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 importer un fichier de modèle 3d à l'aide du bouton "import". Il sera ensuite inviter à choisir un fichier (.obj) 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.

-L'utilisateur peut cliquer sur procedural geometry pour faire apparaitre et disparaitre un mesh 3d procédural. Il est peut-être nécessaire de déplacer la camera avec Camera Start et dragger la souris pour le voir.

-L'utilisateur peut cliquer sur particules, pour faire apparaitre et disparaitre un mesh de points avec une texture de feu.

-L'utilisateur peut créé une courbe de Bézier cubique ou une courbe de Hermite à l'aide du bouton du menu de courbes. Il clique ensuite sur le plan pour choisir les 4 points.

-L'utilisateur peut créé une spline de Bézier à l'aide du bouton du menu de courbes. Il clique ensuite sur le plan pour choisir les 5 points.

-L'utilisateur peut créé un diagramme de Voronoï à l'aide du bouton du menu de courbes. Il peur recliquer pour le faire disparaître.

**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.

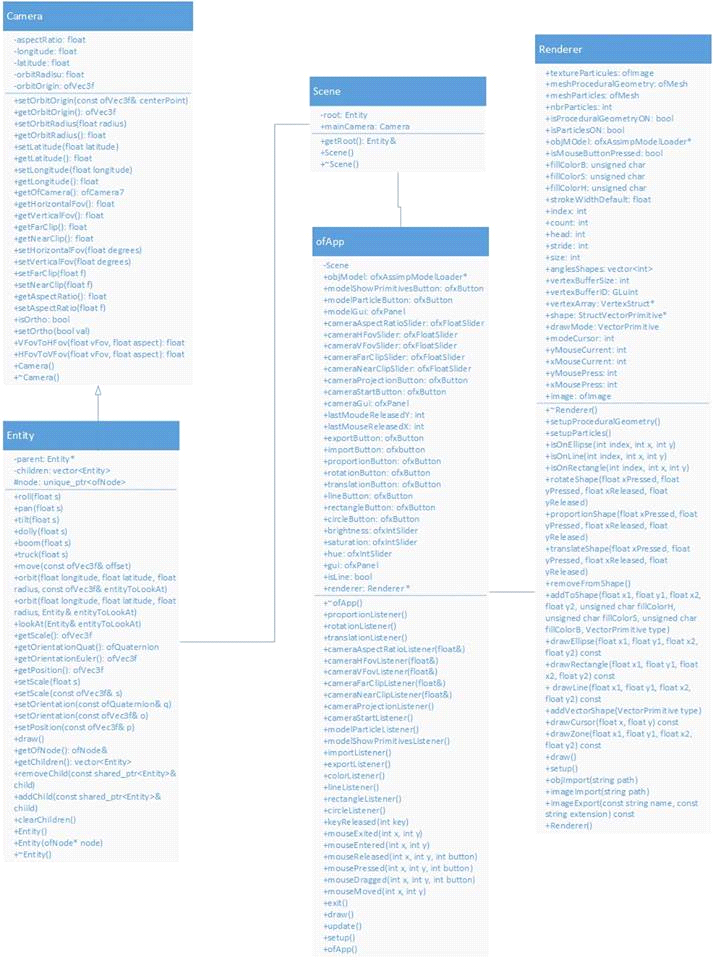
-OpenFrameworks: Outil C++ pour l'utilisation d'OpenGL

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

-ofxAssimpModelLoader: Addon de OpenFrameworks qui permet de charger des modèles 3D en mémoire à partir de fichiers de diverses formats.

-git et GitHub: Système de contrôle de source et hébergement de dépôts git.

