



CENTRE DE DÉVELOPPEMENT
ET DE RECHERCHE
EN IMAGERIE NUMÉRIQUE

RAPPORT TECHNIQUE

E*D Films

Projet

Pont entre Photoshop et Maya

1718_26_EDF

v. 2018-11-15

TABLE DES MATIÈRES

Table Des Matières	2
Introduction	6
Présentation Du Contexte.....	6
Objectifs.....	7
Structure Du Projet Et Distribution	8
Introduction.....	8
Plan du projet	8
<i>Général</i>	8
<i>Photoshop</i>	9
<i>Maya</i>	9
Distribution	10
CMAKE Et Projet C++.....	12
Introduction.....	12
Gestion des dépendances	12
Présentation de CMake.....	13
<i>Projet classique</i>	13
<i>Projet avec CMAKE</i>	14
Gestion facile des plateformes et versions.....	14
Avantage Génération de projet indépendant.....	16
<i>Structure des répertoires</i>	17
La logique de CMAKE	19
Les possibilités.....	20
Curve Algorithme	21
Introduction.....	21
1- Les Intersections	22
<i>Définitions</i>	22
<i>Processus</i>	22
2. Les Voisins.....	23

3. Traitement Supplémentaire	24
<i>Problématique 1</i>	24
<i>Problématique 2</i>	24
<i>Problématique 3</i>	26
4. Les Faces	26
Algorithme De Génération Linéaire	28
Introduction	28
Fonctionnement de l'algorithme	28
<i>Paramètre de précision du mesh</i>	28
<i>Génération de la grille</i>	29
<i>Filtration des contours</i>	29
<i>Identification du contenu</i>	30
<i>Construction du tableau de polygone</i>	31
Les courbes	31
Bounding box	32
<i>Calcul de la bounding box classique</i>	32
<i>Calcul de la bounding box orientée</i>	32
Cas particuliers	33
<i>Les formes creuses</i>	33
<i>Les formes creuses imbriquées</i>	34
<i>Multiple forme par layer</i>	35
Influence Algorithme	36
Introduction	36
Construction des faces	37
Est-ce que l'on doit diviser la face?	37
Point Central	38
Diviser la face	39
Diviser les arêtes	40
Combiner les étapes	41
UI	42

Introduction.....	42
L'interface	42
La sérialisation	45
Les sections.....	46
Les comparaisons	46
Les calques.....	46
Parser PSD	47
Introduction.....	47
Document spécification présentation	47
" <i>File Header</i> "	48
" <i>Image Resources</i> "	48
" <i>Layer & mask Information</i> ".....	48
<i>Image Data</i>	49
Dépendance avec "Util"	49
Références	50
Export Png	51
Introduction.....	51
Notre première tentative	51
Information sur le PNG.....	51
Choix de la librairie.....	52
<i>Génération des textures</i>	52
<i>Conversion des Données</i>	52
Maya Components (Éditeur Et Données).....	54
Introduction.....	54
Détails des options de scripting	54
Les différents objets de l'api	55
<i>Les types d'objets C ++</i>	55
<i>Les classes pratique</i>	55
Gestion des systèmes généraux.....	56
Création d'un mesh et de ses composantes.....	56

<i>UV</i>	57
<i>Modifier et Opération</i>	58
Le "plungin command"	59
L'intégration du UI.....	60
Plugin Photoshop Et Adobe CSXS	61
Introduction.....	61
Environnement de développement	61
Adobe CSXS	62
Les opérations du Plugin.....	63
Analyse Complémentaire - Polygon Subdivision	65
Introduction.....	65
1- Le centre.....	65
<i>Avantage</i>	65
<i>Désavantage</i>	66
<i>Solution aux désavantages</i>	66
<i>Décision final</i>	66
2- La division	67
<i>Avantage</i>	67
<i>Désavantage</i>	67
<i>Solution à ce désavantage</i>	67
<i>Décision final</i>	68
Analyse Complémentaire - Minimal Cycle Basis	69
Introduction.....	69
Les premières idées	69
A*	71
Minimal Cycle Basis	73

INTRODUCTION

Cette section présente le point de départ concernant le partenariat avec E*D films pour la réalisation du passeport Innovation.

PRÉSENTATION DU CONTEXTE

E * D Films est une maison de production de films d'animation et un studio proposant des services de production complets et de post-production.

Que ce soit en 2D, 3D, VR ou en stop-motion, ils développent des propriétés intellectuelles, des concepts et des personnages ainsi que des outils et tutoriels pour les industries de l'animation et du jeu vidéo.

Le processus de modélisation d'un modèle 3D dans Maya à partir d'un fichier PSD d'entrée est essentiellement manuel. Le processus est long et nécessite l'exportation de chaque couche. Certains outils permettent une génération de maillage à partir d'une texture, mais le résultat est un maillage triangle. L'adaptation de ce résultat prend plus de temps que la création manuelle du maillage.

Par conséquent, dans l'industrie de l'animation, où le maillage créé doit être un maillage à base de quad pour une meilleure animation et un meilleur éclairage, il est nécessaire de disposer d'un maillage à base de quad efficace issu de l'outil de génération de fichiers PSD.

La première étape du projet consistait en un prototype générant un maillage à base de quad.

La seconde étape consistait à créer le lien entre la génération de maillage et tous les composants associés au maillage dans maya.

À ce stade, nous avons du améliorer la génération et donner une meilleure interface entre Photoshop et Maya pour gagner du temps en production.

OBJECTIFS

- Une solution doit être trouvée pour créer la connexion entre Photoshop et Maya.
- Les tracés sont créés à partir des différentes couches lors de l'étape d'exportation.
- Une Nomenclature vérifiée avant l'exportation vers Maya, avec commentaires pertinents.
- La génération de maillage crée un maillage à base de quad.
- Une interface conviviale donne le contrôle sur la génération et alimente correctement toutes les modifications dans le PSD.
- Un plugin dans Photoshop aide l'utilisateur à exporter les tracés.
- Des courbes supplémentaires basées sur des masques de vecteurs influencent les maillages générés.

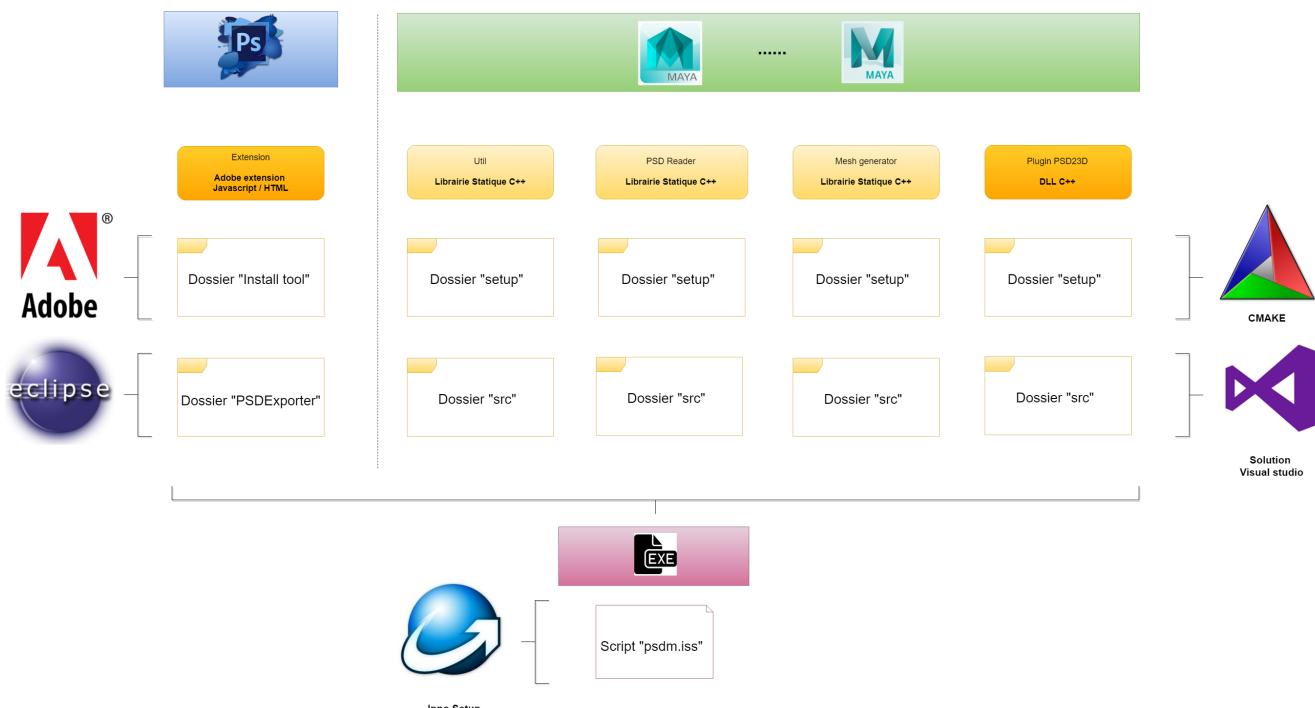
STRUCTURE DU PROJET ET DISTRIBUTION

Introduction

Le projet met en relation 4 technologies différentes, nos principaux acteurs sont Photoshop et Maya. Les deux autres technologies sont CMAKE pour le maintien de la structure du projet et Inno setup pour la distribution des modules aux utilisateurs. Le document suivant présente l'architecture de l'environnement de développement.

Plan du projet

Général



Le projet produit 3 paquets en finalité:

- Une extension Photoshop. (en bleu dans le graphique)
- Un plugin "mll" pour Maya. (en vert dans le graphique)
- Un exécutable qui intègre les deux précédents paquets. (en violet dans le graphique)

Photoshop

Pour **photoshop** nous utilisons Eclipse qui propose un plugin pour la génération d'un template pour les extensions d'adobe.

Maya

Pour Maya le projet est divisé en 4 projets:

- **Util**
 - Contient les classes partagées entre tout les modules et les structures de données.
- **Psd_reader**
 - Permet de lire le contenu d'un PSD et de stocker ces valeurs.
- **Mesh_generator**
 - Contient l'implémentation de plusieurs algorithmes permettant de générer des vertex et des polygones en fonction de différents types de données en entrée.
 - Dispose des algorithmes de manipulation des courbes de Bezier.
- **Plugin_PSD23D**
 - Contient l'implémentation de l'ensemble des fonctionnalités de gestion d'un plugin C++ dans maya.
 - Les modules de génération des différents composants éditeur pour la création et la manipulation des objets de type mesh ou texture dans Maya.
 - L'implémentation de l'interface avec Qt pour l'utilisation du plugin.
 - L'algorithme de conversion des textures au format PNG.
 - L'algorithme de sérialisation des metadatas associées aux paramètres de l'interface.

Chaque projet dispose de script destiné à l'utilisation de **CMAKE** pour la génération de solution indépendante des différents sous projets de maya.

Le dossier **setup** permet de générer des solutions "visual studio" pour chaque projet. Ces projets permettent la génération de librairie indépendante pouvant être réutilisée.

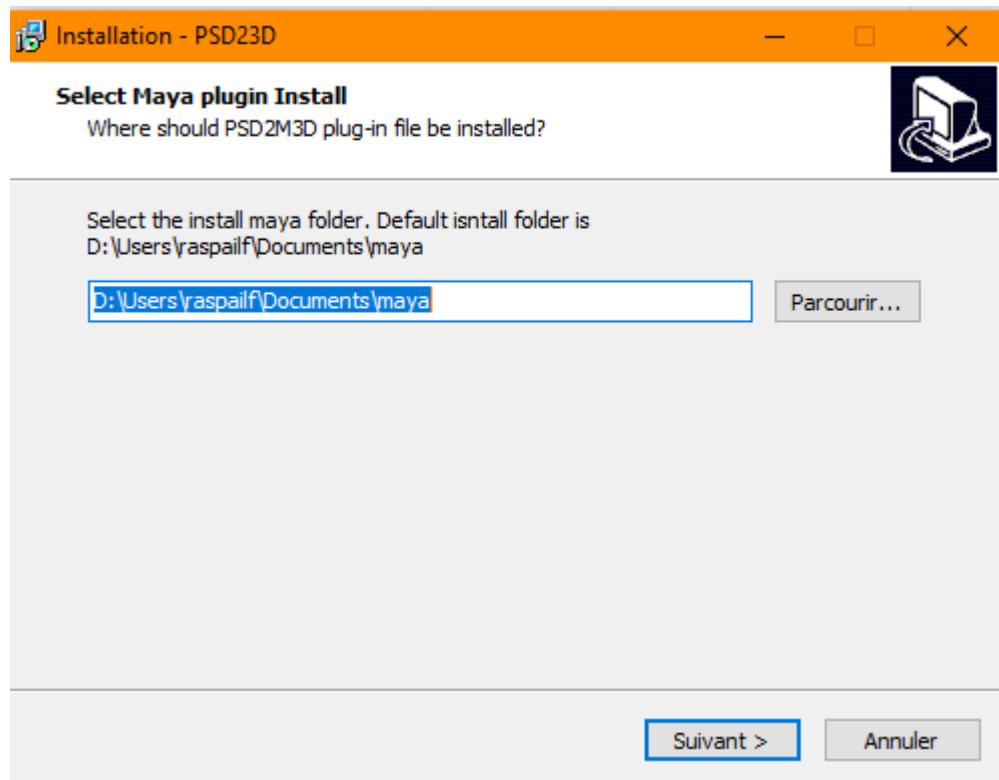
Distribution

Pour la distribution des différents packages du projet, nous avons utilisé **Inno setup** permettant de regrouper l'extension Photoshop ainsi que les différentes versions du plugin Maya (Plugin Maya 2018 et Plugin Maya 2016).

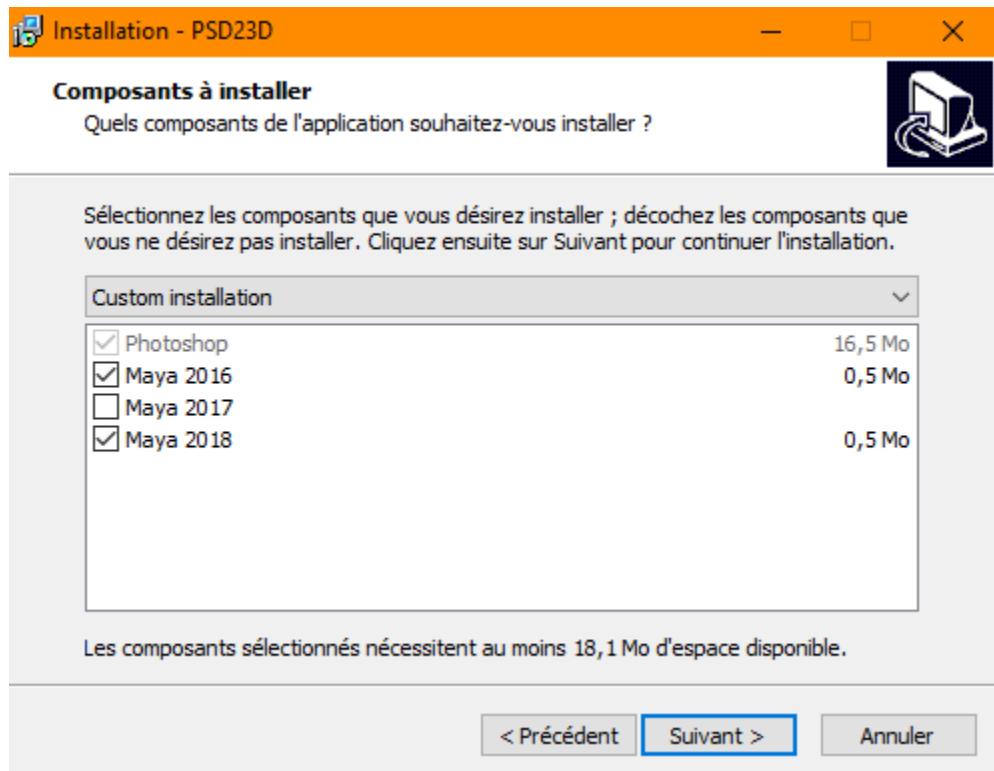
L'installateur propose des options de dépôt pour le plugin et s'occupe d'exécuter un script permettant l'installation et le référencement auprès de Photoshop via le gestionnaire d'extension installé avec la suite créative d'Adobe.

La génération de l'exécutable est réalisé avec un script offrant la capacité de:

- Paramétriser des écrans personnalisés.



- Paramétriser le contenu à intégrer lors d'une installation.
- Paramétriser le contenu à installer.



- Exécuter des opérations par ligne de commande.

La documentation explique assez bien les fonctionnalités et l'API de scripting;

- <http://www.jrsoftware.org/isinfo.php>
- Une série d'attribut permet de spécifier le contenu et leurs utilisation **[Setup]**, **[Languages]**, **[Types]**, **[Components]**, **[Files]**, **[run]**, **[Code]**.
 - **[Setup]** → Variable global de l'installateur, ex: *version*.
 - **[Languages]** → Les différentes langues proposées à l'utilisateur.
 - **[Types]** → La définition des différents types d'installations ex: "custom", "Full".
 - **[Components]** → Un tag permettant d'être utilisé pour l'association de fichier ou d'opération.
 - **[Files]** → les fichiers à intégrer à l'installateur et donc à déployer lors de l'installation.
 - **[run]** → Les lignes de commande à exécuter pour des opérations spécifiques.
 - **[Code]** → Zone permettant de spécifier les écrans, le flot de navigation et les interventions des différentes opérations précédemment référencées.

CMAKE ET PROJET C++

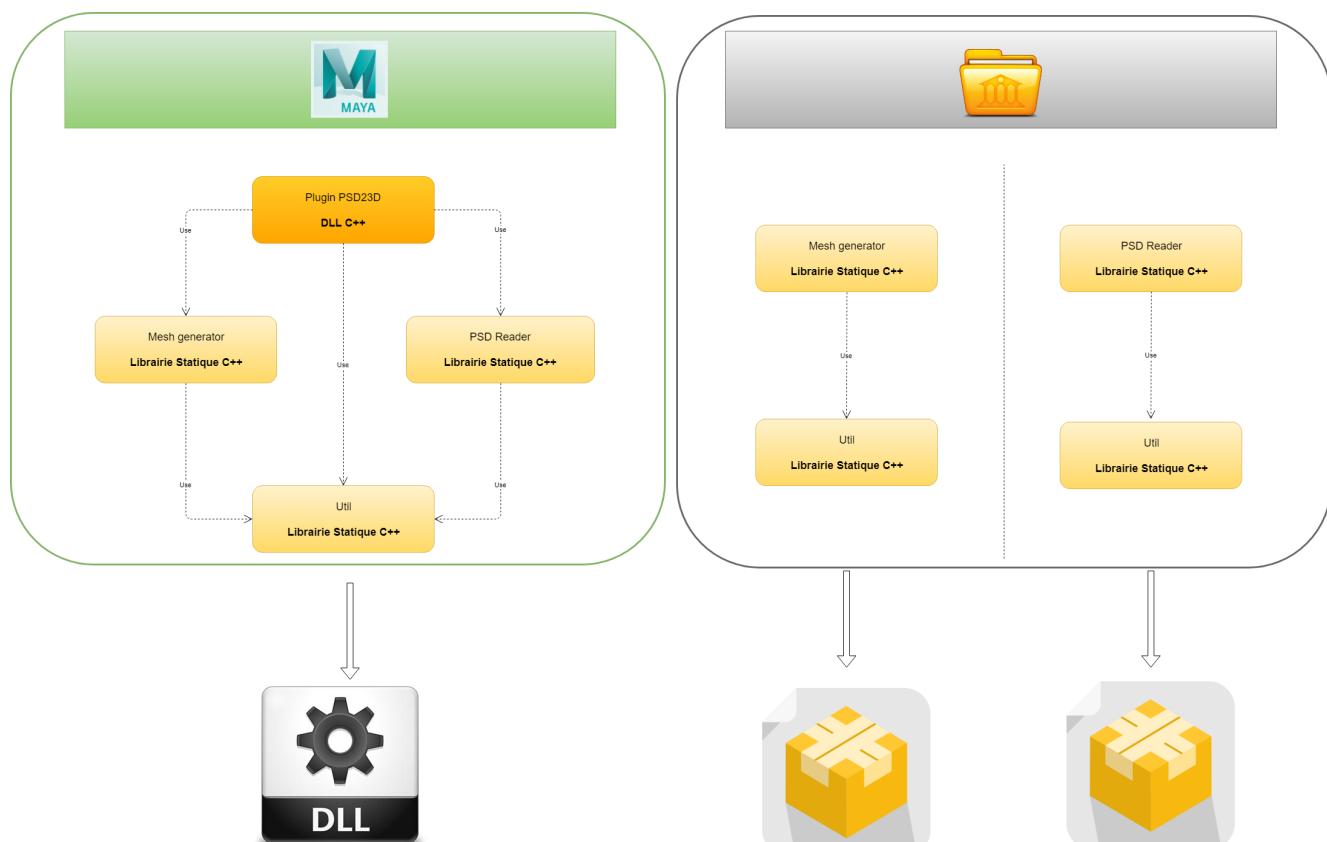
Introduction

Le plugin de maya a été réfléchi pour que les composants du projet puissent servir de manière individuel. Le fait de séparer la partie qui gère les composants de Maya des parties "lecteur de psd" et "génération de mesh", permet à ces modules d'être utilisés dans d'autre projet. Exemple: Nous serions en mesure d'écrire des convertisseurs de Photoshop vers **Unity, Unreal, blender,**

Avoir ce type de structure demande beaucoup de paramétrage pour un environnement de développement. Nous avons donc utilisé un outil permettant de scripter le paramétrage. CMAKE est l'outil sélectionné pour ce projet. Il offre l'avantage de gagner du temps lors de changement dans la structure à grandeur de l'équipe de travail et en vue du transfert technologique.

