sur les dessins vectoriels. Pour cette partie, nous avons implémenté les aspects suivants.

- Il existe au moins 5 représentations visuelles différentes du curseur dessinées à partir de primitives vectorielles (2.1) : nous avons créé 5 différents cas où le curseur peut avoir une représentation visuelle différente. Lorsque l'outil de sélection est actif, le panneau de contrôle permet de clique sur les différents choix offerts. Il est également possible d'utiliser les touches 1 à 5 du clavier. Lorsqu'un curseur différent de celui par défaut est sélectionné, nous cachons la souris avec « ofHideCursor », et dessinons des primitives vectorielles à l'endroit où il devrait être. Cette logique est située dans notre classe de rendu *datRenderer..*

- Il est possible de modifier de manière interactive la valeur des outils de dessin vectoriel tel que l'épaisseur des lignes de contour, la couleur des lignes de contour, la couleur des zones de remplissage et la couleur d'arrière-plan de la scène (2.2) : il est possible de changer les couleurs des lignes qui sont placées dans la scène, comme nous avons mentionné dans 1.4. Il est aussi possible de changer l'épaisseur des lignes avec un panneau en utilisant le même principe de « slider » que les couleurs. De plus, si nous créons une forme vectorielle avec ces lignes, il est possible de changer la couleur de remplissage aussi. Comme avec les lignes, nous avons un panneau « ofxGUI » avec les valeurs RGB et quand les « slider » sont bougé, la couleur de remplissage est modifiée. Pour changer la couleur d'arrière-plan, il faut être dans la fenêtre du bouton « select tool ». Le principe pour changer la couleur est le même qu'avec les deux autres aspects.
-

- Il est possible de créer de manière interactive des instances d'au moins 5 types des primitives vectorielles parmi cet ensemble : point, ligne, carré, rectangle, triangle, quadrilatère, polygone régulier, polygone irrégulier, cercle, ellipse et arc (2.3) : il est possible de créer des lignes dans l'application et avec ces lignes, il est possible de créer a peu près n'importe quelle primitives vectorielles (sauf des cercles, ellipse et arc). Évidemment, il est possible de créer un cercle mais il aurait des vertex qui ne sont pas arrondis. On peut terminer le dessin de 2 manières. Si l'utilisateur décide de cliquer sur le dernier sommet qu'il a inséré, la primitive vectorielle va être insérée dans la scène avec aucun remplissage. Lorsque la géométrie a plus de 2 lignes, il est possible de terminer l'action avec un clic droit, et ainsi obtenir une géométrie fermée ave remplissage.
-
- Un ou des éléments d'interface graphique offrent de la rétroaction informative visuelle à l'utilisateur et des contrôles interactifs pour influencer les états de l'application (2.5) : cet aspect est lié de près avec 2.2, car les contrôles sont évidemment interactifs (il faut glisser les contrôles pour changer la couleur). Il est aussi possible de voir le niveau de chaque élément de couleur RGB(A), l'épaisseur de la ligne, le curseur qui est actif, la couleur de remplissage et la couleur d'arrière-plan. Notre outil de sélection affiche aussi la boîte de sélection avec le déplacement de la souris.

La troisième partie du projet était fixé sur les transformations. Pour cette partie, nous avons implémenté les aspects suivants.

- Tous les éléments visuels présents dans une scène sont organisés dans une ou des structures de données qui permettent l'ajout, la suppression et la sélection d'éléments (3,1) : Lorsque l'outil de sélection est actif, il est possible de cliquer sur un élément géométrique dans le fenêtre ou de dessiner un rectangle de sélection. Notons qu'une zone de sélection créée de gauche à droite

sélectionnera seulement les objets qui sont totalement contenus dans cette zone, alors qu'une zone de sélection dessinée de droite à gauche va sélectionner tous les objets qui sont contenus ou qui touchent la zone dessinée. Ces objets feront désormais partie de la sélection active, et certaines options deviendront disponibles.

- Il est possible de sélectionner plus d'une instance des éléments visuels présents dans une scène et de modifier sur chaque élément de la sélection la valeur de certains attributs qu'ils ont en commun (3,2) : Le principe pour cette fonction est similaire à (3,1). Nous pouvons encadrer plusieurs éléments dans la fenêtre avec le « select tool » et une fois ces éléments sélectionnés, il est possible de modifier leurs attributs S'ils ont des attributs en commun, la modification qui va être faite va s'appliquer sur tous les éléments affectés. S'ils ont des différences, les éléments différents peuvent tout de même être modifiés, mais les changements vont seulement affecter les éléments qui supportent la modification.

- Il est possible de modifier de manière interactive la translation, la rotation et la proportion des éléments visuels présents dans une scène (3,3) : La translation d'un élément peut changer sois les coordonnes de l'axe « X » ou de l'axe « Y ». Un panneau « ofxGUI » est utilisé pour modifier ces variables. Le principe est le même pour la rotation et la proportion. Des panneaux « ofxGUI » ont été créés pour contrôler les variables de rotation et de proportion et quand l'élément du panneau est changé, les variables changent aussi, ce qui change la composante dans la scène.

- Il est possible d'annuler ou de refaire (undo / redo) les dernières actions interactives qui ont un impact sur la transformation des éléments visuels présents dans une scène (3,4) : lorsqu'un élément visual est ajouté dans notre classe de scène, nous assignons à cet objet un identificateur unique. Ainsi, il est

impossible d'ajouter 2 fois la même géométrie, et on peut contrôler l'ajout. Lorsqu'un outil modifie des éléments de la scène, l'outil doit d'avoir en faire une copie, un *clone*. Lorsque l'élément cloné et modifié est renvoyé à la scène, il est trivial de détecter qu'on en avait déjà une copie. On garde ainsi l'historique des éléments qui ont été modifiés, et on se base sur ça pour le *undo* et le *redo*.