**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: Les modèles 3D dans la scène sont représenté par une hièrarchie d'entités en graphe orienté acyclique. Une entité peut avoir aucunes ou plusieurs entités enfants, mais ne peut avoir qu'un seul parent. La position et l'orientation d'une entité est affecté par celle de son parent (elles sont relative à celle du parent). Une structure de donnée Scène est utilisé afin de regrouper les éléments essentiels d'une scéne: une entité racine à laquelle toutes les autres entités seront des enfants ou des sous enfant (etc) ainsi qu'une référence à la caméra active, qui est elle-même une entité enfant de la racine. Cela permet donc d'attacher la caméra à une entité. Voici la définition de la classe entité qui sert de classe mère aux autre type d'entités:

class Entity

{

public:

Entity();

Entity(ofNode\* node);

virtual ~Entity();

void clearParent();

void setParent(Entity& parent);

void clearChildren();

void addChild(const std::shared\_ptr<Entity>& child);

void removeChild(const std::shared\_ptr<Entity>& child);

std::vector<std::shared\_ptr<Entity>>& getChildren();

// Please don't call ofNode::clearParent or ofNode::setParent, thank you.

virtual ofNode& getOfNode();

virtual void draw();

virtual void setPosition(const ofVec3f& p);

virtual void setOrientation(const ofVec3f& o);

virtual void setOrientation(const ofQuaternion& q);

virtual void setScale(const ofVec3f& s);

virtual void setScale(float s);

virtual ofVec3f getPosition() const;

virtual ofVec3f getOrientationEuler() const;

virtual ofQuaternion getOrientationQuat() const;

virtual ofVec3f getScale() const;

void lookAt(Entity& entityToLookAt);

void orbit(float longitude, float latitude, float radius, Entity& entityToLookAt);

void orbit(float longitude, float latitude, float radius, const ofVec3f& entityToLookAt);

void move(const ofVec3f& offset);

void truck(float s);

void boom(float s);

void dolly(float s);

void tilt(float s);

void pan(float s);

void roll(float s);

protected:

std::unique\_ptr<ofNode> node;

private:

Entity\* parent;

std::vector<std::shared\_ptr<Entity>> children;

};

-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

-Géometrie procédurale

La fonctionalité à été implémenté par un mesh de primitives de points qui prend en référence l'image de fond. Elle recrée l'image de fond mais avec une heightmap qui correspond à la luminosité de l'image. Elle est générée et détruite lorsque l'on clique sur procedural geometry. Il est peut-être nécessaire de déplacer la camera avec Camera Start et dragger la souris pour le voir.

void Renderer::setupProceduralGeometry() {

meshProceduralGeometry.clear();

meshProceduralGeometry.setMode(OF\_PRIMITIVE\_POINTS);

isProceduralGeometryON = !isProceduralGeometryON;

if (isProceduralGeometryON) {

int w = image.getWidth();

int h = image.getHeight();

for (int x = 0; x<w; ++x) {

for (int y = 0; y<h; ++y) {

ofColor c = image.getColor(x, y);

float intensity = c.getLightness();

ofVec3f pos(x, y, intensity \* 10);

meshProceduralGeometry.addVertex(pos);

meshProceduralGeometry.addColor(ofColor(c));

}

}

}

}

-Modèle

Le bouton import permet de charger un fichier obj qui ensuite s'affiche au millieu de la caméra.

void ofApp::importListener() {

ofFileDialogResult result = ofSystemLoadDialog("Load file");

if (result.bSuccess) {

string path = result.getPath();

if (path.find(".jpg") != std::string::npos || path.find(".png") != std::string::npos)

{

renderer->imageImport(path);

}

else if (path.find(".obj") != std::string::npos) {

objModel = new ofxAssimpModelLoader();

objModel->loadModel(path);

}

}

}

et dans le draw

if (objModel != nullptr) {

objModel->draw(OF\_MESH\_FILL);

}

-Particules

Lorsque l'utilisateur clique sur Particules, un mesh de points d'une quantité finie est généré. Ces points sont à des positions aléatoires, de couleur blanches et chargent une texture fournie dans l'application (fire.jpg)

void Renderer::setupParticles() {

meshParticles.clear();

meshParticles.setMode(OF\_PRIMITIVE\_POINTS);

for (int i = 0; i < nbrParticles; i++) {

ofVec2f p = ofVec2f(ofRandom(0, ofGetWidth()), ofRandom(0, ofGetHeight()));

meshParticles.addVertex(p);

meshParticles.addColor(ofColor(255,255,255));

}

ofDisableArbTex();

textureParticles.loadImage("fire.jpg");

glPointSize(28);

isParticlesON = !isParticlesON;

isProceduralGeometryON = !isProceduralGeometryON;