Gestion des dépendances

Plan de construction des pacquages et de leur **dépendance**.



La fragmentation de nos projets de développement permet de créer des librairies statiques utilisable dans d'autre projet.

Le plugin maya ne fait qu'utiliser ces librairies.

- Le "**PSDparser**" convertit et stocke des informations en C++.
- Le "**Mesh generator**" prend des courbes et des points en entrée et retourne une structure de vertex et de polygone exploitable par n'importe quel outils.

Le projet "**Util**" regroupe:

- Des opérations mathématiques.
- Des formats de données utilisés en lecture du fichier *PSD* et lors des générations.
- Ce projet est similaire à la notion de "Core" avec des classes minimalistes utiles dans les différents modules.

Comme nous pouvons le voir dans le graphique ci-dessus

- "**Util**" est intégré dans toutes les solutions.
- "**Mesh generator** et **PSDparser**" sont des modules indépendants et autonomes.
- "**PSD23D**" est le projet maître avec l'utilisation des librairies et l'intégration des résultats dans maya. Il a une dépendance vers tous les modules précédents.

Présentation de CMake

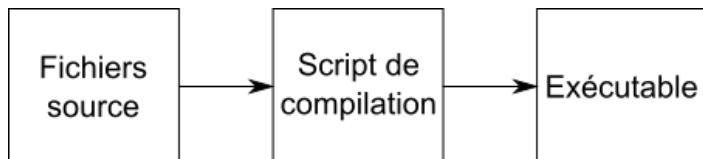
Cmake est un outil pratique de compilation, il permet de:

- Scripter la **composition** d'un projet. (script .cpp, .c, .h, organisé en dossiers)
- Scripter la compilation avec la gestion des différentes dépendances.
- Centraliser des variables de paramétrage comme de compilation.
- Facilement créer et paramétrier un environnement de travail pour Visual Studio ou d'autre environnement de développement.
- Gérer une solution avec de multiple projet interdépendant.

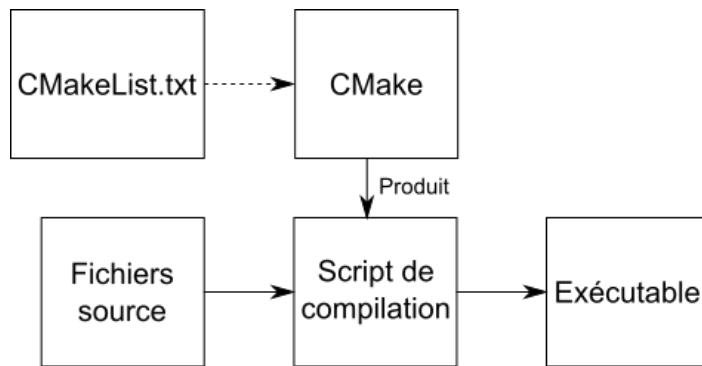
Lien vers l'outil:

- <https://cmake.org>

Projet Classique



Projet Avec CMAKE



Gestion facile des plateformes et versions

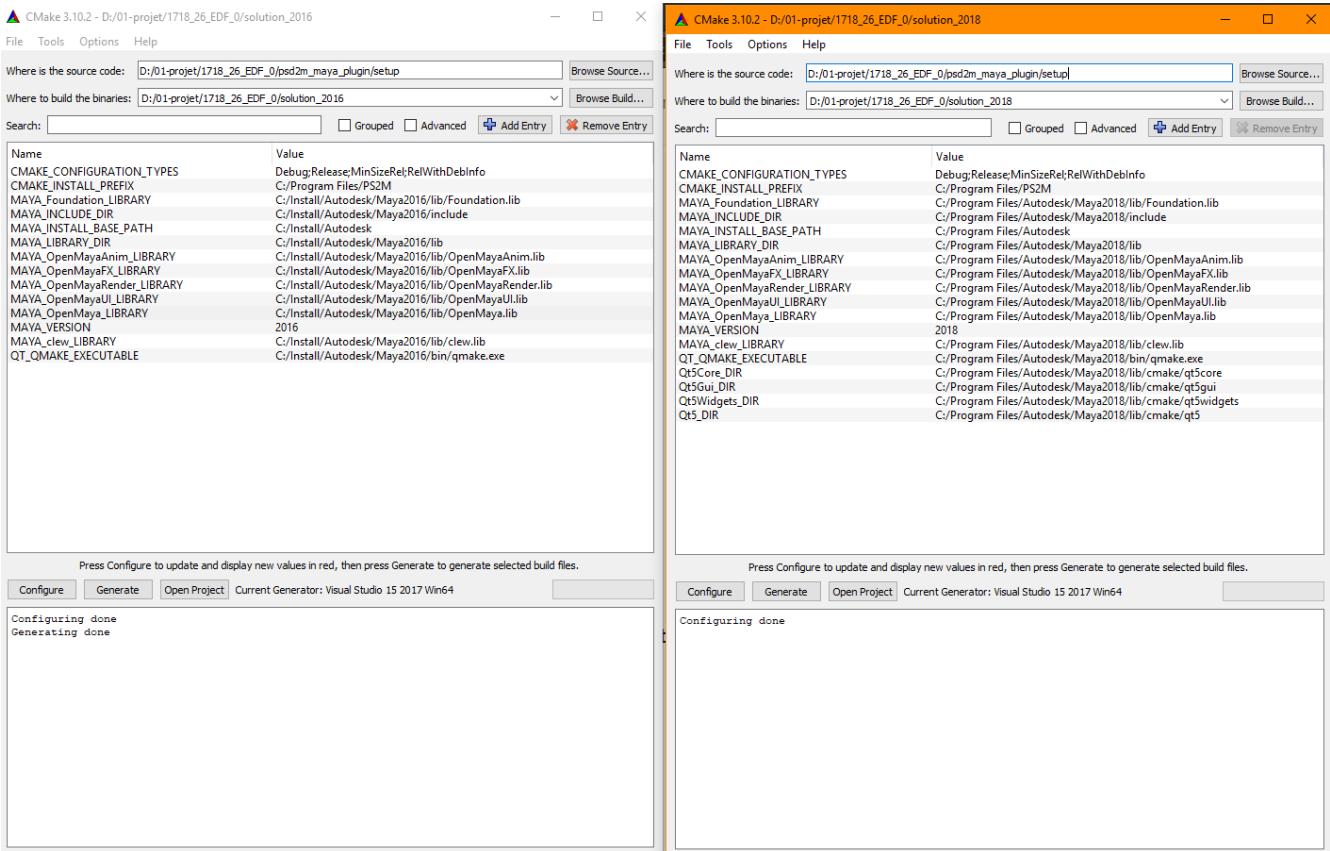
Dans le cadre du projet nous devions livrer le plugin pour Maya 2016 et Maya 2018. C'est deux versions disposent de deux "devkit" différents, les **APIs** étant différentes, les dépendances le sont aussi. Une dépendance étant l'utilisation de librairies exposées par Maya ou d'autres logiciels tiers.

De plus Maya se base sur **Qt** pour faire son menu, entre la version 2016 et 2018 Maya est passé de la version 4.8.6 à la version 5.6.1 de Qt. Ces versions ne suivent pas les versions standard de Qt, elles sont légèrement modifiées pour les besoins de Maya. Dans le projet aucun écart entre la documentation officielle de Qt et les composants utilisés n'a été noté.

En résumé nos dépendances sont:

- l'API de maya pour l'utilisation des composants Maya.
- L'API de Qt embarquée dans Maya, pour les menus.
- Nos dépendances inter-module précédemment présentées.

L'illustration suivante montre la génération d'une solution **visual studio** à gauche pour "**Maya 2016**" et à droit pour "**Maya 2018**".



On observe que:

- Le projet source est **le même** dans les deux cas.
- Le dossier de destination est différent pour avoir les **deux** solutions séparément.
- il y a besoin de moins de paramètre pour Maya 2016 que maya 2018.
 - CMake utilise les scripts **CMakeLists.txt**. pour rechercher les dépendances vers les librairies utilisées.
 - Les variables exposées sont les valeurs qu'il a automatiquement trouvées pour les dépendances visées par les scripts **CMakeLists.txt**.

Ces variables sont utiles comme dans notre cas en spécifiant le champ "**MAYA_VERSION**" les dépendances vers l'API de maya et l'API de Qt seront différentes.

Si des variables ne sont pas identifiées correctement, il vous ait donné la possibilité de spécifier la valeur. Utile lors d'installation non conventionnel d'un librairie impliquée. Dans l'exemple ci dessus vous remarquerez que ma version de Maya 2016 est installée dans un autre dossier que ceux par défaut de Windows. Le script ne trouvera pas de dossier Maya au chemin classique d'une installation, il me faudra renseigner le chemin vers l'installation manuellement puis re-cliquer sur "**Configurer**".

Conclusion:

- Un projet unique.
- Des variables de l'environnement de développement associées au code source via les scripts de CMAKE. (plus besoin de variable d'environnement dans Windows, diminution des risques d'erreurs)

- Automatisation de la génération de la solution ou de la compilation en fonction des variables contenues dans les fichiers CMakeLists.txt.

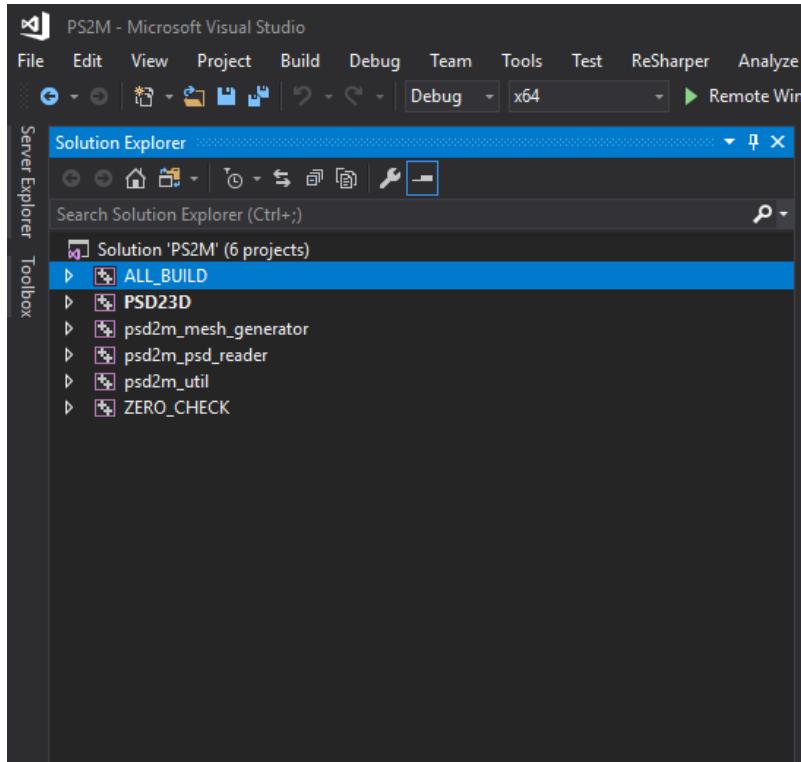
Avantage Génération de projet indépendant

Un autre point important dans notre projet c'est la présence d'un répertoire "**Setup**" pour chaque projet. Ce dossier contient le CMakeLists.txt de référence. permettant de configurer le projet et ses dépendances de manière autonome.

Dans notre cas le **setup** de :

- **Util** ne créera une solution qu'avec le code source de Util.
- **Mesh generator** et **Psdreader** auront respectivement le code de leur projet ainsi que celui de util étant en dépendance direct.
- **Psd23d** disposera de tous les projets, étant dépendant des trois précédents.

L'image suivant illustre la solution générée avec le dossier "**setup**" de **Psd23d**.

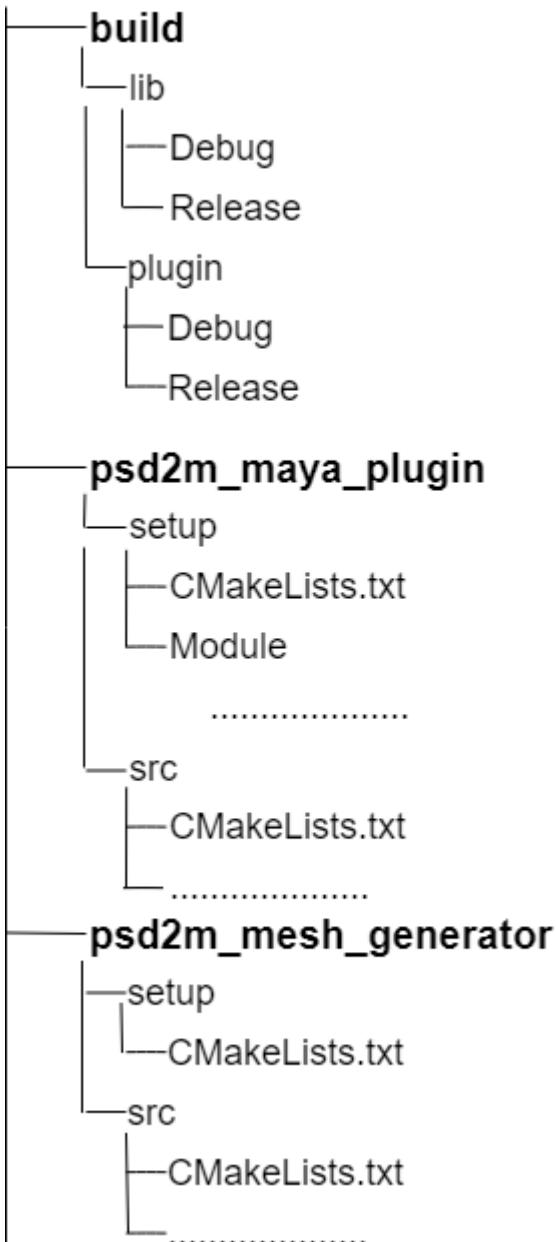


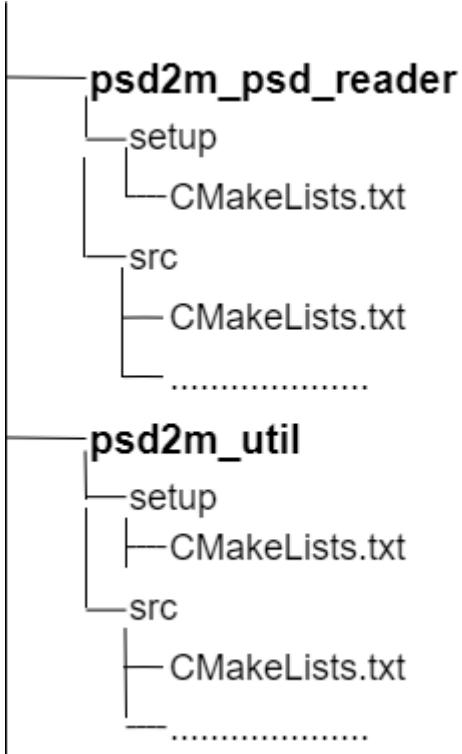
Cette configuration nous permet de facilement débugger un projet et ses dépendances.

Zero_check est un projet généré par CMAKE, il utilise les scripts CMakeLists.txt. Il re-compile l'environnement sans passer par l'interface ou les lignes de commande de CMAKE. Très pratique pour pousser une modification au autre programmeur qui n'auront qu'a re-compiler **zero_check** pour voir les modifications de la solution s'appliquer. (variable de compilation / structure du projet / ajout suppression de script / ...)

Structure Des Répertoires

Voici l'organisation des répertoires et des projets.





Les dossiers permettent de générer des solutions juste du projet et ses dépendances.

Les dossier **sources** (src) contiennent le code source de chaque projet

La structure des fichiers dans le répertoire est différent de celle des solutions, se sont les CMakeLists.txt qui la définissent.

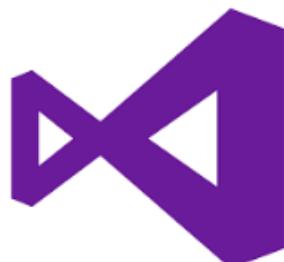
Le dossier **build** est un dossier spécifié dans les CMakeLists.txt.

- Cette valeur sera injecté automatiquement dans les solutions générées.
- Avoir le dossier build en dehors des projets permet de les exclure automatiquement du système de versionnement.

La logique de CMAKE



CMAKE

Solution
Visual studio

- Modification de l'environnement de compilation
- Ajout de script au projet
- Ajout d'une librairie

- Modification du code des scripts du projet.
- Compilation
- Debugging
- Profiling

Environnement de développement

Produit développé

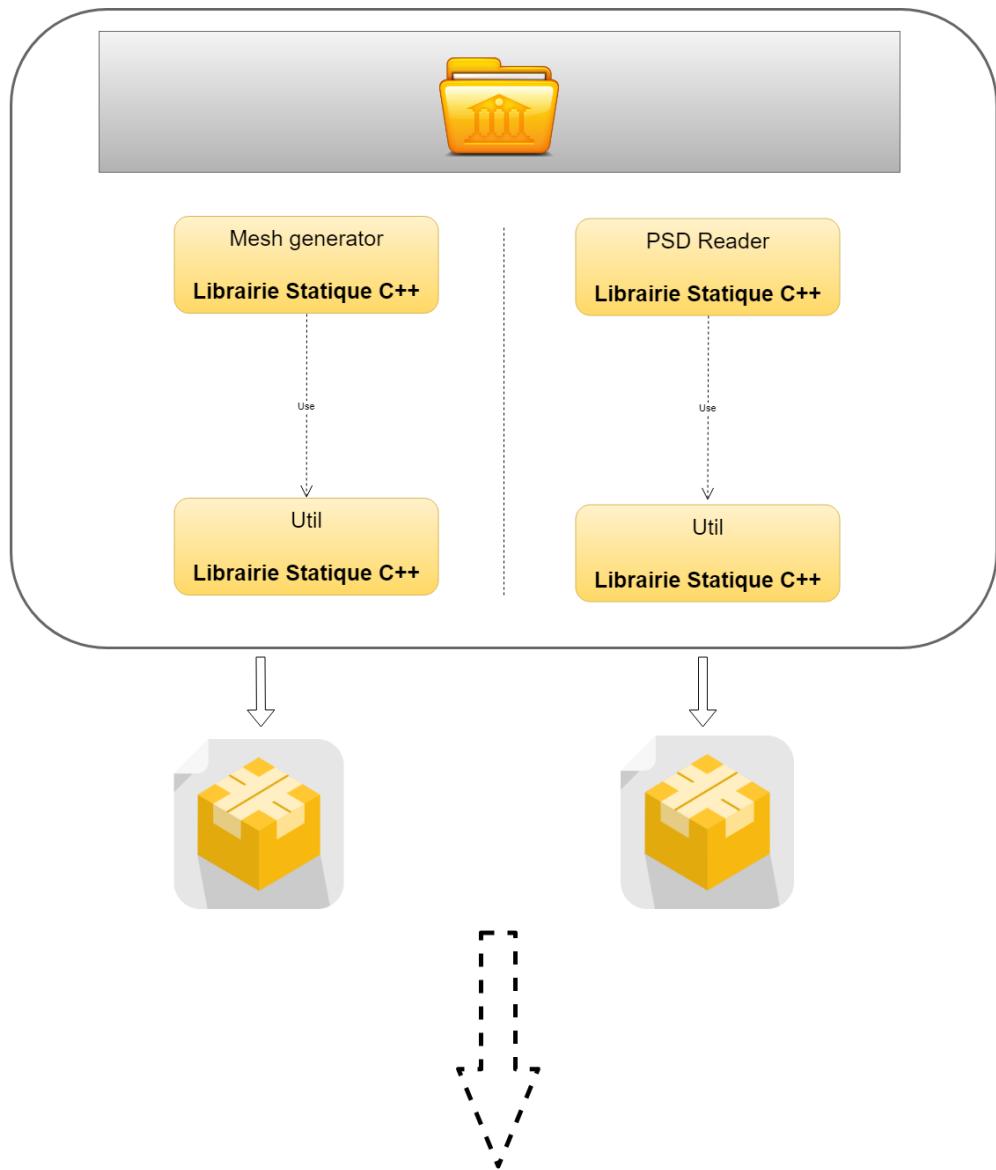
CMAKE s'occupe de configurer un projet, il faut donc que toutes les informations impliquant un changement dans la structure d'un solution soient renseignées dans un fichier de script CMAKE pour les voir s'appliquer dans la solution ou à la compilation:

- Ajout d'une nouvelle librairie (gestion des dépendances)
- Ajouter un script
- Les options de compilation.

