Boivin, Philippe (910 101 347)

Carbonneau-Leclerc, Élise (111 126 813)

Desloges, Simon (111 161 956)

Lavoie, Laurie (111 126 242)

Infographie

IFT-3100

Projet de session

Travail présenté à

Monsieur Philippe Voyer

Département d'informatique et de génie logiciel

Université Laval

Hiver 2017

**1. Sommaire**

L'objectif de ce projet de session est de développer une application qui permet de construire, éditer et rendre des scènes 3D. Ce projet permet d'appliquer les notions vues dans le cadre du cours d'infographie.

Ces scènes contiennent plusieurs primitives vectorielles dont des lignes, des cercles et des rectangles. On peut choisir la couleur et la grandeur des celles-ci. On peut également importer une image de fond ou exporter l'image créée. Toutes les couleurs utilisées sont dans l'espace de couleur HSB (Hue, Saturation et Brightness). Une interface permet ces options. Elle a été créée avec ofxgui. Des curseurs dynamique permette de savoir le mode que l'utilisateur utilise: mode normal, mode de création et mode caméra.

Un système d'historique est présent. L'utilisateur peut donc refaire le dernier élément ajouté ou enlever le dernier élément ajouté.

Notre application pourrait servir à créer une scène de jeu vidéo avec des obstacles pour le héro.

**2. Interactivité**

-L'utilisateur peut importer une image de fond à l'aide du bouton "import". Il sera ensuite inviter à choisir un fichier (.png ou .jpg acceptés) à l'aide d'un explorateur de fichier.

-L'utilisateur peut exporter une image de sa scène créée. Pour cela, il utilise le bouton "export".

-L'utilisateur peut modifier la valeur de "hue" de la couleur voulue pour créer ses formes. Il utilise alors le curseur nommé "hue".

-L'utilisateur peut modifier la valeur de "saturation" de la couleur voulue pour créer ses formes. Il utilise alors le curseur nommé "saturation".

-L'utilisateur peut modifier la valeur de "brightness" de la couleur voulue pour créer ses formes. Il utilise alors le curseur nommé "brightness".

-L'utilisateur peut créer une ligne. Il clique alors sur le bouton "line" et peut ensuite choisir l'endroit où il veut le point de départ de sa ligne en cliquant avec la souris. Il peut ensuite déplacer la souris pour déterminer la position de fin de la ligne. Il relâche la souris pour indiquer la position qu'il souhaite.

-L'utilisateur peut créer un rectangle. Il clique alors sur le bouton "rectangle" et peut ensuite choisir l'endroit où il veut le point de départ de son rectangle en cliquant avec la souris. Il peut ensuite déplacer la souris pour déterminer la position de fin du rectangle. Il relâche la souris pour indiquer la position qu'il souhaite.

-L'utilisateur peut créer une ellipse. Il clique alors sur le bouton "circle" et peut ensuite choisir l'endroit où il veut le point de départ de son ellipse en cliquant avec la souris. Il peut ensuite déplacer la souris pour déterminer la position de fin de l'ellipse. Il relâche la souris pour indiquer la position qu'il souhaite.

-L'utilisateur peut recréer l'élément déjà («redo») ajouté à l'aide de la touche "r". L'élément sera créé un peu décalé du précédent pour qu'on le voit.

-L'utilisateur peut annuler une action («undo») à l'aide de la touche «u». Le dernier élément sera enlevé de l'application.

**3. Technologie**

Nous avons utilisé plusieurs outils technologiques pour réaliser notre projet:

-Visual Studio 2015: L'environnement de développement utilisé pour le développement de l'application.

-OpenFrameworks: Outil C++ pour l'utilisation d'OpenGL

-ofxGui: Librairie permettant de créer le menu de notre application.

**4. Architecture**

**5. Fonctionnalités**

Section 1: Image

-Importation: L'utilisateur peut importer une image de fond à l'aide du bouton "import". Il sera ensuite inviter à choisir un fichier (.png ou .jpg acceptés) à l'aide d'un explorateur de fichiers. Le code utilise essentiellement la fonction «load» avec le path choisi dans l'explorateur de fichiers.

-Espace de couleur: L'utilisateur peut modifier la valeur de "hue", "saturation" et "brightness" de la couleur voulue pour créer ses formes. Il utilise alors les curseurs nommé "hue", "saturation" et "brightness". Le code utilise essentiellement cette ligne: c.setHsb(shape[index].fillColor[0], shape[index].fillColor[1], shape[index].fillColor[2]);

-Exportation: L'utilisateur peut exporter une image de sa scène créée. Pour cela, il utilise le bouton "export". Le code utilise la fonction imageExport qui contient les éléments suivants: ofImage imageTemp;

string timestamp = ofGetTimestampString("-%y%m%d-%H%M%S-%i");

string fileName = name + timestamp + "." + extension;

imageTemp.grabScreen(0, 0, ofGetWindowWidth(), ofGetWindowHeight());

imageTemp.save(fileName);

Section 2: Dessin vectoriel

-Curseur dynamique: 3 curseurs différents existe. Un curseur création lors de la création de primitive. Un curseur transformation lors de la transformation interactive. Un curseur caméra lors de la modification des éléments de caméra. Le code se trouve essentiellement dans la fonction drawCursor.