La quatrième partie du projet était fixé sur la géométrie. Pour cette partie, nous avons implémenté les aspects suivants.

- Une option permet de dessiner les arêtes d'une boite d'une taille juste assez grande pour envelopper tous les sommets d'un modèle 3D pour chaque type de modèle qu'il est possible d'utiliser avec l'application (4,1) : lorsqu'on ajoute une géométrie à la scène, nous allons chercher le minimum et le maximum sur chaque axe, ce qui nous donne deux points en 3 dimensions qui nous permettent de dessiner une boîte de délimitation. Si l'élément est en 2D, la boîte va seulement avoir des lignes le long des axes de X et d'Y, alors qu'en 3D elle va avoir des lignes le long des axes de X, Y et Z.

- Il est possible de dessiner des instances d'au moins 2 types de modèles 3D importés à partir d'un fichier externe (4,3) : Cet aspect est assez simple, car avec l'intégration de la librairie *ofxAssimpModelLoader*, plusieurs types de fichiers peuvent être lus et affichés en tout simplicité. Nous avons simplement créé une classe *wrapper* qui gère les modèles provenant de cette librairie de la même manière que tout le reste de notre géométrie.

La cinquième partie du projet était orientée vers la manipulation de textures sur les objets. Les fonctionnalités suivantes ont été ajoutées au code.

- L'utilisateur est capable de composer deux textures ensembles pour arriver à un résultat où il peut observer la contribution de chacune des textures sur la scène. Pour ce faire, il n'a qu'à ajouter une photo, la sélectionner avec l'outil de

sélection et il peut ensuite appuyer sur la touche d'ajout de texture. Lorsque cette action est effectuée, une fenêtre d'exploration de fichier s'ouvre et l'utilisateur peut choisir le fichier de texture à ajouter à son image pour la charger en mémoire. Par la suite, le code du *shader* en *glsl* est importé et généré et une espace mémoire est allouée au *fbo*. À ce point-ci, la texture de l'image et la nouvelle texture sont chargées dans le *shader* et un *draw* est appellé sur l'image. Ce qui a pour effet de mettre sur le *fbo* la nouvelle texture qui peut être ensuite conservé en mémoire comme étant une nouvelle image.

Ressources

Nous avons essayé de faire le projet en utilisant du code généré par nous-mêmes. Pour la majorité du projet, nous avons réussi à faire ça, mais il y a évidemment des places où nous avions besoin d'utiliser des ressources externes.

Grâce à openFrameworks, nous avions des outils à notre disposition qui nous ont facilité la tâche pour certains éléments de l'application. Un des outils que nous avons utilisé est ofxGui. Cette classe est utilisée pour créer des interfaces graphiques simples pour l'usager. Nous avons créé des panneaux interactifs qui permettent à l'usager de modifier des attributs de certaines composantes dans la scène. En déplacent les éléments dans l'interface, les variables sont mises à jour et changent l'apparence de la composante dans la scène par l'entremise de rappels de fonctions que nous avons préalablement configurés.

Nous avons aussi utilisé ofxAssimpModelLoader qui permet de charger en mémoire et traiter des modèles 3D de manière pratique. Il faut simplement déclarer une variable de type ofxAssimpModelLoader et de charger le modelé avec les fonctions appropriées. Il est ensuite assez facile de dessiner les aspects désirés avec les autres fonctions.

Le forum d'openFrameworks nous a aussi aidés durant le projet. Quand nous avons des problèmes, ou nous n'étions pas sur de comment approcher un aspect du projet, le forum a été un outil de référence pour des solutions. Par contre, nous avons essayé de ne pas trop nous fier à cet outil, car, comme il est mentionné, nous voulions essayer de créer quelque chose original.

GitHub a aussi été utile pour le projet, car nous pouvions aller voir les exemples du professeur pour nous aider. Avec la base du code que le professeur nous montrait, il était plus facile de créer un aspect du projet que d'avoir à tout commencer à zéro.

Présentation

Alexandre Gagnon

Originaire de Québec et diplômé au baccalauréat en génie informatique depuis 2014, je travaille depuis 3 ans chez Bentley Systems en tant que développeur sur un projet dans le domaine du génie civil. J'ai toujours eu de l'intérêt pour les problèmes mathématiques, la géométrie 3D. Je prends des cours comme celui-ci afin de parfaire mes connaissances sur des aspects de la programmation que je n'ai pas eu la chance d'explorer lors de mon passage initial à l'Université Laval.

Thomas Dixon

Je suis présentement au baccalauréat en Génie Informatique que je vais terminer l'an prochain. Je travaille l'agence de Revenu Québec depuis mai 2017 en tant que programmeur analyste. Bien que le baccalauréat et mon emploi m'ont permis d'acquérir des connaissances sur différents aspects de la programmation, plusieurs avenues restent à explorer. C'est pourquoi, je m'intéresse à ce cours. Je tiens à le suivre dans le but de mieux comprendre et d'être en mesure de créer des outils graphiques qui pourront être utilisé dans l'industrie de la technologie.

Samuel Parent

Je suis originaire du Québec et je vais à l'Université Laval depuis 2 ans. Je suis dans le programme baccalauréat en informatique depuis le début de mes études. Je me considère assez nouveau dans le monde de la programmation étant donné que ça fait environ 2 ans — 2 ans et demi que j'en fais alors j'essaye d'absorber autant d'information possible des gens avec qui je travaille. J'ai choisi ce cours, car je crois qu'il est important de voir tous les aspects de l'informatique avant de prendre une décision sur le domaine dans lequel on veut travailler. Le cours m'avait l'air très intéressant et je ne suis pas déçu d'avoir choisi de le prendre.