Alors notre outil de développement ne servira plus qu'à:

- Modifier le code des fichiers.
- Compiler
- Débugger / Profiler.

Les possibilités



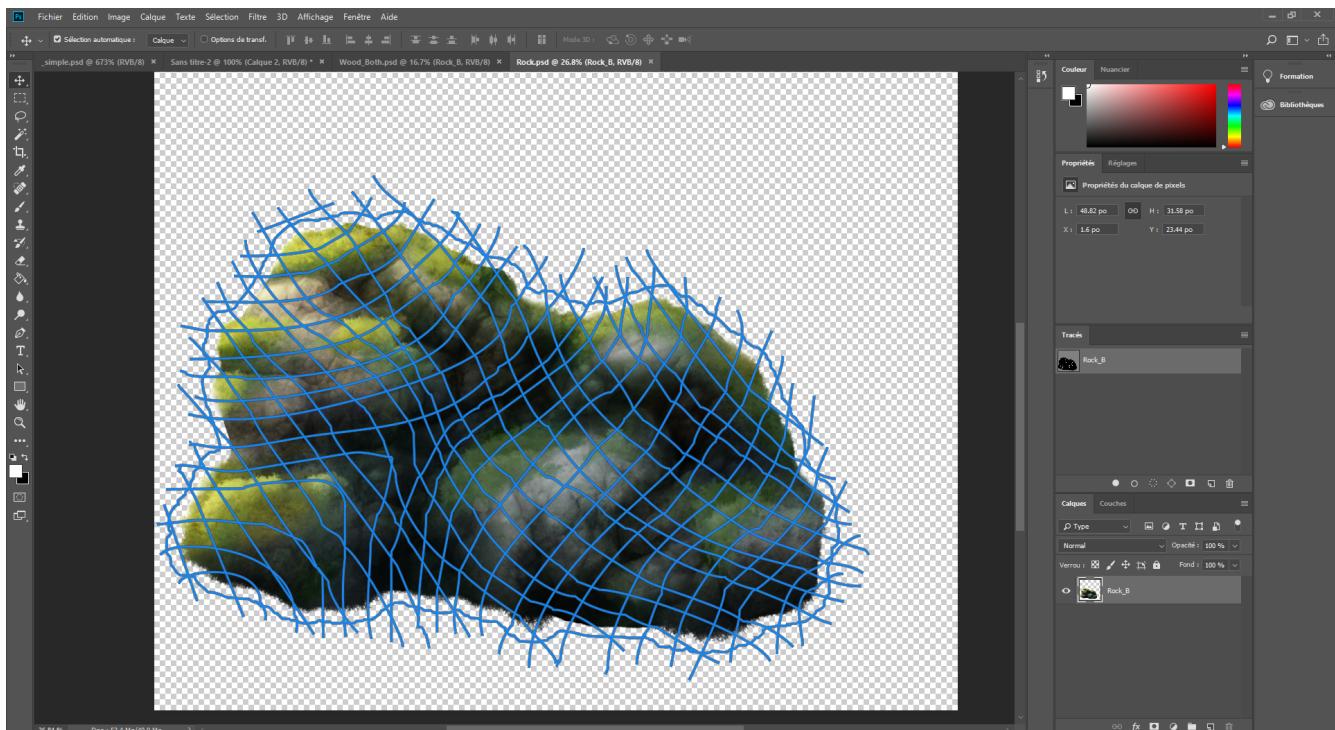
Le concept de lire les données du PSD restent inchangé d'une plateforme à l'autre ni lors de la génération de la donnée d'un mesh. Il suffit d'écrire l'équivalent de **PSD23D** pour chaque environnement présenté ci-dessus.

CURVE ALGORITHME

Introduction

Le "Curve Algorithm" permet à un utilisateur de construire un maillage de "Tracés" dans Photoshop pour construire sois-même un mesh et ses faces.

Voici un exemple de ce type de maillage.



Pour réussir l'algorithme va récupérer toutes les intersections, puis va réaliser la construction du mesh qui se divisera en 4 grandes parties.

1. Trouver toutes les intersections entre les Curves de Bézier.
2. Trouver tout les voisins de chaque intersection.
3. Faire un traitement Pre-Algo pour nettoyer les intersections.
4. Trouver toutes les faces.

Après ces 4 grandes étapes, on aura assez d'information pour construire le Mesh.

Ces étapes ainsi que tout le code relié à l'algorithme ont pour point d'entrée la fonction [GenerateMesh](#) dans le script [curveMeshGenerator.cpp](#). Le suivi de cette documentation sera plus facile s'il est suivi en parallèle avec le code.

1- Les Intersections

Définitions

Avant de commencer, nous devons donner la définition des classes (et donc les expressions) utilisées dans l'algorithme.

- **Bezier** : Quatre **Vector2F** qui définissent une [courbe b茅zier cubique](#).
- **Curve** : La combinaison de un ou plusieurs **Bezier** se suivant parfaitement l'un apr猫s l'autre (le dernier point du Bezier 001 sera le premier du Bezier 002, etc.)

Au tout d茅but de la fonction [GenerateMesh](#) on commence par cr茅er une structure de donn茅es "`std::vector<CurveData> paths`".

- La classe **CurveData** contient une **Curve** et un *vector* de **PathIntersection**.
- Tandis que **PathIntersection** contient une valeur qui repr茅sente le pourcentage de cette intersection versus la **Curve** et la **Node** r茅sultant de l'intersection. (ex. Si le troisi猫me B茅zier de la curve a une intersection 脿 56% de celui-ci, la valeur sera de 3.56).
 - Les **PathIntersection** seront trouv茅s tr猫s bient猫t.
- Ensuite, **Node**, qui sera utilis茅 dans toutes les prochaines 茅tapes est un 茅l茅ment contenant la position de l'intersection et l'index de la node dans la collection qui le contient ainsi que tous ses voisins.
 - Les voisins seront trouv茅 dans la prochaine 茅tape.

Processus

Une fois qu'on a tous nos *paths*, on va it茅rer 脂ravers tous les *paths* pour trouver les intersections. Nous n'avons pas besoin de comparer un path deux fois (ie. A -> B et B -> A vont donner le m猫me r茅sultat) alors la deuxi猫me boucle d'it茅ration commence toujours 脿 la deuxi猫me. Une **CurveData** peut 茅tre compar茅 avec lui-m猫me par contre.

Regardons par la suite la fonction *FindIntersections*, celui-ci va continuer le processus pr茅c茅dent en comparant tous les **Bezier** de chaque **Curve**. Notre but sera de trouver tous les Bezier qui se croisent et de cr茅er une **Node** et deux **PathIntersection** pour chacune d'entre elles. Les Nodes sont rajout茅es 脿 la structure de donn茅es "`std::vector<Node*> & nodes`". Les **PathIntersections** seront rajout茅s 脿 chaque **CurveData**.

Maintenant nous allons 茅tudier l脿g猫rement l'algorithme pour trouver tous les intersections entre deux B茅zier. Le code a 茅t茅 pris de

- https://github.com/erich666/GraphicsGems/tree/master/gemsiv/curve_isect

Ce dernier s'est bas茅 sur <https://www.particleincell.com/2013/cubic-line-intersection/>. Ce qui ressemble tr猫s sommairement 脿 ce qui suit :

```

class Bezier:
    p0, p1, p2, p3

func BezierIntersection(Bezier A, Bezier B):
    // Depth est trouvé basé sur Wang's Theorem (https://vdocuments.site/documents/flatness-criteria-for-subdivision-of-rational-bezier-curves-and-surfaces.html)
    depthA = A.depth
    depthB = B.depth

    if !BoundsOverlap(A, B):
        return

    RecursiveIntersection(A, B)

func RecursiveIntersection(Bezier A, int depthA, Bezier B, int depthB):
    if depthA == 0 && depthB == 0:          // On peut maintenant comparer les bezier
    comme s'ils étaient des segments droits
        return SegmentIntersection(A.p0, A.p3, B.p0, B.p3)

    bezierASplits = depthA <= 0 ? { A } : A.split()
    bezierBSplits = depthB <= 0 ? { B } : B.split()

    --depthA
    --depthB

    forall subA in bezierASplits:
        forall subB in bezierBSplits:
            if BoundsOverlap(subA, subB):
                return RecursiveIntersection(subA, depthA, subB, depthB)

```

2. Les Voisins

Pour toutes les prochaines étapes il va falloir connaître les voisins de chaque **Node** / Intersections. Cette étape va se faire rapidement grâce à tous nos efforts dans l'étape précédent.

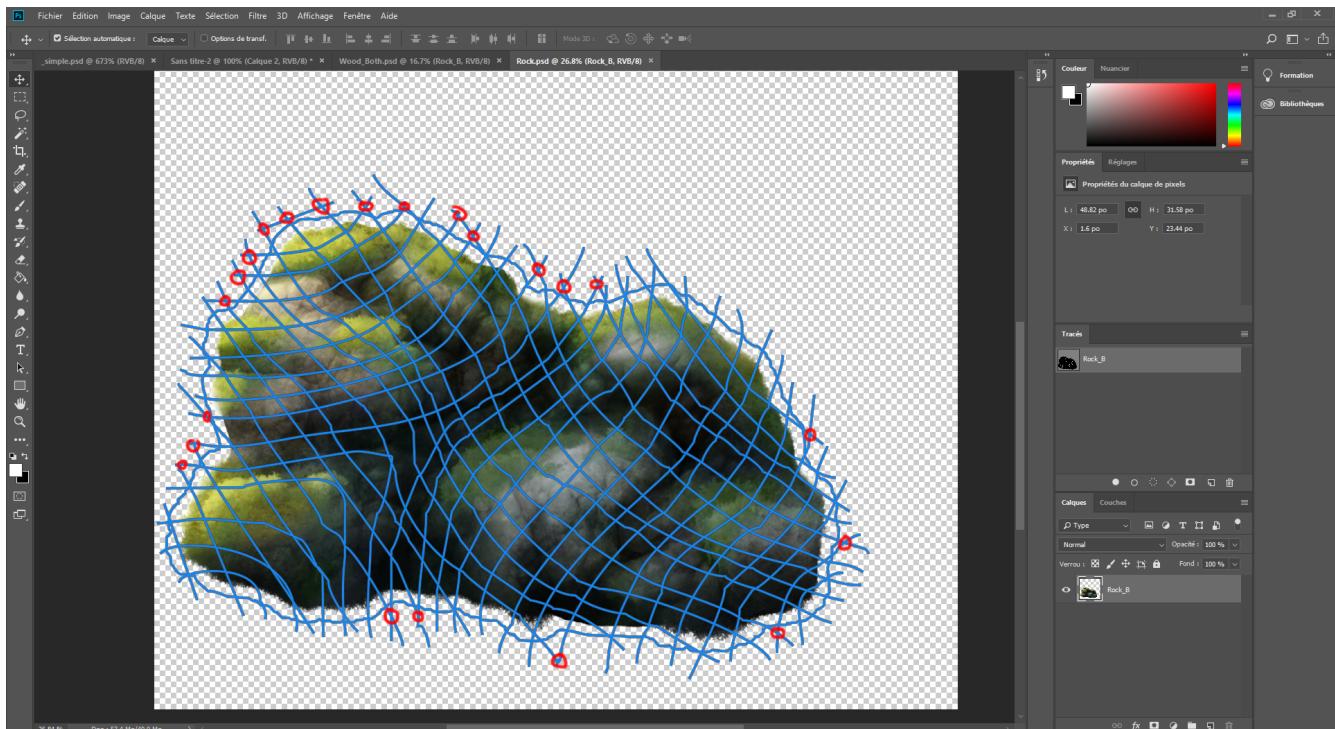
Chaque liste de **PathIntersection** dans **CurveData** sera ordonné du plus petit au plus grand basé sur la valeur du pourcentage. Ensuite, on connectera en ordre les **Nodes** contenues dans les **PathIntersection** (ex. A.Node.AddNeighbour(B.Node) & B.Node.AddNeighbours(A.Node)). Cette itération va automatiquement créer tous les connections entre toutes les **Nodes**.

3. Traitement Supplémentaire

Nous allons maintenant nous occuper de trois problèmes supplémentaire.

Problématique 1

Pour faire un premier nettoyage du Mesh, on enlève les intersections qui se trouvent à l'extérieur de la Curve originale (celui qu'on peut voir faire le tour de la roche). Vu qu'on ne sait pas nécessairement quelle Curves est l'original, on s'occupe d'enlever toutes les intersections qui se retrouvent à l'extérieur de toutes les Curves qui font une boucle.

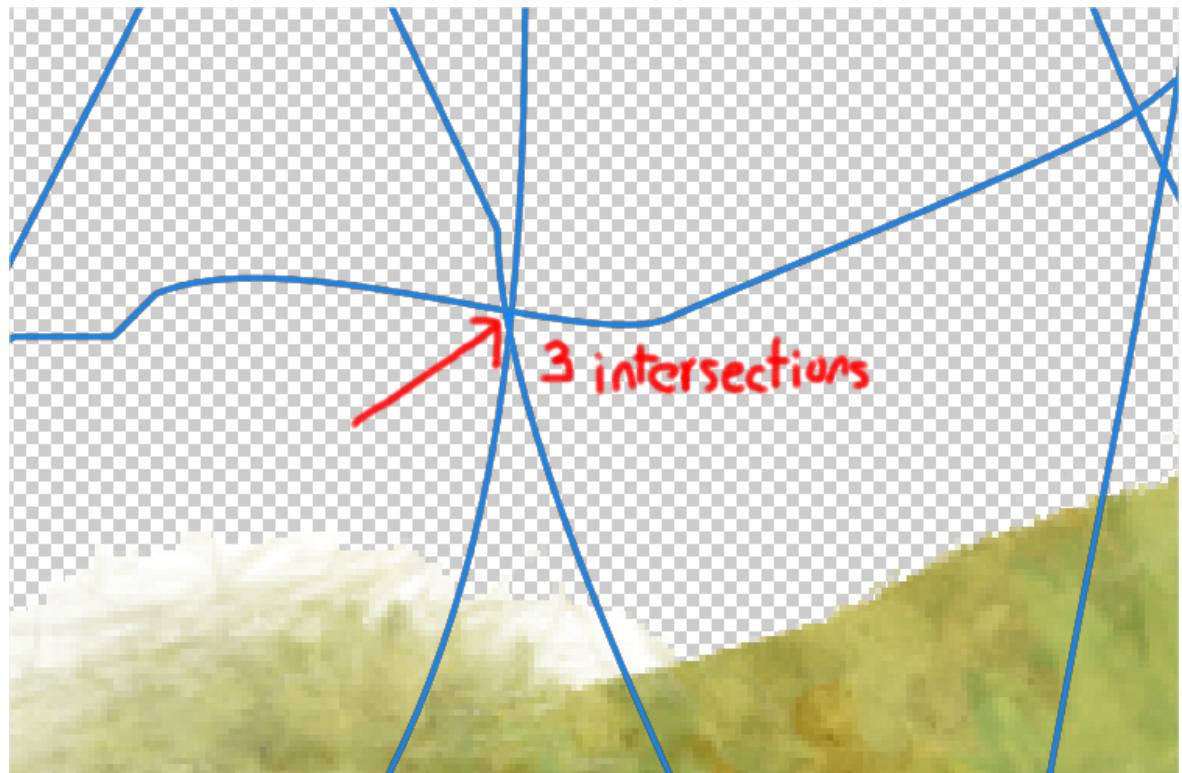


Problématique 2

Par la suite, on va combiner toutes les intersections qui sont trop proche l'une de l'autre.

Il y a deux raisons pour ceci:

1. Des intersections peuvent se retrouver très proche l'une de l'autre à cause des Tracés de l'utilisateur



- a.
2. Il est possible d'avoir deux intersections exactement aux même endroits à cause du fait qu'un b茅zier (A) termine sur un b茅zier (Z) et le b茅zier (B) qui suit (A) commence sur b茅zier (Z). Une intersection sera rajout茅 pour A et pour B d'apr猫s l'algorithme des intersections.



Problématique 3

Enfin, après toutes ces **Nodes** enlevées, il faut remettre à neuf les *Index* de chaque **Node**. Ces *index* sont supposés représenter leur position dans le *vector* de **Node**, mais vu qu'on a enlevé des **Nodes** les *index* ne sont plus bons. On en profite en même temps pour créer le *vector* de **Vector2F** qui sera nécessaire pour le Mesh.

4. Les Faces

Maintenant il faudra passer à travers toutes ces **Node** pour trouver les faces. Dans notre cas, une **Face** équivaut à un *cycle minimal*, que l'on peut exprimer autrement comme "un cycle qui ne contient pas d'autre cycle". Le problème de trouver tous ces *cycles minimales* s'appelle le Minimal Cycle Basis.

Heureusement, certaines personnes ont déjà proposé des solutions à ce problème et nous avons utilisé l'un d'entre eux :

- <https://www.geometrictools.com/Documentation/MinimalCycleBasis.pdf>.

L'implémentation de cet algorithme peut se retrouver dans la fonction MinimalCycleSearch. Voici comment elle fonctionne

```
func MinimalCycleSearch:
    faces = []
    nodes.SortBy(x position)
    forall current in nodes:
        if current.neighbours.empty:
            continue

        leftPos = current.pos - (x:1, y:0)
        // Tant que nous avons plus qu'un voisins on continue de chercher des
faces
        while current.neighbours.size > 1:
            neighbour = current.GetMostClockwise(leftPos)           // On
cherche le voisin qui est le plus à droite (ou direction horaire) de notre "current"

            face = []
            // Maintenant que nous avons le voisin qui est le plus à droite
on se remet à chercher à gauche avec CompleteFace
            CompleteFace(current, neighbour, face)
```

```

if !face.valid:           // Validation de l'état de la face. (voir
la méthode ValidateFace dans le script curveMeshGenerator.cpp pour plus d'info)
    current.removeNeighbour(neighbour)
    neighbour.removeNeighbour(current)
    continue

    current.removeNeighbour(neighbour)
    neighbour.removeNeighbour(current)
    faces.add(face)

    current.ClearNeighbours()

func CompleteFace(first, second, face):
    face.add(first.index)

    previous = first
    current = second
    while current != first:
        face.add(current.index)
        next = current.GetMostCounterClockwise(previous)

        previous = current
        current = next

```

Pour résumer:

- On cherche une première fois vers la droite.
- On cherche ensuite vers la gauche
 - Ceci nous permet de trouver nos faces, ou des *cycle minimal*.
- À chaque itération on enlève la branche entre *current* et *neighbour*.
 - Ce qui nous permet de ne jamais retrouver les mêmes faces.
- On fait ceci jusqu'à ce que *current* ai un seul voisin
- Un fois arrivé au cas du voisin unique on arrête de chercher des faces à partir de cette Node et on passe au prochain.