}

Dans le draw

if (renderer->isParticlesON) {

ofEnableAlphaBlending();

ofEnablePointSprites();

renderer->textureParticles.getTextureReference().bind();

renderer->meshParticles.drawFaces();

renderer->textureParticles.getTextureReference().unbind();

}

Section 5: Caméra

-Propriétés de caméra: Les attributs de la caméra active peuvent être modifiés à l'aide de l'interface graphique offert par l'application. Ces attributs inclus les angles de champs de vision (horizontal et vertical), le ratio d'aspect et la distance du plan de clipping avant et arrière. Voici par exemple le code qui permet de faire la conversion pour passer d'un FOV horizontal vers un FOV vertical, en radians:

float HFovToVFov(float hFov, float aspect) {

return 2.0f \* atan(tan(hFov / 2.0f) / aspect);

}

-Mode de projection: Deux modes de projections sont supportés, c'est à dire projection en perspective et projection orthogonale. Voici le code qui permet cela:

void Camera::setOrtho(bool val)

{

if (val)

{

this->getOfCamera().enableOrtho();

}

else

{

this->getOfCamera().disableOrtho();

}

}

-Caméra interactive: Il est possible de contrôler la caméra active à l'aide de la souris. En clquant sur le bouton Camera Start de l'interface graphique, le mode de contrôle de la caméra sera activé et il sera alors possible d'enfoncer le bouton gauche de la souris afin de déplacer la caméra et d'enfoncer le bouton droit de la souris afin de se raprocher ou s'éloigner du sujet. Voici le code:

void ofApp::mouseDragged(int x, int y, int button)

{

auto delta\_x = x - renderer->xMouseCurrent;

auto delta\_y = y - renderer->yMouseCurrent;

renderer->xMouseCurrent = x;

renderer->yMouseCurrent = y;

if (renderer->modeCursor == 5)

{

if (button == OF\_MOUSE\_BUTTON\_LEFT)

{

auto longitude = scene->mainCamera.getLongitude() + delta\_x;

auto latitude = scene->mainCamera.getLatitude() + delta\_y;

scene->mainCamera.setLongitude(longitude);

scene->mainCamera.setLatitude(latitude);

}

else if(button == OF\_MOUSE\_BUTTON\_RIGHT)

{

auto oldRadius = scene->mainCamera.getOrbitRadius();

auto radius = oldRadius + (delta\_y \* log(oldRadius) \* 0.75f);

radius = radius <= 100.0f ? 100.0f : radius;

scene->mainCamera.setOrbitRadius(radius);

}

}

ofLog() << "<app::mouse drag at: (" << x << ", " << y << ") button:" << button << ">";

}

Section 9

-Courbe cubique: L'utilisateur peut créé une courbe de Bézier cubique ou une courbe de Hermite à l'aide du bouton du menu de courbes. Il clique ensuite sur le plan pour choisir les 4 points.

inline void bezierCubic(

float t,

float p1x, float p1y, float p1z,

float p2x, float p2y, float p2z,

float p3x, float p3y, float p3z,

float p4x, float p4y, float p4z,

float& x, float& y, float& z)

{

float u = 1 - t;

float uu = u \* u;

float uuu = uu \* u;

float tt = t \* t;

float ttt = tt \* t;

x = uuu \* p1x + 3 \* uu \* t \* p2x + 3 \* u \* tt \* p3x + ttt \* p4x;

y = uuu \* p1y + 3 \* uu \* t \* p2y + 3 \* u \* tt \* p3y + ttt \* p4y;

z = uuu \* p1z + 3 \* uu \* t \* p2z + 3 \* u \* tt \* p3z + ttt \* p4z;

}

enum class Curve { BEZIER\_CUBIC, HERMITE, SPLINE, NONE };

inline void hermite(

float t,

float p1x, float p1y, float p1z,

float p2x, float p2y, float p2z,

float p3x, float p3y, float p3z,

float p4x, float p4y, float p4z,

float& x, float& y, float& z)