-Primitives vectorielles: L'application permet de créer des ellipses, des rectangles et des lignes à l'aide du premier menu. Le code utilise essentiellement les fonctions ofDrawEllipse, ofDrawRectangle et ofDrawLine.

-Interface: L'interface se trouve dans le coin en haut à gauche. Il permet de choisir les différents mode de création et leur valeur ou option. Voici le code de la première section de l'interface: gui.add(hue.setup("hue", 0, 0, 255));

gui.add(saturation.setup("saturation", 0, 0, 255));

gui.add(brightness.setup("brightness", 0, 0, 255));

gui.add(circleButton.setup("circle"));

circleButton.addListener(this, &ofApp::circleListener);

gui.add(rectangleButton.setup("rectangle"));

rectangleButton.addListener(this, &ofApp::rectangleListener);

gui.add(lineButton.setup("line"));

lineButton.addListener(this, &ofApp::lineListener);

gui.add(importButton.setup("import"));

importButton.addListener(this, &ofApp::importListener);

gui.add(exportButton.setup("export"));

exportButton.addListener(this, &ofApp::exportListener);

Section 3: Transformation

-Transformation interactive: L'utilisateur peut effectuer une translation, une rotation ou une transformation de proportion.

L’utilisateur peut cliquer sur le bouton translation, le curseur se modifie et il peut déplacer une primitive créée.

void Renderer::translateShape(float xPressed, float yPressed, float xReleased, float yReleased)

…

shape[index].position1[0] = shape[index].position1[0] + translateX;

shape[index].position1[1] = shape[index].position1[1] + translateY;

shape[index].position2[0] = shape[index].position2[0] + translateX;

shape[index].position2[1] = shape[index].position2[1] + translateY;

L’utilisateur peut cliquer sur le bouton proportion, le curseur se modifie et il peut modifier la dimension d’une primitive créée en sélectionnant une position.

void Renderer::proportionShape(float xPressed, float yPressed, float xReleased, float yReleased)

shape[index].position2[0] = xReleased;

shape[index].position2[1] = yReleased;

L’utilisateur peut cliquer sur le bouton rotation, le curseur se modifie et lorsqu’il clique sur une primitive, la primitive fait une rotation de 45 degré.

void Renderer::rotateShape(float xPressed, float yPressed, float xReleased, float yReleased)

if (anglesShapes[index] == 360) {

anglesShapes[index] = 0;

}

anglesShapes[index] += 45;

}

ofPushMatrix();

ofSetRectMode(OF\_RECTMODE\_CORNER);

ofTranslate(shape[index].position1[0], 0, 0);

ofTranslate(0, shape[index].position1[1], 0);

ofRotateZ(anglesShapes[index]);

drawEllipse(

0,

0,

shape[index].position2[0] - shape[index].position1[0],

shape[index].position2[1] - shape[index].position1[1]);

ofPopMatrix();

-Structure de scène

-Historique: L'utilisateur peut recréer l'élément déjà («redo») ajouté à l'aide de la touche "r". L'élément sera créé un peu décalé du précédent pour qu'on le voit. L'utilisateur peut annuler une action («undo») à l'aide de la touche «u». Le dernier élément sera enlevé de l'application. Voici le code correspondant:

void Renderer::addToShape(float x1, float y1, float x2, float y2, unsigned char fillColorH, unsigned char fillColorS, unsigned char fillColorB, VectorPrimitive type)

{

if (head >= 1)

{

shape[head].position1[0] = x1;

shape[head].position1[1] = y1;

shape[head].position2[0] = x2;

shape[head].position2[1] = y2;

shape[head].fillColor[0] = fillColorH;

shape[head].fillColor[1] = fillColorS;

shape[head].fillColor[2] = fillColorB;

shape[head].type = type;

shape[head].strokeWidth = strokeWidthDefault;

head = ++head >= count ? 0 : head;

}

}

void Renderer::removeFromShape()

{

if (head >= 1)

{

shape[head-1].type = VectorPrimitive::NONE;

head = head - 1;

}

}

Section 4: Géométrie

-Primitives

-Modèle

-Texture

Section 5: Caméra

-Propriétés de caméra

-Mode de projection

-Caméra interactive

**6. Ressources**

Le Github du cours a été utilisé comme inspiration pour certaines fonctions: <https://github.com/philvoyer/IFT3100H17>

La documentation d'OpenFrameworks a servi d'aide: <http://openframeworks.cc/learning/>

Un tutoriel sur ofxGui a également servi d'aide: <https://sites.google.com/site/ofauckland/examples/6-addons---ofxsimpleguitoo>

**7. Présentation**

**Philippe Boivin**:

**Élise Carbonneau-Leclerc**: Élise termine sont BAC en informatique cet été. Auparavant, elle a fait un DEC au Cégep de Sainte-Foy en informatique. Elle aime le cinéma et les jeux vidéo.

**Simon Desloges**: Étudie en BAC en informatique de l'Université Laval. Intéressé entre autre par les jeux vidéo.

**Laurie Lavoie**: Laurie travaille temps plein au CHU de Québec. Elle fait son BAC en informatique à temps partiel. Elle a également étudié au Cégep de Sainte-Foy en informatique.