ALGORITHME DE GÉNÉRATION LINÉAIRE

Introduction

Un des premiers algorithme du projet permettait de générer rapidement un mesh en récupérant les contours de l'objet dans un layer de "Tracé" dans photoshop.

L'algorithme est très pratique pour certain objet de forme linéaire (objet régulier, immeuble, arbre, ...). Nous avons conservé l'algorithme malgré l'arrivée de l'algorithme basé sur les courbes des masques de vecteurs.

Fonctionnement de l'algorithme

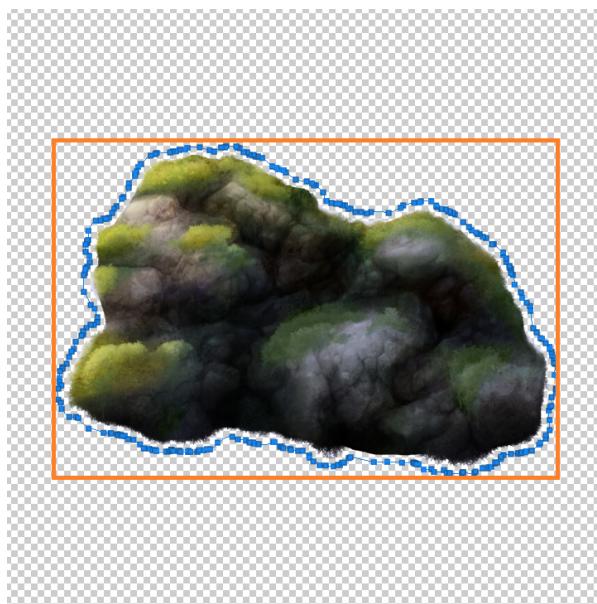
L'algorithme fonctionne en plusieurs étapes, il prend en paramètre plusieurs valeurs différentes

- La "**Bounding box**".
- Le nombre de polygones souhaité sur la **hauteur**.
- Une liste des **courbes de Bezier**.

On retrouve le point d'entrée dans le script **linearMesh ccp** avec la fonction *GenerateMesh*.

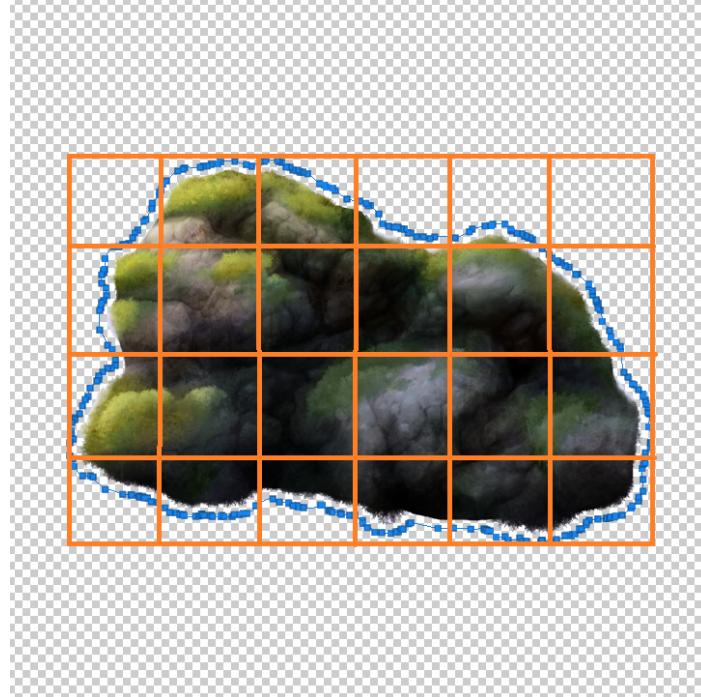
Paramètre De Précision Du Mesh

On récupère la hauteur de la bounding box. On divise cette hauteur par le nombre de polygone désiré sur la hauteur (valeur en paramètre). On détermine ainsi la taille des arrêtes de nos polygones que l'on appellera précision.



Génération De La Grille

Nous utilisons le point situé en haut à gauche de la bounding box, puis nous définissons des points horizontalement et verticalement à intervalle régulier de la taille de la valeur calculée dans l'étape précédente.



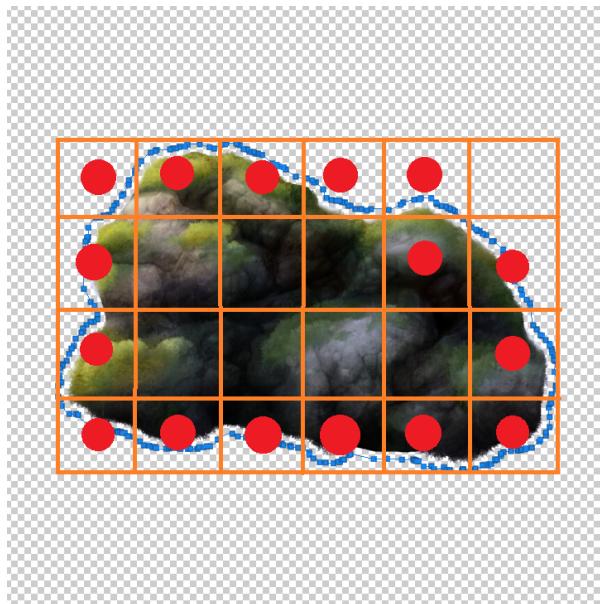
Voici la première étape de notre génération de mesh, Nous allons ajouter tout les points dans un tableau représentant les vextex. Le tableau est un tableau à deux entrées, ainsi un polygone à cette étape n'est défini que par l'index ligne / colonne du tableau.

Filtration Des Contours

Nous allons maintenant parcourir la courbe de contour du mesh (en bleu) et détecter toutes les intersections avec la grille. C'est à dire le moment où un point change de polygone. C'est principalement du calcul vectoriel permettant de savoir si un point est d'un bord ou de l'autre d'un vecteur. Le vecteur étant l'arrête du polygone au moment où un point est parcouru. On tient compte de l'orientation du déplacement pour faire le calcul, ce qui nous permet de faire le calcul avec la bonne arrête du polygone en cours. Lorsque qu'un changement de polygone est détecté on conserve la position de l'intersection ainsi que l'arrête intersectée (gauche, droite, haut bas).

Ainsi à la fin de cette étape chaque polygone intersecté par la courbe de contour contiendra l'information des arrêtes intersectées ainsi que la ou les valeurs de la position d'intersection.

Maintenant, nous allons simplement parcourir la liste des polygones définies par la position des points de la grille. Nous avons ainsi la liste de polygone de contour correspondant au point rouge.

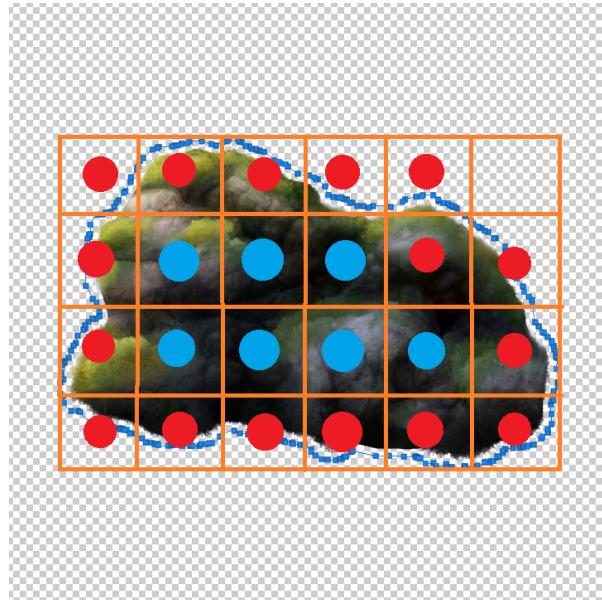


Identification Du Contenu

La dernière étape consiste à identifier les polygones appartenant au mesh. Pour se faire on parcours chaque ligne de la grille de gauche à droite à la recherche d'intersection.

Sur un ligne si une intersection est détectée nous identifions les polygones suivant comme appartenant au mesh, jusqu'à rencontrer le prochain polygone de contour sur la même ligne. L'algorithme arrêtera alors d'identifier les polygones suivant comme appartenant au mesh.

Les polygones suivant seront en l'absence d'intersection détectée, considérés comme n'appartenant pas au mesh. En cas d'une nouvelle détection d'intersection sur une même ligne le processus précédent recommencera.



Ce processus offre la possibilité de traiter les cas de forme creuse ou avec plusieurs zones de pixels distinctes séparées par une valeur d'alpha égal à 0, sur un même layer.

Construction Du Tableau De Polygone

À la fin de cette étape de filtration il suffit de parcourir les polygones identifiés et de remplir une structure MeshPoly, contenant les index des 4 vertex associés aux polygones identifiés.

Les curves

En fonction de la liste des points lus lors de la lecture du PSD nous utilisons la formule mathématique suivant pour calculer une série de point à intervalle régulier.

$$\mathbf{P}(t) = \mathbf{P}_0(1-t)^3 + 3\mathbf{P}_1t(1-t)^2 + 3\mathbf{P}_2t^2(1-t) + \mathbf{P}_3t^3$$

Cette série de point est utile au parcours du contour ainsi que dans le calcul des points d'intersections avec la grille.

Bounding box

Calcul De La Bounding Box Classique

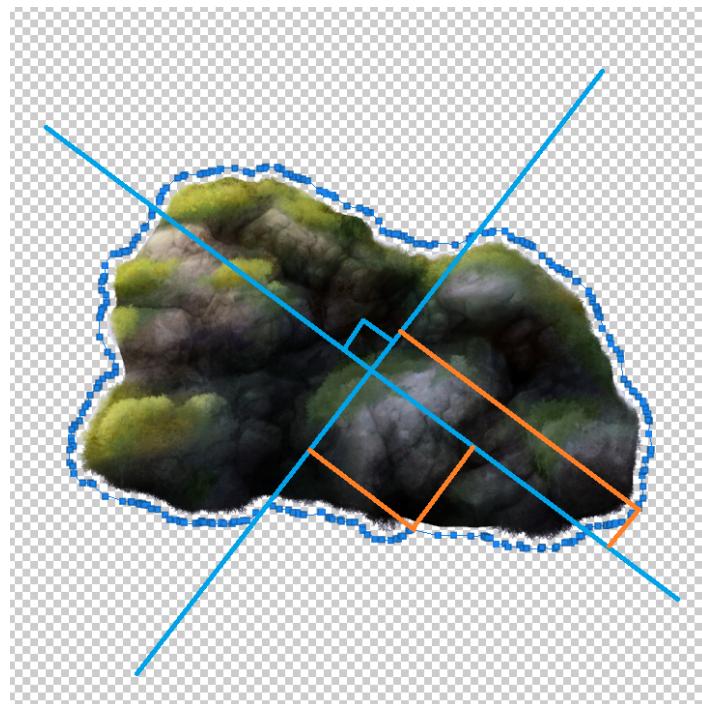
La bounding box est calculée par défaut selon l'orientation horizontal et vertical du calque. Son obtention consiste à parcourir tout les points de la courbe et d'identifier les points ayant les valeurs extrêmes en X et en Y. Une fois obtenue il suffit de prendre les valeurs et de les combinées pour obtenir les coordonnées des points permettant de tracer un cadre incluant tout les points de la courbe.

La notion de **valeur extrême** implique, la valeur la plus petite et la valeur la plus grande.

Calcul De La Bounding Box Orientée

Il est possible de donner une orientation au mesh. Pour ce faire un paramètre nous donne la valeur d'un angle. Nous calculons alors un vecteur à partir de cet angle ainsi que son vecteur orthogonal.

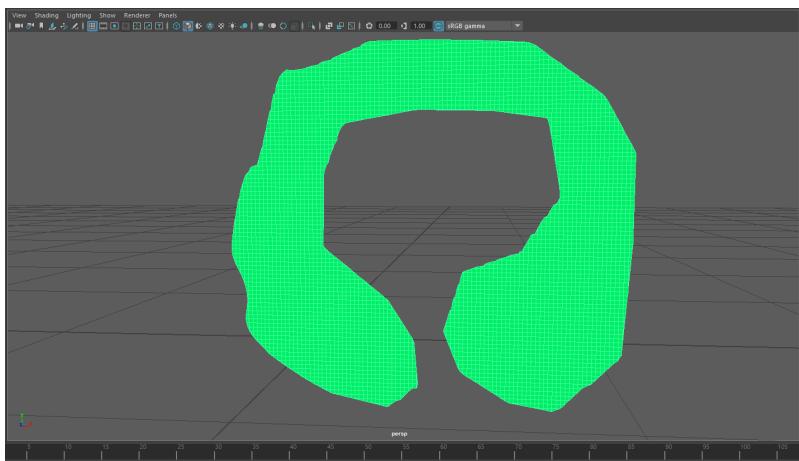
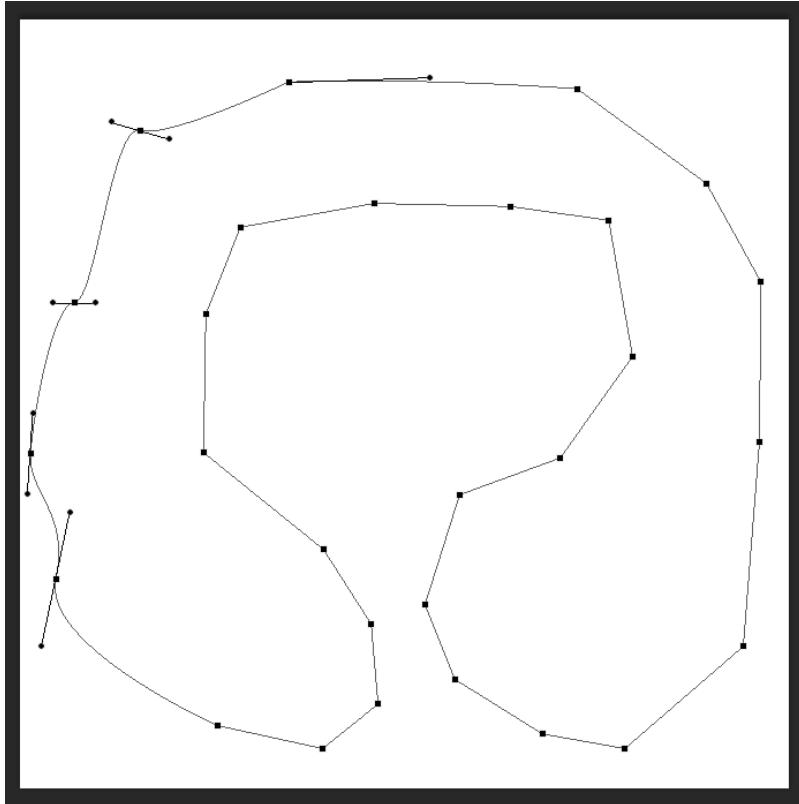
Par la suite nous faisons une projections des points de la courbe sur ces deux vecteurs. Nous procéderons alors comme avec une bounding box classique, nous combinerons les valeurs des positions des points projetés aux extrêmes de chacun des vecteurs. Nous obtiendrons ainsi les points définissant la bounding box orientée.