{

float u = 1 - t;

float uu = u \* u;

float uuu = uu \* u;

float tt = t \* t;

float ttt = tt \* t;

x = (2 \* ttt - 3 \* tt + 1) \* p1x + (ttt - 2 \* tt + t) \* p2x + (ttt - tt) \* p3x + (-2 \* ttt + 3 \* tt) \* p4x;

y = (2 \* ttt - 3 \* tt + 1) \* p1y + (ttt - 2 \* tt + t) \* p2y + (ttt - tt) \* p3y + (-2 \* ttt + 3 \* tt) \* p4y;

z = (2 \* ttt - 3 \* tt + 1) \* p1z + (ttt - 2 \* tt + t) \* p2z + (ttt - tt) \* p3z + (-2 \* ttt + 3 \* tt) \* p4z;

}

-Courbe paramétrique: L'utilisateur peut créé une spline de Bézier à l'aide du bouton du menu de courbes. Il clique ensuite sur le plan pour choisir les 5 points.

inline void spline(

float t,

float p1x, float p1y, float p1z,

float p2x, float p2y, float p2z,

float p3x, float p3y, float p3z,

float p4x, float p4y, float p4z,

float p5x, float p5y, float p5z,

float& x, float& y, float& z)

{

float u = 1 - t;

float uu = u \* u;

float uuu = uu \* u;

float uuuu = uuu \* u;

float tt = t \* t;

float ttt = tt \* t;

float tttt = ttt \* t;

x = uuuu \* p1x + 4 \* uuu \* t \* p2x + 4 \* uu \* tt \* p3x + 4 \* u \* ttt \* p4x + tttt \* p5x;

y = uuuu \* p1y + 4 \* uuu \* t \* p2y + 4 \* uu \* tt \* p3y + 4 \* u \* ttt \* p4y + tttt \* p5y;

z = uuuu \* p1z + 4 \* uuu \* t \* p2z + 4 \* uu \* tt \* p3z + 4 \* u \* ttt \* p4z + tttt \* p5z;

}

-Triangulation: L'utilisateur peut créé un diagramme de Voronoï à l'aide du bouton du menu de courbes. Il peur recliquer pour le faire disparaître.

ofxVoronoi2D voronoi;

vector<ofVec2f> pts;

pts.push\_back(ofVec2f(float(15), float(155)));

pts.push\_back(ofVec2f(float(78), float(261)));

pts.push\_back(ofVec2f(float(456), float(82)));

pts.push\_back(ofVec2f(float(756), float(432)));

pts.push\_back(ofVec2f(float(715), float(264)));

voronoi.compute(pts, ofRectangle(0, 0, ofGetWidth(), ofGetHeight()));

for (ofxSegmentIterator it = voronoi.edges(); it; ++it) {

ofxSegment s = \*it;

ofLine(s.p1, s.p2);

}

ofMesh mesh;

voronoi.buildMesh(mesh);

mesh.setColorForIndices(0, mesh.getNumIndices(), ofFloatColor(1.0f, 1.0f, 1.0f));

for (unsigned int i = 0; i < mesh.getNumIndices(); i += 3) {

mesh.setColor(mesh.getIndex(i), ofFloatColor(0.0f, 0.0f, 1.0f)); // cell centers

}

mesh.draw();

**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--- HYPERLINK "https://sites.google.com/site/ofauckland/examples/6-addons---ofxsimpleguitoo" HYPERLINK "https://sites.google.com/site/ofauckland/examples/6-addons---ofxsimpleguitoo" HYPERLINK "https://sites.google.com/site/ofauckland/examples/6-addons---ofxsimpleguitoo"ofxsimpleguitoo](https://sites.google.com/site/ofauckland/examples/6-addons---ofxsimpleguitoo)

Documentation openframeworks sur la generation procédurale à partir d'une image

<http://openframeworks.cc/ofBook/chapters/generativemesh.html#generativemeshusinganimagetodrivethecreationofamesh>

Tutoriel sur les point sprites

<http://www.kamend.com/2011/11/point-sprites-in-openframeworks/>

Librairie Voronoi2D

https://github.com/xionluhnis/ofxVoronoi2D

Ressources utilisées

fichier .obj téléchargé sur internet pour fin de tests (airboat.obj)

<http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html>





**7. Présentation**

**Philippe Boivin**: Philippe étudie à l'Université Laval en informatique. Dernière année en cours. Grand fan de jeux vidéos.

**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.