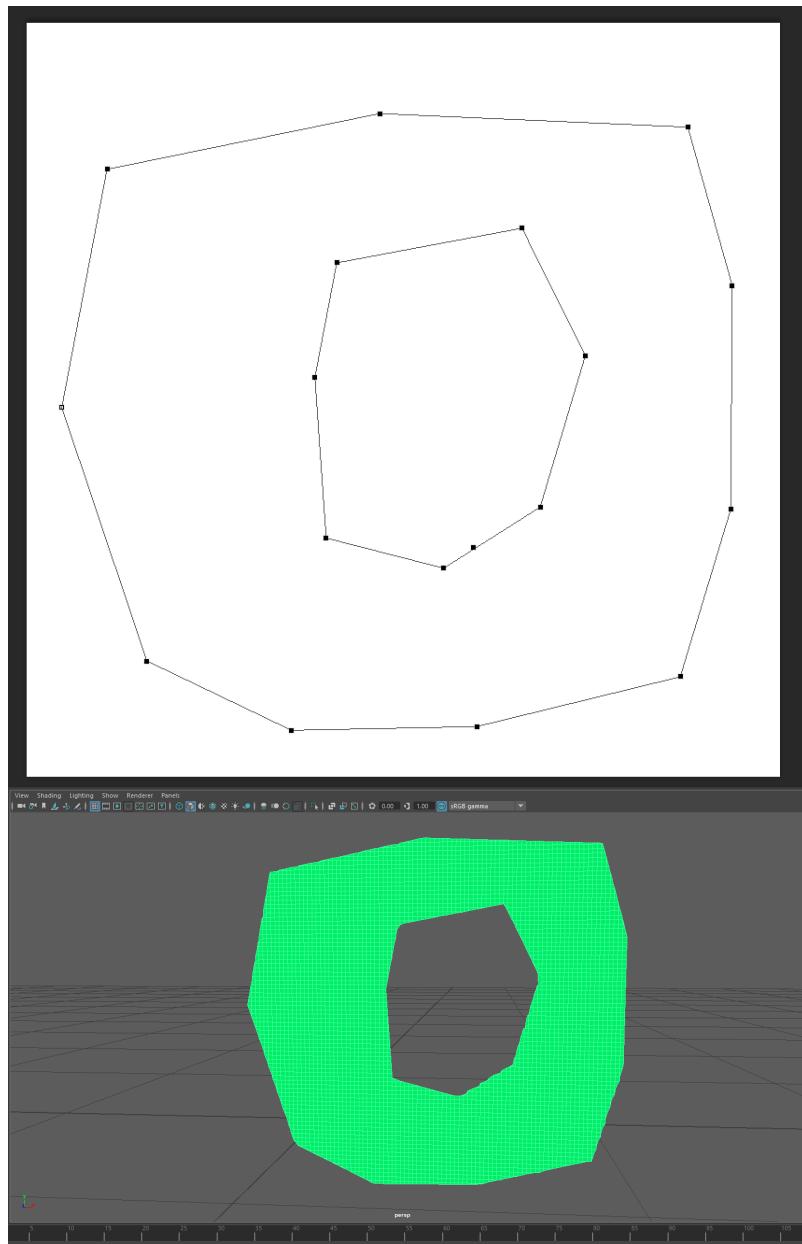
Cas particuliers

Comme précédemment cité, l'algorithme traite les cas suivant

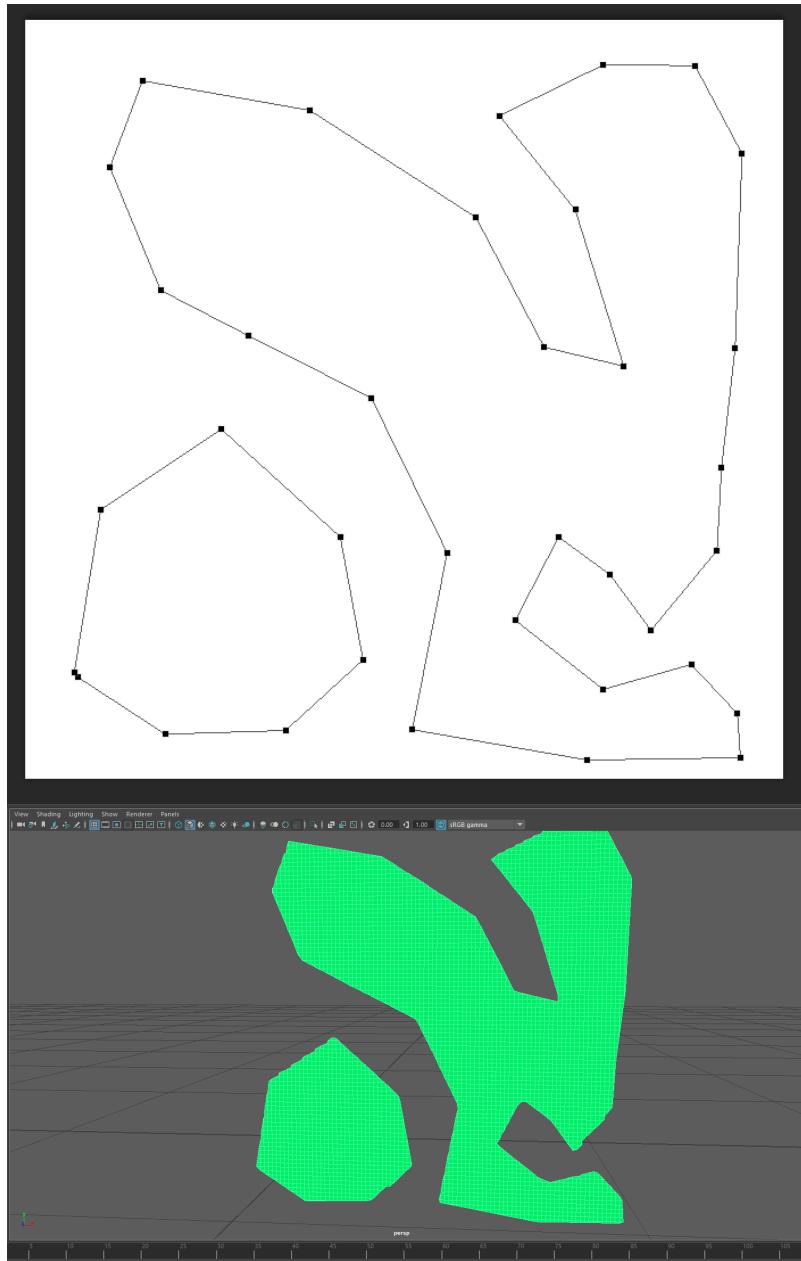
Les Formes Creuses



Les Formes Creuses Imbriquées



Multiple Forme Par Layer



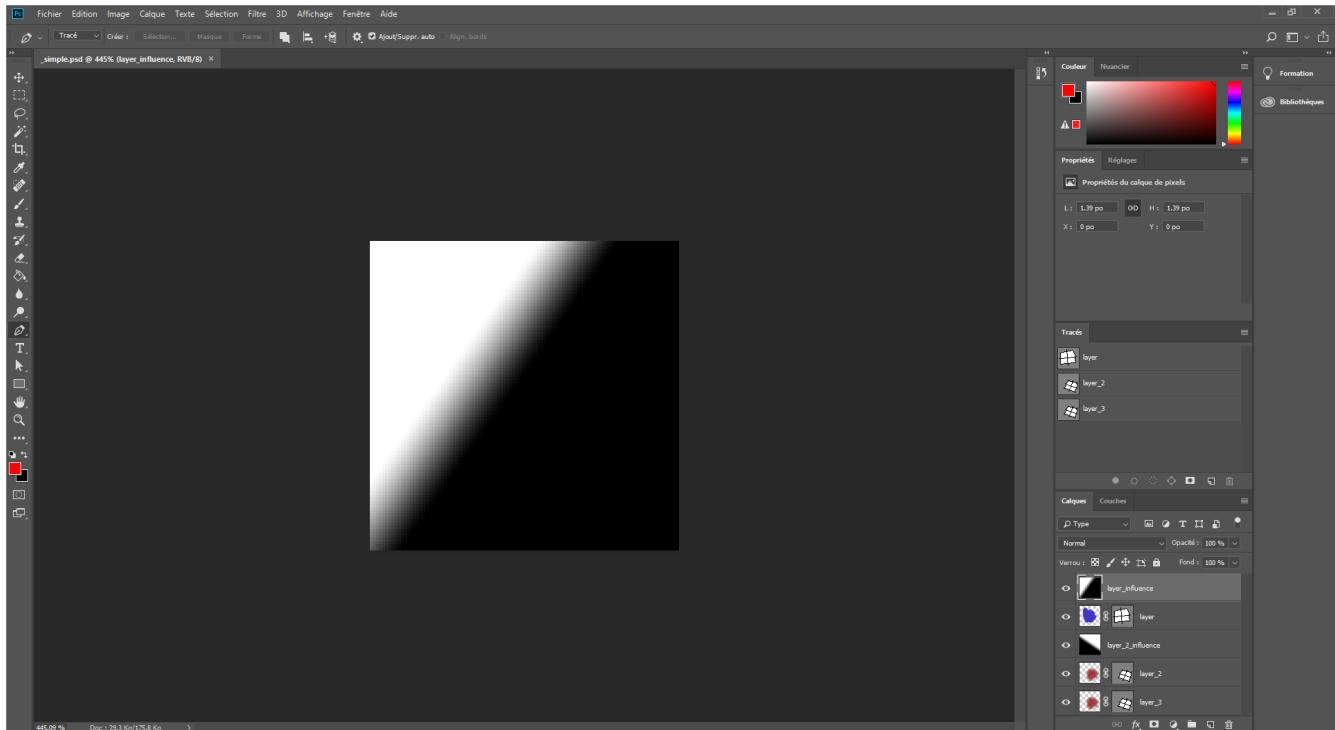
Disposant des points d'intersection il est possible de redéfinir la position des points des polygones pour obtenir un contour respectant la forme de la courbe.

INFLUENCE ALGORITHME

Introduction

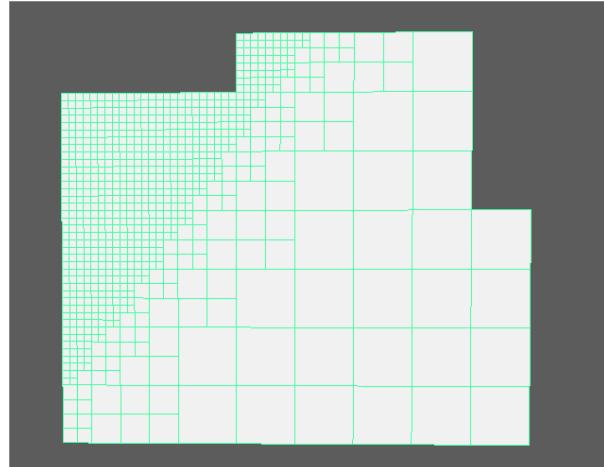
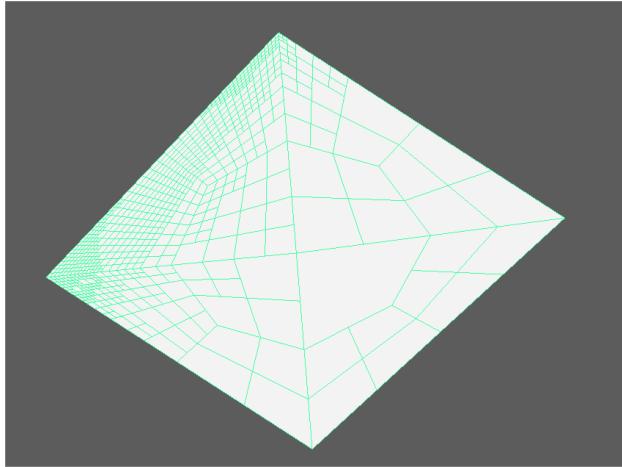
Une fois qu'un Mesh a été généré par le "[Linear Algorithm](#)" ou le "[Curve Algorithm](#)", il est possible de sous-diviser les faces avec le "**Influence Algorithm**". Cet algorithme prend la forme d'un calque en nuance de gris avec comme contrainte que le nom de ce calque soit le même nom que le calque original avec le suffixe "**_influence**".

Dans l'image ci-bas on peut voir que les calques "layer" et "layer_2" ont chacun un calque d'influence qui pourra s'appliquer.



Le calque d'influence fonctionne comme un mask, une face qui se retrouve au dessus d'un pixel blanc sera divisée (voir plus bas pour plus d'informations).

Voici le résultat de l'algorithme appliqué au calque "layer" (l'image de gauche est générée avec le "**Curve Algorithm**" et celui de droite avec le "**Linear Algorithm**") :



Cet algorithme comprend quelques petites étapes

qui seront ensuite combinées.

1. Construire la structure de données des faces.
2. Trouver si une face doit être plus subdivisée ou non.
3. Trouver le point au centre de la face.
4. Faire la division de la face.
5. Avertir les autres faces qu'une arête a été coupé en deux.
6. Combiner toutes les étapes précédentes ensemble.

Ces étapes et tout le code relié à l'algorithme peuvent trouver leur point d'entrer dans la fonction [SubdivideFaces](#) dans le script [influenceMesh.cpp](#). Suivre cette documentation sera plus facile si elle est accompagnée du code.

Construction des faces

Avant de commencer on débute en se construisant une structure de données

"std::queue<MeshFace> toProcess". On procède dans cette étape à une conversion de la structure de données Mesh, qui n'est pas très malléable, à un autre auquel on pourra facilement ajouter et enlever des éléments. Il aurait été possible et certainement plus optimal de procéder dans l'algorithme de génération sans créer une nouvelle structure de données, mais cet algorithme aurait été beaucoup plus difficile à lire et debugger.

- La classe **MeshFace** contient la liste d'index (qui indique quel vertex font partie de la face) et quelques fonctions pour faciliter l'algorithme.

Pour chaque **MeshFace**, on appelle la fonction "[SetShouldSubdivide](#)" qui va immédiatement décider si cette face devrait être divisée (voir la section suivante pour plus d'informations sur le sujet).

Est-ce que l'on doit diviser la face?

Comme il est expliqué dans l'introduction, un calque d'influence est utilisé pour diviser les faces. Voici du pseudo code rapide qui explique son fonctionnement :

```

// Une boîte 2 dimensions dans laquel la face est inscrite.
boundingBox = GetBoundingBox(vertices)
// La valeur la plus élevée (comprise entre 0 et 1) de la texture (à l'intérieur des
// limites de la boundingBox).
highestValue = HighestValueInMask(boundingBox)

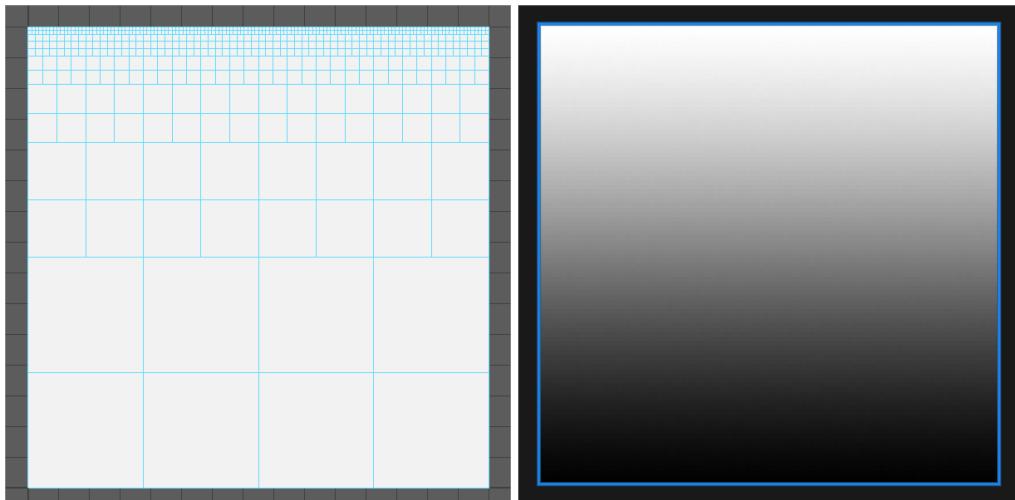
// La taille de la diagonale de la boundingBox, c'est ce dont on va se servir pour la
// comparaison.
boxSize = magnitude(boundingBox.topLeft, boundingBox.bottomRight)
// Ce calcul va nous donner la taille minimale qu'une face peut avoir à cet endroit.
currentValue = (minSize - maxSize) * highestValue + maxSize;

// Si la taille de la boundingBox est plus grand que la valeur, c'est qu'il faut encore
// la subdiviser.
return boxSize > currentValue

```

En appliquant cette formule à toutes les faces puis à toutes les faces qui sont divisées (récurivité) on aura redivisé toutes les faces en respectant le mask.

Voici un exemple super simple d'un seul carré qui est redivisé basé sur le calque d'influence que l'on voit à la droite. On peut voir que les carrés en bas de l'exemple sont moins divisés que ceux d'en haut.



Point Central

Pour diviser une face de la façon qu'on le veut, il faut d'abord trouver le point central à tous les vertex de la face. Pour ce faire, nous faisons la moyenne des vertex du polygone.

Supposons les faces suivantes avec leur vertex en orange et leur arrêtes en noir.



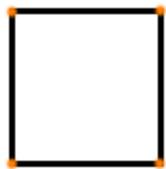
Une fois divisées elles vont ressembler à ça :



Diviser la face

Maintenant qu'on a le centre de la face, on peut la diviser en d'autre polygone.

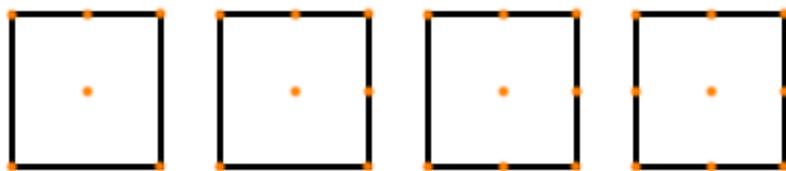
On commence avec une face standard :



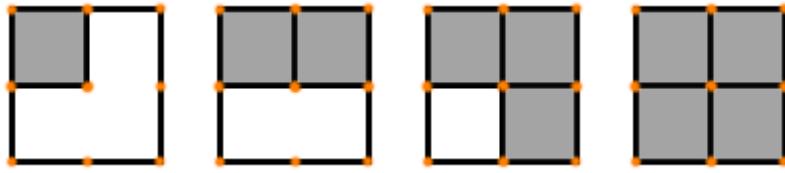
On trouve le point central :



Par la suite on rajoute un après l'autre des vertex entre chacun des vertex originaux :



Finalement on reconstruit les nouvelles faces à partir de ces résultats.



La face original est enlevé de notre *queue* de face et ces nouvelles faces seront rajoutées.

Il est à noté, qu'une face divisée donnera toujours N nouvelles faces, où N = le nombre de vertex.

Diviser les arrêtes

Après avoir divisé une face on se retrouve avec un problème : qu'est-ce qui arrivent aux faces qui se trouvent à côté de celle qui se font diviser?

Il faudrait donc trouver un moyen pour avertir ces faces qu'une de leur arrête a été coupé en deux. C'est ici que rentre "**SplitEdge**", une classe qui nous aide à nous rappeler quelle arrête a été coupé.

- "**SplitEdge**" contient trois index; les deux index de l'arrête originale et l'index du nouveau vertex qui entre-coupe maintenant l'arrête.

On introduit maintenant des nouvelles étapes dans la section 4 : On ne coupe pas une arrête qui a déjà été coupée, on le rajoute à notre face. En plus, lorsqu'on coupe une arrête on va chercher dans les faces "terminé" pour trouver une face qui a une arrête de coupée. Si on le trouve on le remet dans notre *queue*.

```
EdgeSplit split = FindEdgeSplit(vertex, vertex + 1)
if split != null:
    return split.SplitPoint
else:
    center = CenterPoint(vertex, vertex + 1)
    split = EdgeSplit(vertex, vertex + 1, center)
    allEdges.Add(split)

    MeshFace face = FindFaceWithEdge(split)
    if face != null:
        allFaces.Add(face)

return center
```

On fait aussi une opération similaire quand une face ne se re-divise plus. Avant d'être enlevé de la *queue* de face on cherche si elle a des arrêtes qui ont été coupées précédemment.

Combiner les étapes

Maintenant qu'on a toutes nos étapes on va les mettre ensemble :

```

class SplitEdge:
    start, end, center

    // Remplie avec toutes les faces.
    queue toProcess = BuildFaces()
    list splitEdges = {}
    list completedFaces = {}

    while !toProcess.empty:
        current = toProcess.pop()

        if current.shouldSubdivide:
            // Cette méthode va chercher à travers splitEdges pour voir si la face a
            besoin de diviser une de ses arrêtes.
            UpdateFaceWithDividedFaces(current, splitEdges)
            completedFaces.add(current)
            continue

        forall vertex in current.vertices:
            edge = FindEdge(vertex, vertex + 1)
            center = null
            if edge == null:
                center = Mid(vertex, vertex + 1)

                edge = SplitEdge(vertex, vertex + 1, center)
                splitEdges.add(edge)

            // Cette méthode va chercher à travers completedFaces pour
            trouver une face qui a une arrête qui a été coupée, cette dernière est enlevée de
            completedFaces et déplacée dans toProcess.
            ReturnCompletedFaceIfSplit(edge, completedFaces, toProcess)
            else:
                center = edge.center

                face.AddBetween(vertex, vertex + 1, center)

        forall vertex in current.vertices:
            toProcess.add(Face(vertex, vertex + 1, vertex + 2, current.center))
    
```

Ce pseudo code sert seulement comme explicatif simplifié de l'algorithme, je propose fortement de visiter le script [influenceMesh.cpp](#) pour comprendre plus en profondeur l'algorithme.

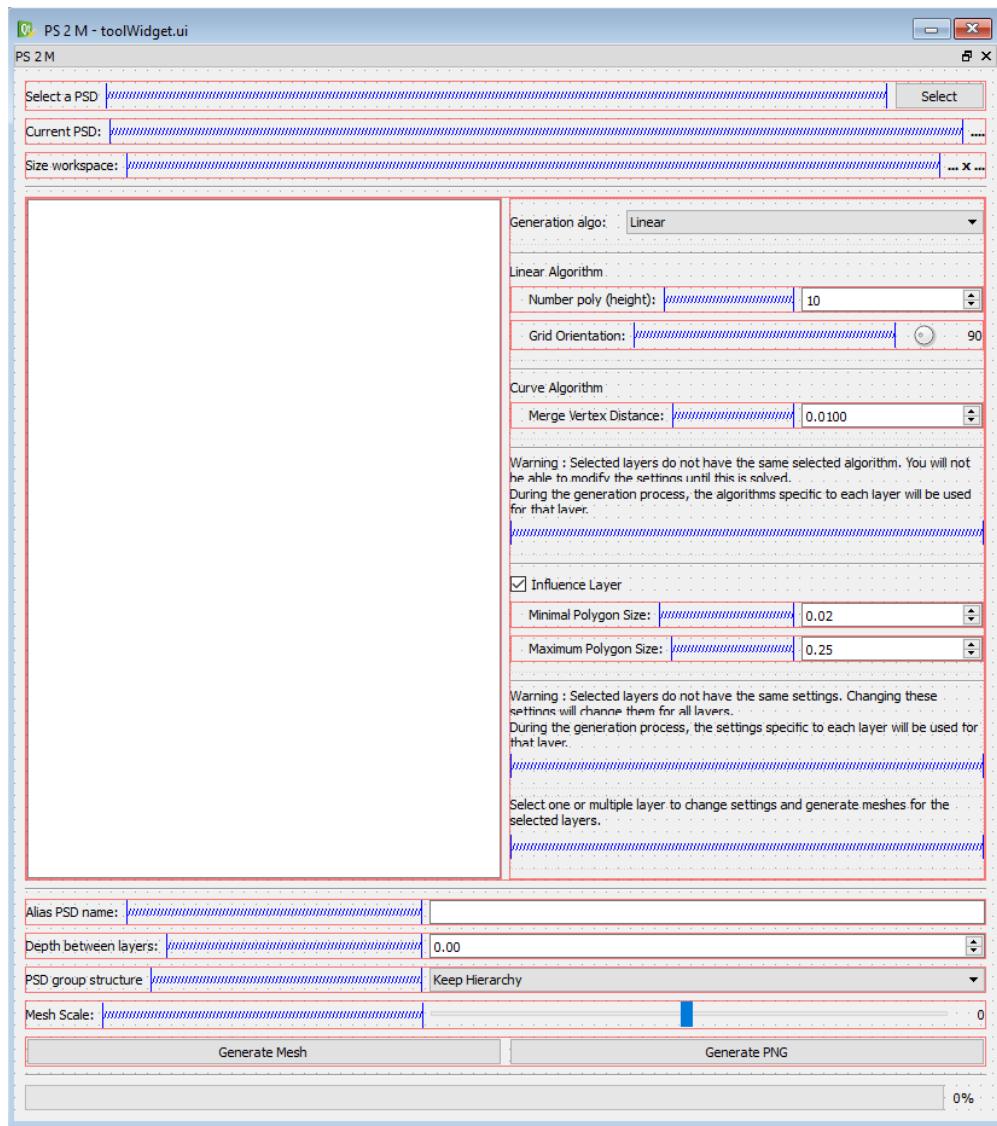
UI

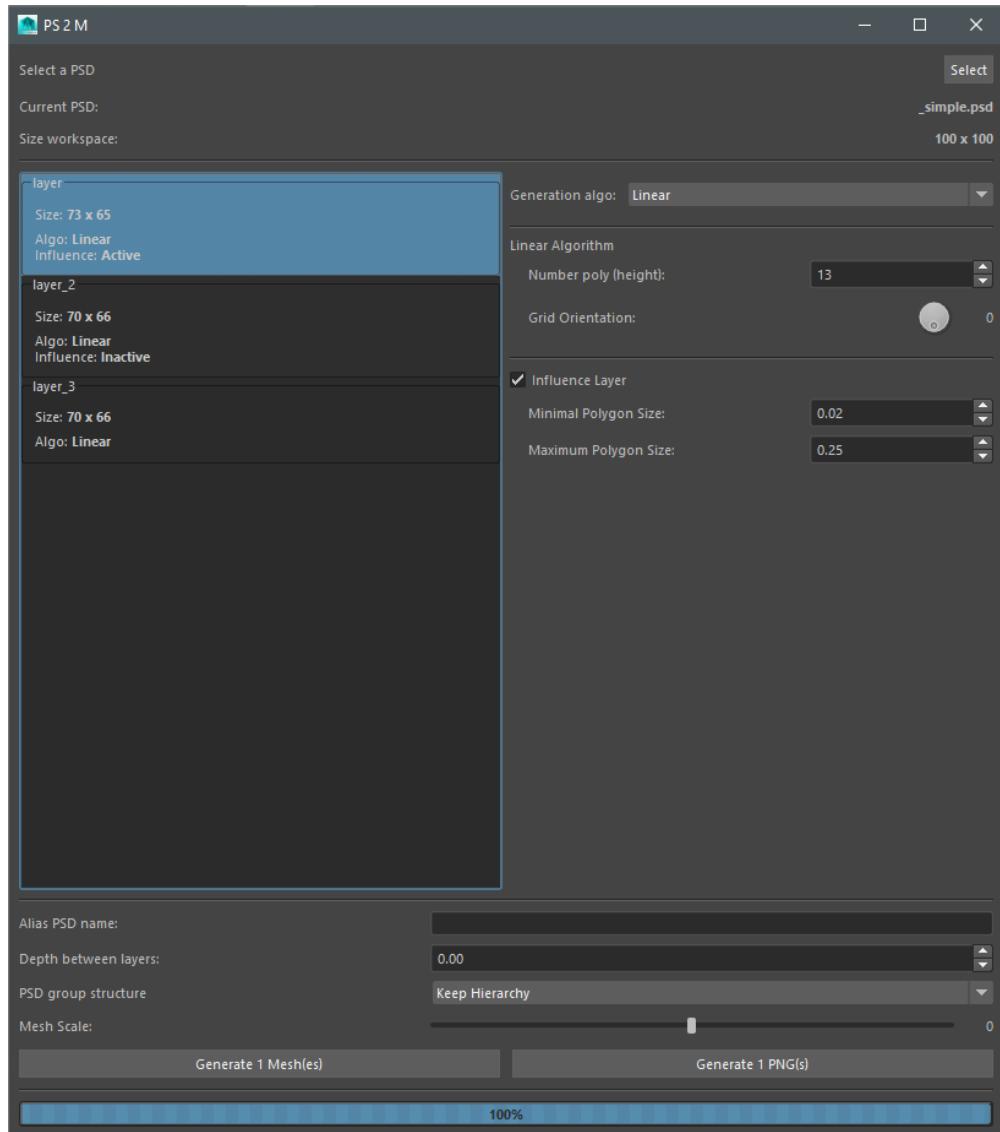
Introduction

Cet outil est livré conjointement à une interface qui peut être ouverte dans Maya, le logiciel 3D. Cette interface est construite dans [Qt Designer](#), un logiciel dédié aux interfaces.

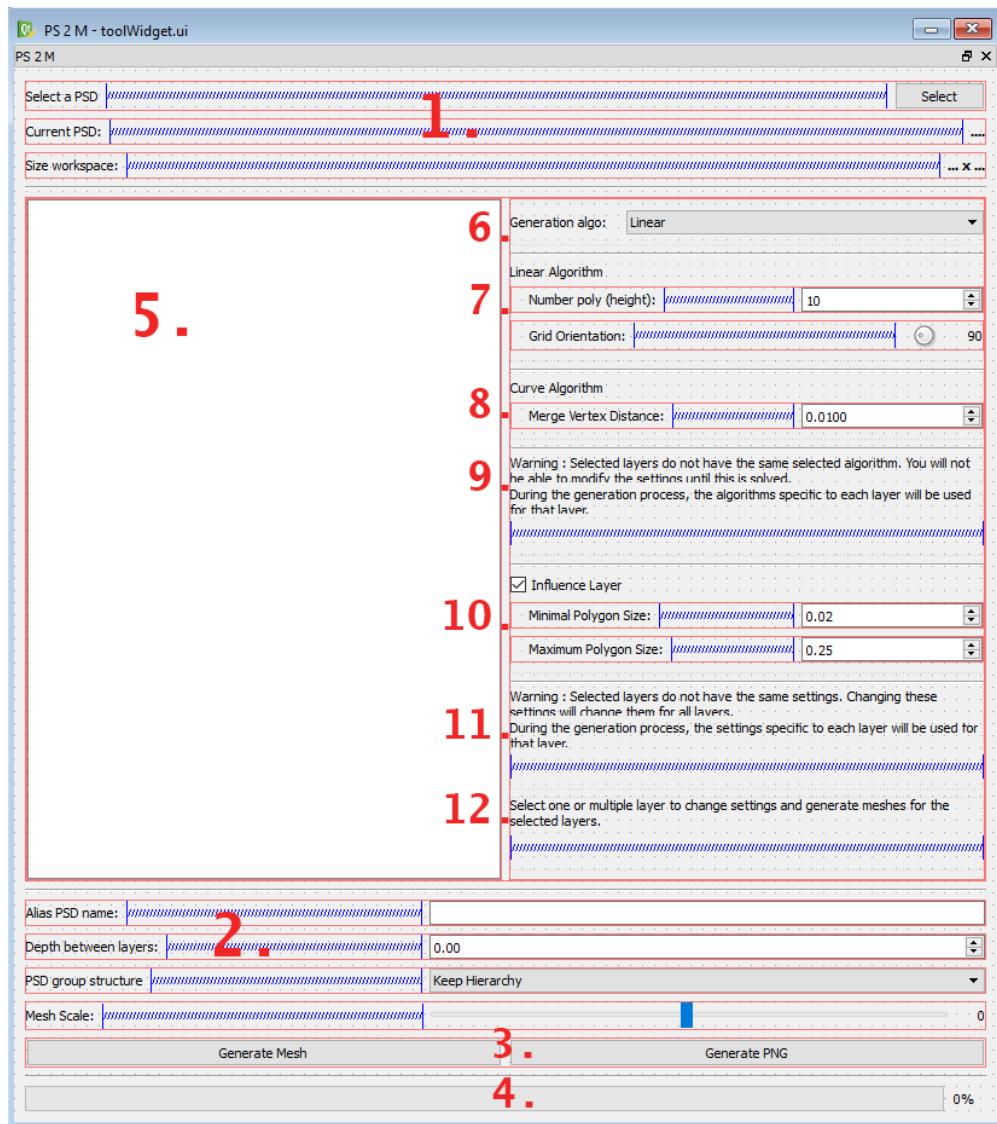
L'interface

Lorsqu'on travaille dans Qt Designer nous pouvons voir une interface de travail (en gris clair) qui devient par la suite une interface produit par Maya (en gris foncé / noir) :





Voici une explication de l'interface issue de Qt Designer :



1. Cette section est l'entête, elle permet de sélectionner et d'afficher les informations d'un fichier PSD.
2. Les paramètres en bas de la page sont globaux à tous les calques.
 - a. **"Alias PSD name"** : Le nom de remplacement du fichier PSD dans Maya. Si cette valeur est différente de "vide", la racine de la hiérarchie dans Maya aura ce nom au lieu du nom du fichier PSD.
 - b. **"Depth between layers"** : La distance en unité Maya entre chaque Mesh généré.
 - c. **"PSD group structure"** : Permet de décider si la hiérarchie dans le fichier PSD sera gardée, ou si tous les calques seront générés avec pour racine commune un "**transform**" au nom du PSD.
 - d. **"Mesh Scale"** : Un multiplicateur de la taille des Mesh générés. (à 0, la taille est multipliée par 1, à 0.5, la taille est multipliée par 1.5)
3. Ces boutons permettent de générer soit les Mesh soit les PNGs.
 - a. Ces boutons se mettent à jour basés sur les calques sélectionnés dans la section (5). Donc ils vont seulement générer ce qui est sélectionné, ou ***s'il n'est pas sélectionné il va tout être généré.***
4. La bar de progression qui affiche l'avancement des tâches.
5. Cette section va contenir la liste de tous les calques du fichier PSD. On peut aussi y voir certaines informations reliées aux paramètres de ce calque.

- a. Sélectionner un ou plusieurs éléments de cette liste va mettre à jour la section à droite de la liste. Les sections (6) à (12) vont être enlevé / rajouté dépendant de la sélection et des paramètres de cette sélection.
- b. Le changement de la sélection va aussi modifier l'état des boutons de la section (3).
- 6. Cette section permet de changer l'algorithme utilisé pour générer le Mesh.
 - a. Choisir "**Linear**" fera apparaître la section (7) et disparaître la section (8).
 - b. Choisir "**Curve**" fera apparaître la section (8) et disparaître la section (7).
- 7. Cette section contient les paramètres utilisés lors de l'algorithme "**Linear**"
 - a. "**Number poly (height)**" : Cette valeur décide du nombre de division que le Mesh aura en hauteur.
 - b. "**Grid Orientation**" : Cette valeur décide de l'orientation du Mesh. À noter que la valeur 0 et 180 donneront le même résultat.
- 8. Cette section contient les paramètres utilisés lors de l'algorithme "**Curve**"
 - a. "**Merge Vertex Distance**" : Cette valeur sert à l'algorithme lors de l'étape de fusion des Vertex qui se retrouvent trop proche les uns des autres.
- 9. Cette section contient un avertissement qui explique que la sélection dans la section (5) contient des calques qui ont des algorithmes différents.
 - a. Si cette section apparaît, les sections (7) et (8) seront enlevées.
- 10. Cette section contient les paramètres utilisés lors de l'algorithme "**Influence**".
 - a. Cette section apparaît seulement si l'un des calques sélectionnés a un calque d'influence associé.
 - b. Cette section peut être désactivé, et l'algorithme "Influence" ne sera pas appliquée.
 - c. "**Minimal Polygon Size**" & "**Maximal Polygon Size**" : ces deux valeurs représentent les tailles des polygones reliés respectivement au blanc et noir d'un mask.
- 11. Cette section contient un avertissement qui explique que les calques sélectionné n'ont pas les mêmes paramètres.
 - a. Si cette section est affiché, les paramètres dans les sections (7), (8) et (10) représentent le premier calque de sélectionné.
 - b. Si un paramètre est modifié, il sera modifié pour tous les calques.
- 12. Cette section contient un avertissement qui explique qu'aucun calque n'est présentement sélectionné.
 - a. Si cette section apparaît, aucun autre ne sera visible.

La sérialisation

Les paramètres des fichiers PSD et leurs calques sont sérialisés et sauvegardés dans un fichier. Ce fichier est au format JSON et est enregistré dans le dossier qui accompagne le PSD.

1. Le format JSON a été choisi pour sa simplicité et facilité de trouver une librairie qui est rapide et minime.
 - a. De plus l'avantage du JSON (versus une sérialisation binaire) est que l'information dans le fichier est lisible et modifiable par un utilisateur facilement.
2. Lorsqu'un fichier PSD est sélectionné dans l'interface un nouveau dossier sera créé avec le même nom que le PSD dans le même répertoire.
 - a. Ce dossier est donc utilisé pour la sérialisation des paramètres et pour l'exportation des PNG.

La sérialisation des paramètres ne se fait pas automatiquement, on doit entrer par code quel valeur à rajouter ou non. Lorsqu'on veut rajouter une nouvelle valeur dans la sérialisation il faut :

1. Le rajouter dans la fonction "**GlobalParameters::DeserializeContents**" dans la section de code correspondante. Cette fonction prend un objet JSON en paramètre et assigne les paramètres à partir de celui-ci.
2. Le rajouter dans la fonction "**GlobalParameters::SerializeContents**" dans la section de code correspondante. Cette fonction crée un objet JSON et assigne les paramètres à celui-ci.

3. De plus, si la nouvelle valeur est global, elle pourra être rajouté à "**GlobalParameters::SetDefaultValues**" aussi.

Les sections

Pour faire apparaître et disparaître les différentes sections de l'interface nous appliquons une logique différente à chaque section. Cette logique est appliquée dans la fonction "**ToolWidget::UpdatePanels**".

- Génération Algo (6) : S'il y a au moins un calque de sélectionné.
- Linear Algo (7) : Si tous les calques sélectionnés sont de type Linear.
- Curve Algo (8) : Si tous les calques sélectionnés sont de type Curve.
- Multiple Algo (9) : S'il y a plus qu'un type d'algorithme de sélectionné.
- Influence Layer (10) : S'il y a au moins un calque de sélectionné qui a un calque d'influence d'associé.
- Paramètres Multiple (11) : S'il y a au moins un calque de sélectionné et que les valeurs sont différentes à travers ces calques.
- Paramètres Vide (12) : S'il n'y a aucun calque de sélectionné.

Les comparaisons

Un message d'erreur affiche que certains paramètres selon les calques sélectionnés ne sont pas égaux. Ceci permet à l'utilisateur de savoir que s'il modifie cette valeur, elle sera modifiée pour tous les calques de sélectionnés. Si aucun changement n'est fait, les valeurs resteront comme elles le sont déjà. De plus, lors de la génération, les paramètres de chaque calque sont utilisés et non ceux qui sont affichés.

Trouver les valeurs différentes ne se fait pas automatiquement, la formule est fait dans la fonction "**ToolWidget::AreSelectedValuesDifferent**". Donc, lorsqu'une nouvelle valeur est rajoutée aux paramètres il faut aussi le rajouter dans cette fonction pour qu'elle soit prise en compte.

Les calques

Chaque calque dans la section (5) a une description qui lui est propre. Voici ce qu'elle contient :

- Le nom du calque.
- La taille tronquée du calque.
- Le type d'algorithme qui sera utilisé.
- Si le calque a un layer d'influence d'associé, il sera aussi affiché.

Les fonctions suivantes sont disponible pour modifier la description :

"ToolWidget::ApplyLayerDescription" et **"LayerParameters::UpdateDescription"**.

PARSER PSD

Introduction

La création du "parser" de PSD est basé sur la documentation de Adobe concernant le format du fichier.

- <https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/>

Document spécification présentation

Le lecteur de PSD s'occupe de transcrire le contenu dans un format de donnée en C++, permettant d'utiliser ces données dans d'autre module C++.

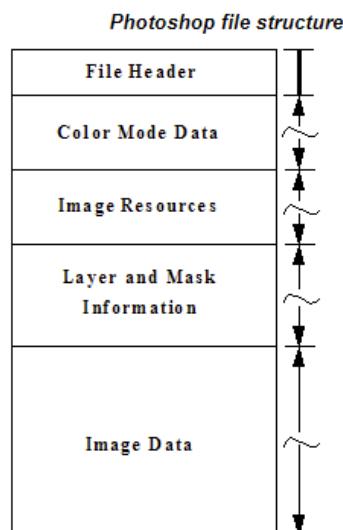
L'intérêt dans le projet était d'aller récupérer les données suivante:

- Dimension de la zone de travail.
- Les informations de "Tracé général".
- L'information de chaque layer, dimensions / nom / effets.
- Les textures de chaque layer indépendant.
- Les masque de vecteur associé au layer.
- La Hiérarchie de Group pour les calques.

Nous avons utilisé la documentation suivante pour lire et construire la structure de donnée.

- <https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/>

Voici la composition d'un fichier PSD, nous avons essayé de conserver des noms similaires pour les structures de données destinées à la réception du contenu.



"File Header"

- Les propriétés de base du PSD, dimension.

"Image Resources"

Cet espace de ressources regroupe tout les informations du psd liés aux opérations générale de photoshop. Elles sont enregistrées par bloc de donnée, chacune étant différents.

Dans le cadre de notre projet nous ne lisons que les structures qui nous sont utile, le reste n'est pas conservé.

- Les informations de Tracé général. ("Paths")
 - L'id pour les Tracés est : **2000-2997**.
- Les informations de la résolution.

"Layer & Mask Information"

L'information des layers est fragmentées en deux parties:

- Une liste de toutes les propriétés de chacun des layers.
 - L'information de chaque layer, dimensions / nom / effets.
 - Les masques des vecteurs associés au layer.
 - La Hiérarchie de Group pour les calques.
 - Par défaut chaque entrée dans la section "**Calques**" de photoshop est un layer.
 - Un layer est par défaut une texture.
 - Autrement une valeur type lui est assignée pour spécifier si c'est **Group** ouvert ou fermé (Dossier contenant des calques).
- Une liste des textures ordonnées par layer et séparées par channel. Ainsi pour chaque layer l'écriture est faite par channel, selon l'ordre ARGB.
 - Les textures de chaque layer sont indépendantes.
 - Différents formats peuvent être enregistrés, nous traitons Raw et RLE.

! La lecture de "Layer & mask Information" offre plusieurs possibilités. En effet les layers peuvent être contenue dans la section "Layers records" dans 70% des cas. Des exceptions arrivent et la section des "Layers records" est vide. L'information est alors stocké de la même manière (structure "*Layer info*" + "*Channel image data*") mais dans un "Additional Layer Information" introduit par l'ID "**Lr16**".
 Donc si "*Layer info*" présente un taille de 0, vérifiez la taille de "*Global layer mask info*" puis passez au "*Additional Layer Information*" tout est dans une ID.

Image Data

- Cette section contient les données de la texture fusionnée de chaque layer, elle est au même format que celui des layers.
- Aucun intérêt dans le projet de lire cette section, elle n'est pas enregistrée.

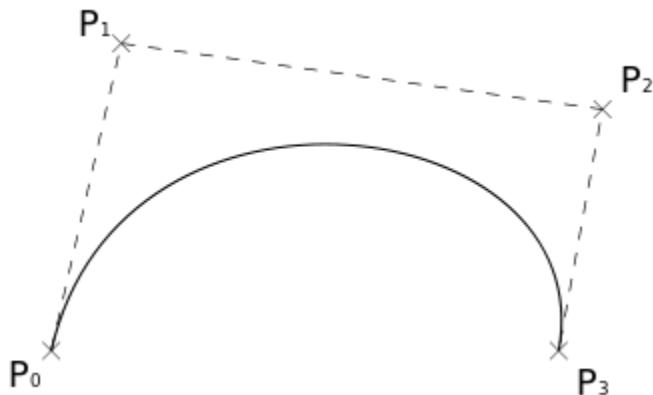
Dépendance avec "Util"

Le projet utilise la notion de courbe de bázier que l'on retrouve dans Photoshop. C'est courbes sont archivées dans le PSD selon la notion de courbe "**fermée**" ou "**ouverte**". Les courbes de bázier sont en réalité une fonction mathématique permettant de dessiner une série de point en fonction de point d'ancrage. Ce sont ses points qui sont enregistrés dans le fichier Photoshop et que nous utilisons dans ce module pour reconstruire les Courbes.

L'avantage des courbes de bázier réside dans le fait que peu importe la résolution, chaque trait n'est pas mémorisé comme une suite de pixel, mais comme une valeur obtenue par calcul.

On parle de courbe de bázier cubique dans notre cas:

- [https://fr.wikipedia.org/wiki/Courbe_de_Bázier](https://fr.wikipedia.org/wiki/Courbe_de_B%C3%A9zier)



"Util" contient donc la structure de stockage d'un point. Dans notre exemple:

- **P1** est le point de sortie d'influence de **P0**.
- **P2** est le point d'entrée d'influence de **P3**.
- **P0** et **P3** sont des points d'encrage, soit les points utilisés pour le tracé de la courbe.
- **P1** et **P2** sont des points d'influence du tracé selon la formule mathématique.

Soit pour un point P il existe toujours un point P1 et P2 associés à ce point. C'est ce que nous conservons.

Si les points P1 et P2 ne sont pas apparent c'est qu'ils ont la même valeur que leur point d'encrage.

On retrouve l'utilisation de cette structure de données dans Le projet "Mesh generator" pour calculer les courbes et trouver les intersections.

Références

Voici une liste des liens utiles pour la conception de ce lecteur de PSD.

- <https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/>
- <http://telegraphics.com.au/svn/psdparse/trunk>

EXPORT PNG

Introduction

Dans le cadre du projet nous voulions exporter les textures de chaque layer dans un format pratique pour Maya.

Nous avons essayé certains format comme le "iff", un format non compressé que maya intègre.

Les besoins d'utilisation nous ont ramené vers notre choix initial le PNG. Un format compressé gérant très bien le channel d'alpha pour la transparence.

Notre première tentative

Le format iff présentait l'avantage de pourvoir charger la texture directement lu dans du PSD directement dans le gestionnaire des textures de maya.

- Pas besoin de librairie de compression.
- Moins d'opération de transformation des textures.
- Directement en lien avec l'outil de destination, MAYA.

Des fonctions de maya permettaient d'enregistrer sur le disque l'image au format iff. Ce format non compressé créait des fichiers de taille importantes sachant que chaque valeur de pixel était écrite.

à l'utilisation maya refusait de charger plus de 5 textures de 4096 par 4096 simultanément.

C'est à partir de ce point que nous avons regardé le format PNG.

Information sur le PNG

Le Png est un format pratique de compression, particulièrement pour la qualité de l'image mais aussi pour sa gestion du channel d'alpha utile en animation pour conserver la transparence.

La documentation suivante dispose de l'information sur le format du PNG:

- https://fr.wikipedia.org/wiki/Portable_Network_Graphics

Dans le cadre du projet nous lisons la texture soit en RAW soit au Format de compression RLE, puis nous la stockons de manière à pouvoir la préparer vers un format spécifique à la compression en PNG.

Nous stockons 4 tableaux contenant les valeurs de chaque pixel par channel RGBA. Lorsque nous le donnons à la librairie PNG, nous transformons cette donnée en un tableau de pixel où les valeurs de chaque channel pour un pixel se suivent selon l'ordre RGBA.

Choix de la librairie

Nous avons envisagé d'intégrer la librairie **Libpng** dans le projet, c'est une librairie pratique mais assez importante.

- <http://www.libpng.org/pub/png/libpng.html>

Nous avons donc orienté notre choix vers **Lodepng** présentant plusieurs avantages.

- <https://lodev.org/lodepng/>
- Elle dispose d'un simple script.
- Aucune dépendance vers une librairie de compression.

Génération Des Textures

Un des avantages de faire la génération de la texture via le plugin dans maya est la rapidité de compression que nous avons en utilisant le script intégré.

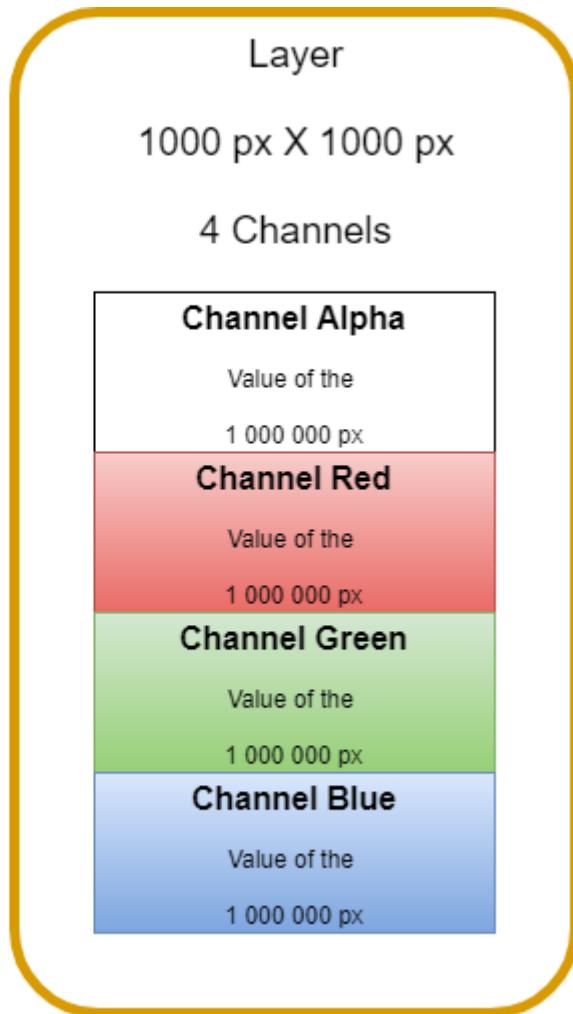
En effet l'utilisation de la compression par scripting dans Photoshop est beaucoup plus longue. Le format de compression proposé par **lodepng** équivaut au format de compression le plus volumineux proposé par Photoshop pour la compression Png.

L'utilisation de ce script:

- Simplifie l'intégration de la fonctionnalité dans le projet.
- Centralise les opérations à même le projet.
- Augmente significativement le temps d'exportation des layers.

Conversion Des Données

Un psd archive pour chaque layer toute les valeurs d'un channel à la fois soit pour 4 channels nous avons les données comme suit:



Pour permettre au Convertisseur de Png de comprendre la donnée nous modifions la texture conservée en écrivant l'information comme suit:

- Texture = **Layer * Valeur** d'un pixel.
- Valeur d'un pixel = valeur **RGBA** concaténé.

Photoshop conserve ses layers recadrés en fonction de l'alpha mais aussi compressé au format RLE.

Lors de la lecture du layer, nous l'enregistrons recadré mais au format Raw. Le recadrage nous permet de sauver de la mémoire. Le format Raw n'a pas de compression appliquée.

Pour répondre au besoin du projet nous transformons la texture à la taille de la zone de travail du PSD en remplissant les valeurs des pixels absent par 4 bytes de valeur 0, soit un pixel d'alpha, ainsi nous aurons des UVs correspondant visuellement à notre mesh. Puis nous appliquons la compression Png avant de l'enregistrer à l'aide de LodePng.

MAYA COMPONENTS (ÉDITEUR ET DONNÉES)

Introduction

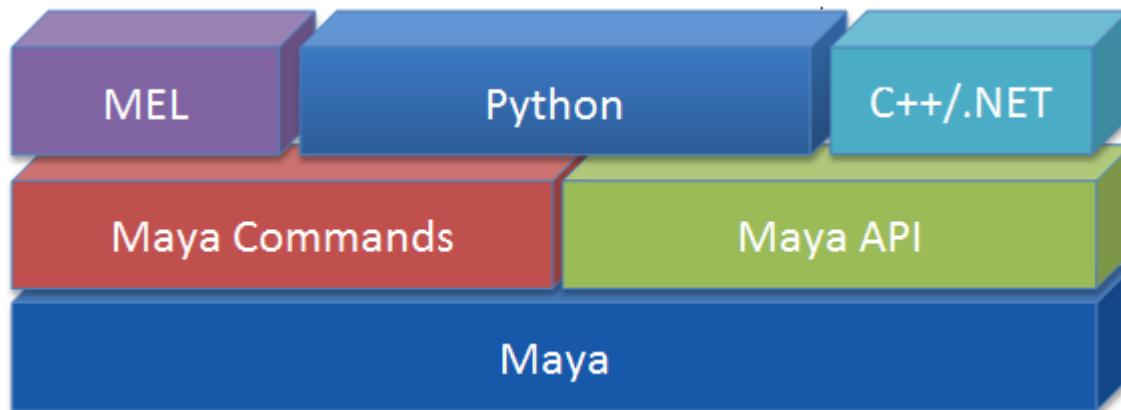
Cette section représente une partie importante du projet, elle gère le chargement du plugin dans maya, les commandes, le lien entre la lecture du PSD et les algorithmes de génération, le UI et ses paramètres. Mais surtout elle crée tous les composants qui permettront à maya d'afficher un mesh et sa texture.

Maya expose via l'API de nombreuses options, mais trouver comment le cœur de maya fonctionne demande un temps d'adaptation.

Cette documentation tente de vous donner les bases essentielles pour naviguer dans les fonctions de maya.

Détails des options de scripting

Maya est composé de plusieurs couches permettant d'accéder aux différentes ressources internes.



Nous avons pris la décision de développer le plugin avec l'API de maya en C++. La documentation suivante précise que l'utilisation de l'API offre un réel gain de performance au niveau des temps d'exécutions des opérations.

- https://help.autodesk.com/view/MAYAUL/2018/ENU/?guid=__files_API_Introduction_htm

Dans le cadre du projet il existe quelques objets créés via l'API par un appel d'une "Maya Command".

L'appel d'une command passe par la classe MGlobal, colonne vertébral de l'API.

Exemple:

- `"MGlobal::executeCommand("ls -sets", result);"`

Les différents objets de l'api

Les Types D'objets C ++

- Maya objects (**MObject**)
 - Tout est un MObject (curve, surface, Dag nodes, shader,).
 - Ce ne sont que des pointers vers différents sous composants.
 - Ne pas garder de Référence sur un MObject, l'utilisation des itérateurs est recommandé lorsque l'on recherche un objet plus précis.
- Fonction sets (**MFn**...)
 - Ces objets sont en réalité les objets permettant de faire des opérations sur un objet du type correspondant à ce qui suit le **MFn**.
 - Exemple **MFnMesh** permet de faire des opérations sur les mesh.
 - Ils sont importants car ils opèrent sur les structures de données de l'objet, tout en faisant des opérations auprès de service plus généraux de maya.
 - **MFnMesh** modifie ou crée les données de mesh associées à un objet identifié, tout en référençant cette donnée auprès du système de gestion de la Géométrie de maya.
- Proxy objects (**MPx**...)
 - Type d'objet dont il faut dériver pour créer un nouveau type d'objet reconnue et géré par maya.
 - Ce sont des objets conteneurs de la donnée.
 - Ajouter un command nécessite d'hériter de **MPxCommand**, alors Maya ajoutera cet objet au système de gestion des commandes.
- Wrappers (**M**...)
 - Les **wrappers** sont des classes d'opérations souvent sur des propriétés indépendante des objets de maya ou des systèmes généraux de maya.
 - Exemple des classes d'opération de mathématique.
- Iterators (**MIt**...)
 - Ces classes sont des Ittératrices vers les différents systèmes de maya et des objets gérés par ces mêmes systèmes
 - Exemple il existe un ittérateur sur la géométrie des mesh, il en existe un vers les nodes éditeurs.
 - Se sont des objets fréquemment utilisés pour récupérer le contenu d'un objet et lui appliquer des opérations.

Les Classes Pratique

- **MPxCommand**
 - Dériver de cette classe permet de créer de nouvelle commande reconnue par maya.
 - Une structure permet d'appeler l'exécution d'opération.
- **MGlobal**
 - Classe maître de Maya, contient une liste importante d'opération de haut niveau.
 - Affichage de log.
 - Suppression d'objet au niveau de l'éditeur avec la donnée référencée dans les systèmes centralisés.
- **MDagPath**
 - Référence vers l'objet **DAG** (Graphe de dépendance acyclique), soit la nodes éditeur des objets maya.
- **MSelectionList**
 - Permet de stocker une liste de composant présent dans une scène.
- **MItDag/MItDependencyGraph**
 - Les deux sont utiles pour parcourir les nodes éditeur et les différentes connexions entre les objets.

- La classe **MItDependencyGraph** facilite le parcours de node en node.
- **MPlug**
 - Représente les entrées et sorties d'un node éditeur.
 - Pratique pour gérer les connexions entre les différents composants.
- **MDGModifier / MDagModifier**
 - **MDGModifier** permet d'effectuer des opérations sur la partie visuel des objets maya (node éditeur).
 - Elle gère la création et la modification des nodes en appliquant un liste d'opération sur d'un objets. Gestion du **Undo / redo**.
 - **MDagModifier** présente les mêmes fonctions mais de manière plus spécifique pour la donnée contenue dans les objets.

Gestion des systèmes généraux

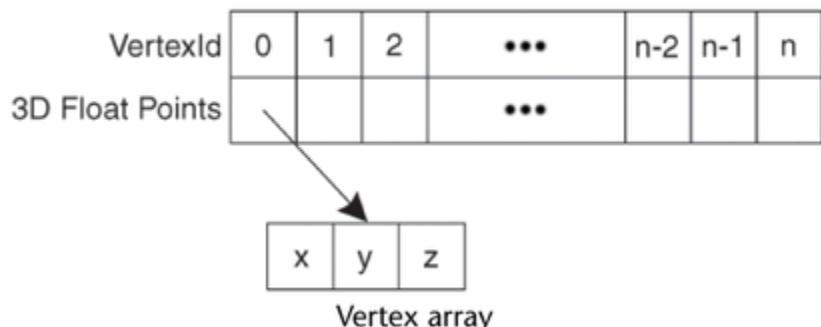
Maya intègre de nombreux manager implicite et inaccessible. La gestion des shaders, la gestion des géométries, celle des textures et des nodes. Il ne faut pas hésiter à utiliser les itérateurs pour parcourir l'ensemble du projet et raffiner cette recherche et évaluant les "**kType**" des objets et réduire le spectre de travail.

Il est commun de vérifier si des propriétés, sont présentes pour sélectionner des objets.

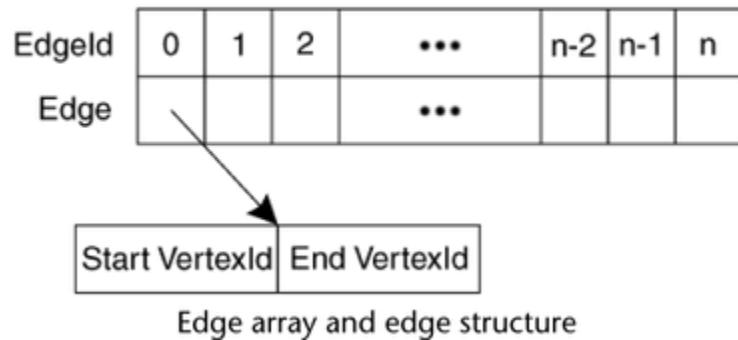
Création d'un mesh et de ses composantes

Voici la structure d'un mesh tel qu'illustré par la documentation de maya.

Les vertex sont stockés sous la forme d'un tableau ou les valeurs sont enregistrées sous la forme d'un tableau de 3 floats. **L'index** du vertex est important car il sera utilisé comme référence dans la construction des polygones.



La spécification des arrêtes, permet de manipuler le lien entre deux points en spécifiant l'index dans le tableau précédent du point de départ et de celui d'arrivé.



La construction du mesh se fait en remplissant deux tableaux, le premier tableau où l'on ajoute pour chaque face la liste des arrêtes impliquées ordonnées dans le même sens pour chaque polygone.

Le second tableau spécifie la lecture du premier tableau en conservant pour chaque index le nombre d'arrête à lire dans le tableau précédent. Pour un point donné la somme des valeurs précédentes donne l'offset.

Index	0	1	2	3	4	5	...	$m-2$	$m-1$	m
Edgeld							...			

Face 0 Face 1 Face n

(a)

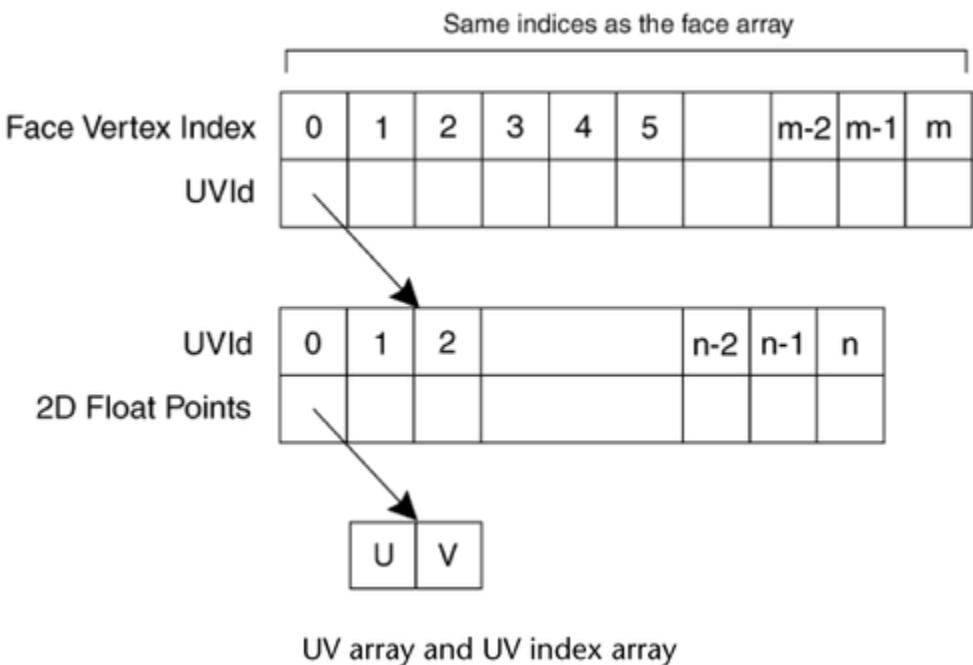
Faceld	0	1	2	3	4	5	...	$n-2$	$n-1$	n
Offset	0	3					...			

(b)

Face array and face index array structure

UV

La gestion des UVs revient à un aplatissement des points sur une texture. Pour chaque vertex ont spécifié les coordonnées sur la texture. C'est donc une association ID vertex (x,y,z) / Coordonnée 2D (x,y).



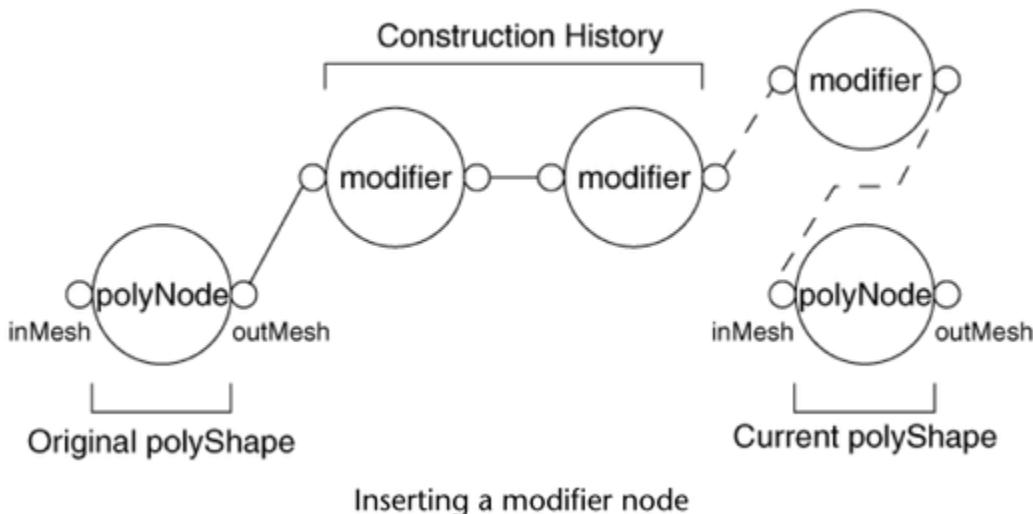
Modifier Et Opération

L'opération de création d'un mesh est en deux étapes, la première consiste à utiliser Un MFnMEsh objet pour créer un object contenant le type "**kMesh**".

MFnMesh permet ainsi de faire les opérations sur la géométrie du mesh, mais aussi sur les UVs.

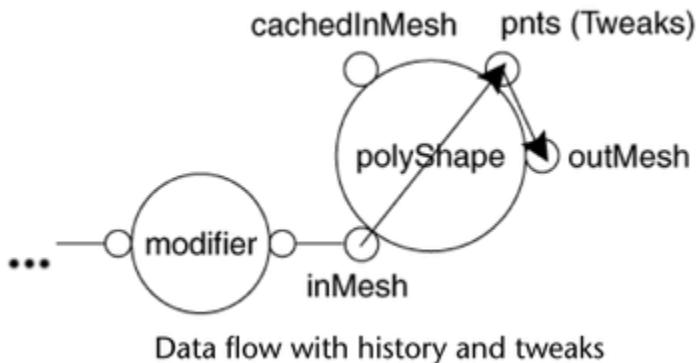
Par la suite pour que ses opérations soient effectives et enregistrées pour un objet auprès des différents systèmes le référençant, il faut utiliser un objet de type "**modifier**" comme le **MDAGModifier**.

Le "DAG Modifier" enregistre les opérations, puis lors de l'appel de la fonction "**doIt()**", les actions sont appliquées, l'historique des opérations sur l'objet est actualisé pour permettre le undo / redo. C'est à ce moment que les nodes deviennent apparente dans le "DAG système".



Inserting a modifier node

Il est important de comprendre que le "**Modifier**" retient la série d'opération et l'injecte dans l'objet, L'historique retenu sera l'opération et le changement qui en résulte sera la donnée conservée dans l'objet ainsi que la différence entre avant et après l'opération.



Le "plungin command"

Dans le cadre du projet nous avons une seul commande implémentée. Elle gère l'ouverture du menu du plugin. Il faut implémenté autant de commande que d'action en lien avec le plugin. La définition d'une opération de commande se fait dans l'appel du "**doIt()**". Il faut que la classe hérite de **MPxCommand** pour que Maya abonne à son système les nouvelles commandes créées.

Dans notre cas l'interface Qt s'occupe de gérer toutes les opérations.

Il faut prévoir une commande par opération.

Deux méthodes permettent de gérer l'abonnement des commandes au système, "**InitializePlugin()**" et "**UninitializePugin()**". Les commandes sont alors ajoutées dans maya lors

de l'initialisation du plugin et supprimées lors du déchargement du plugin. Ces commandes sont considérées comme l'exécution principale lors du chargement et du déchargement, ("Main").

L'intégration du UI

Maya offre l'intégration de **Qt** permettant de facilement définir une interface graphique avec le style de maya. Chaque version de maya dispose d'un version modifier de **Qt**. Maya 2016 utilise **qt 4.8.6** et maya 2017 / 2018 utilisent la version **5.6.1**.

Le "dev kit" contient toutes les références pour la compilation. Dans notre cas le UI est un singleton pour n'avoir qu'une seul instance de notre menu.

De ce fait on conserve le dernier PSD chargé même lorsque la fenêtre est fermée.

PLUGIN PHOTOSHOP ET ADOBE CSXS

Introduction

Créer un plugin pour photoshop demande de connaître deux systèmes différents.

Le premier concerne les actions de scripting de Photoshop, plus précisément les "**actions**" que l'on peut faire en javascript pour automatiser des opérations en faisant appel au code natif de photoshop.

Le second concerne le système de gestion des extensions qui prend en compte l'aspect UI, l'aspect gestion de l'installation et la désinstallation de l'extension, ainsi que les appels des actions précédemment cités.

Environnement de développement

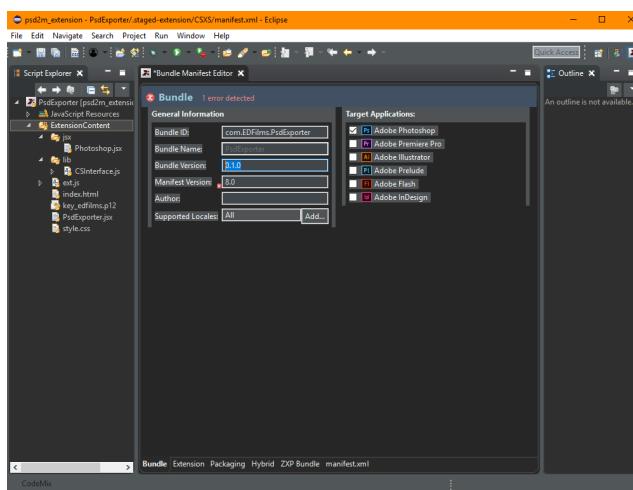
Il existe un "**Template**" de création de projet ici:

- <https://labs.adobe.com/downloads/extensionbuilder3.html>

Ce "template" offre une structure des fichiers ainsi que l'implémentation basique des scripts utiles à la définition d'une extension. Le "template" permet de créer des plugins pour toute la suite créative d'adobe.

Un des points importants est la gestion du "manifest". C'est un fichier XML permettant de paramétriser les informations de l'extension ainsi que les versions et les outils de la suite Créative qui peuvent l'utiliser.

- gestion des versions de photoshop
- gestion des versions du système d'extension supporté.

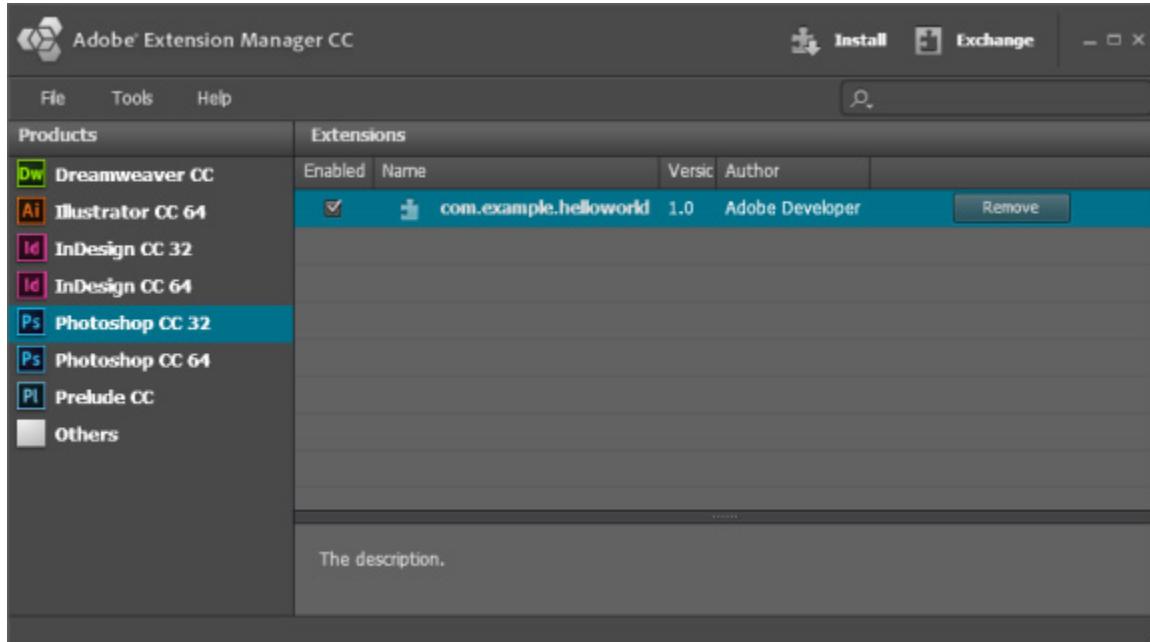


La structure du projet présenté dans l'illustration précédente se détail ainsi:

- "index.html" et "style.css" → Permettent de définir l'interface de l'extension.
- "ext.js" → Script opérant le lien entre le UI et les opérations exécutées (Actions).
- "jsx/Photoshop.jsx" → Contient la définition des opérations exécutées sur chaque layer.

Adobe CSXS

Adobe propose un module embarqué avec la suite créative. Ce module CSXS est en fait le gestionnaire des extensions gérées sur votre machine. L'utilitaire suivant permet de Visualiser les extensions présente pour chaque produit adobe.



ZXP permet d'installer les extensions, c'est ce que nous utilisons pour l'extension du projet. Il est embarqué dans installateur du plugin.

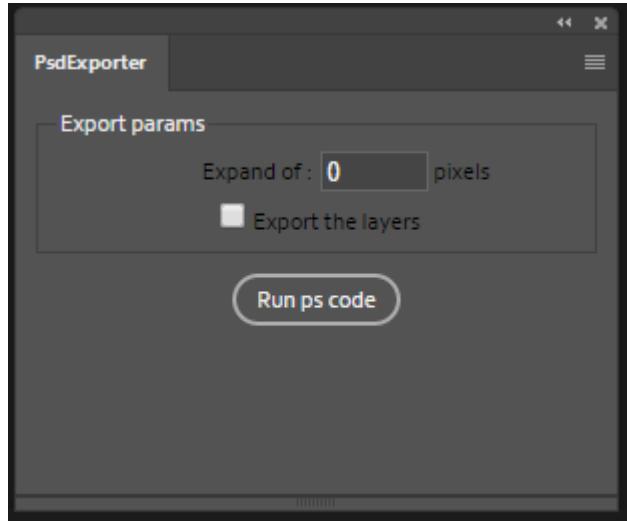
- https://helpx.adobe.com/ca_fr/animate/kb/install-animate-extensions.html

Procédure:

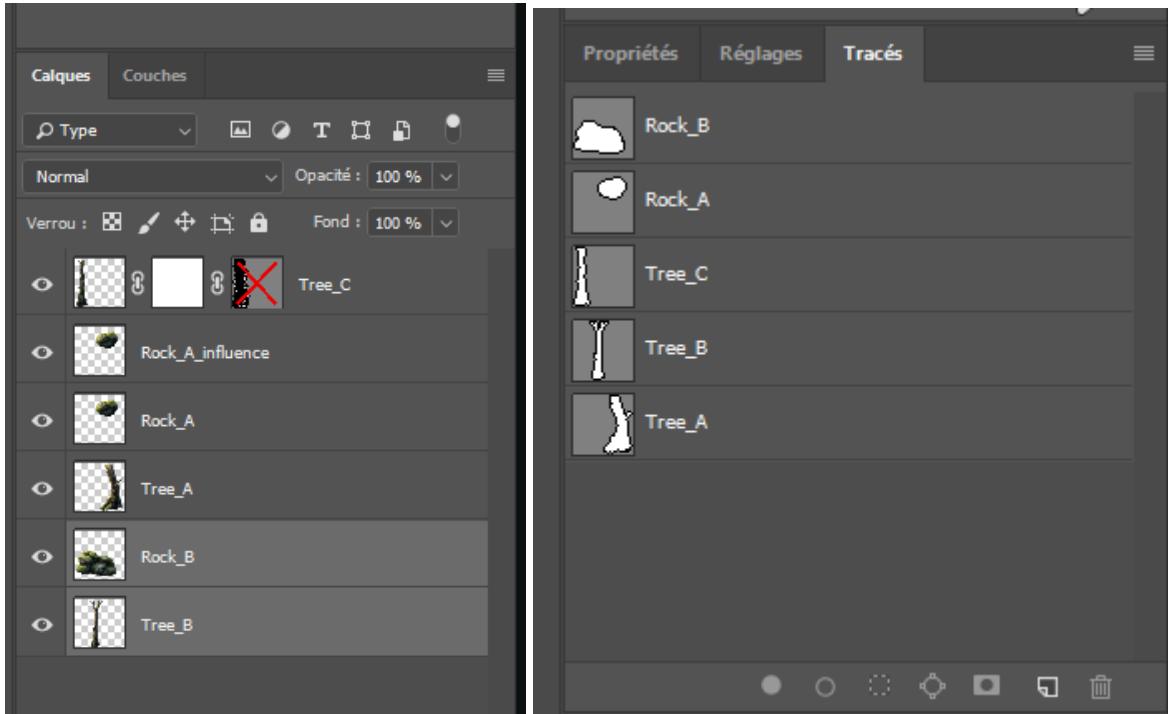
1. [Téléchargez](#) l'outil de ligne de commande ExManCmd pour Windows.
2. Extraire le fichier .zip à un emplacement sur votre bureau, tel que /Utilisateurs\[NomD'Utilisateur]\Bureau\
3. Copiez et collez l'extension (fichier .zxp) que vous souhaitez installer dans le même dossier (C:\Utilisateurs\[NomD'utilisateur]\Bureau\ExManCmd_win\") pour le retrouver plus facilement.
4. Ouvrez l'invite de commande en sélectionnant Démarrer > Tous les programmes > Accessoires > Invite de commandes.
5. Saisissez **cd** et renseignez le chemin d'accès pour accéder au dossier extrait :
 - a. cd C:\Utilisateurs\[NomD'Utilisateur]\Bureau\ExManCmd_win
6. Une fois dans le répertoire ExManCmd_win, saisissez la commande suivante dans l'invite de commandes et appuyez sur "Entrée" :
 - a. ExManCmd.exe /install "SampleCreateJSPlatform.zxp"
7. **Remarque :** si le nom de l'extension contient des espaces, placez-les entre guillemets ("").

Les opérations du Plugin

Voici la liste des opérations qui s'opèrent lorsque vous utilisez le plugin d'exportation du psd.



- Spécification d'un dossier de copie du PSD pour effectuer les opérations sur une copie, permet à l'utilisateur de garder son travail.
- Rasterisation des layers.
- Suppression des layers de "tracé généraux", sélection et dilatation des contours de chaque layer en fonction du paramètre spécifier dans l'interface "**expand of**".
- Génération d'un layer de tracé pour chaque layer selon la sélection réalisé dans l'étape précédente.
 - Nous utilisons le nom du layer pour conserver le lien entre layer et layer de tracé.



- Sauvegarde des modifications.
- L'exportation de PNG est disponible mais très lente. "**export the layers**"
 - En effet l'appel par la couche de scripting de photoshop ne permet pas d'utiliser directement son API, ce qui entraîne un ralentissement du traitement. Il est préférable de passer par maya pour cette étape.

Après étude des différentes possibilités pour ce plugin, l'implémentation d'une extension à partir de l'API C++ de adobe semblait très difficile et incertaine. Ayant peu de documentation sur le sujet, nous n'étions pas en mesure de garantir un résultat.

Adobe propose un ensemble de script ou d'action haut niveau permettant d'interagir sur les layers de la zone de travail.

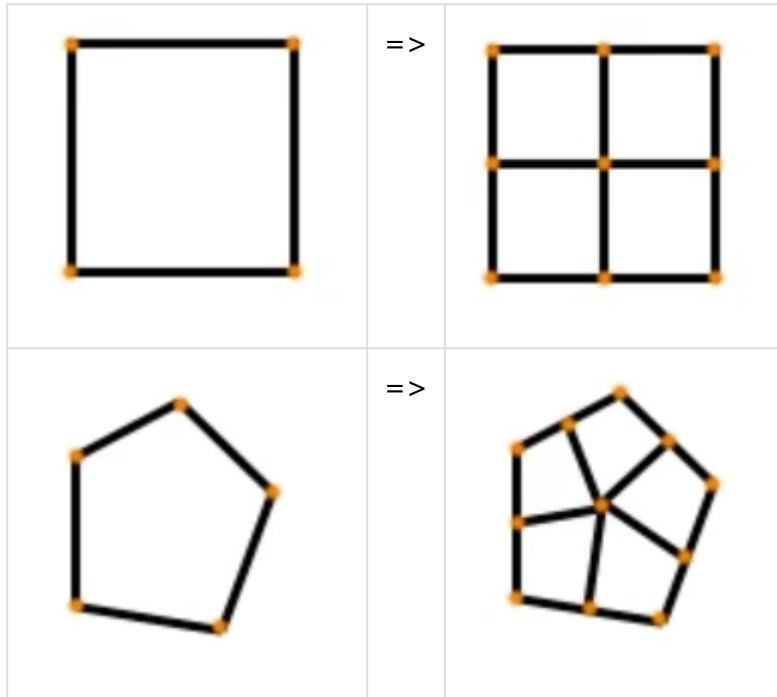
Nous avons donc combiné dans cette extension le scripting (actions) et la création d'un UI.

ANALYSE COMPLÉMENTAIRE - POLYGON SUBDIVISION

Introduction

L'une des étapes de [l'algorithme d'influence](#) est de diviser une face en utilisant le concept de [Subdivision Surface](#). Chaque division va rajouter un nombre de face égale au nombre de vertex de l'original et le nombre de vertex sera **doublé** et **incrémenté** de 1.

Les exemples suivants démontrent les résultats appliqués à des polygones simples.



Pour compléter la subdivision, il faudra résoudre deux problèmes :

1. Trouver le centre du polygone.
2. Diviser les arêtes du polygone.

1- Le centre

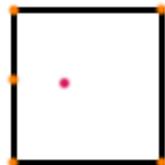
La première méthode qui a été considérée est de faire le moyen des vertex du polygone.

Avantage

- La moyenne de position est un calcul très rapide.
- Pour la majorité de nos cas, la moyenne donne un point intéressant et suffisant.

Désavantage

- Si le polygone est asymétrique il pourrait y avoir un débalancement sur le point du centre.

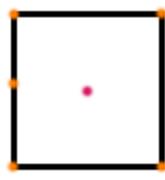


- Si le polygone est concave on pourrait avoir un centre qui est à l'extérieur de notre polygone.



Solution Aux Désavantages

Au lieu de chercher pour un centre, on pourrait chercher pour un centroid. Un centroid est un point qui se situe au centre de gravité du polygone (ou centre visuel).



Quelques solutions s'offrent à nous pour résoudre ce problème :

- <https://www.mathopenref.com/polygonirregular.html>
 - Cette référence utilise l'air du polygone.
- <https://blog.mapbox.com/a-new-algorithm-for-finding-a-visual-center-of-a-polygon-7c77e6492fbc>
 - Cette référence utilise une formule fractale (réursive).

Décision Final

Nous avons décidé de continuer de travailler avec la moyenne des vertex.

- Pour les besoins que nous avions sur le projet, la précision offerte par de meilleures solutions n'auraient pas nécessairement amener de valeur ajoutée.
- Les algorithmes pour trouver un centroid (tel que les liens plus haut) rajoutent une très grande quantité de calcul versus la moyenne des vertex, ce qui rajoutera un temps non négligeable lors du processus total.
- Bien que l'[Algorithm d'Influence](#) pourrait être appliquée sur d'autre mesh, dans le cadre de ce projet il est appliquée auprès des algorithmes de générations que nous avons fait nous même, où nous pouvons assurer une certaine "régularité" des polygones.

2- La division

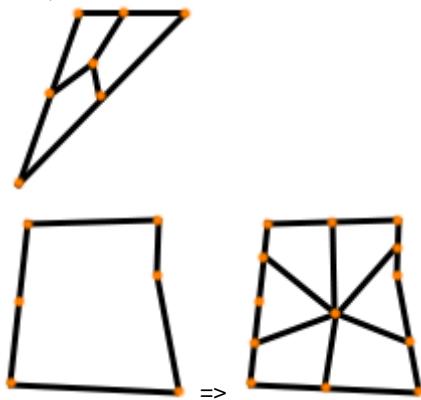
La première méthode qui a été considéré est de séparer chaque arrête en deux et de placer un nouveau vertex à ce point.

Avantage

- Calcul rapide.
- Cette technique fonctionne parfaitement pour des polygones réguliers.

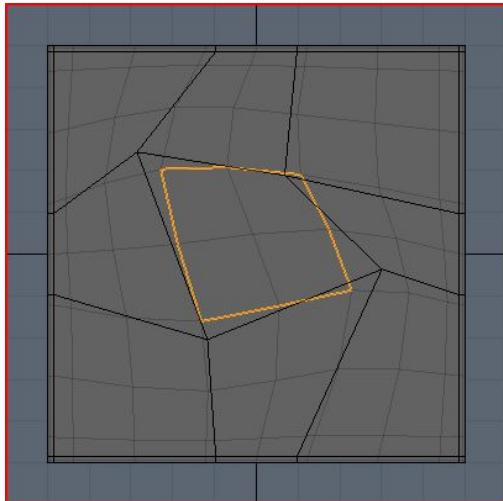
Désavantage

- Une face étroite ou déformée (par rapport à un carré) va souvent donner des faces étroites ou déformées après leur division, ex:



Solution À Ce Désavantage

L'algorithme de [Catmull-Clark](#) est une technique qui divise un mesh en face lisse. Surtout utilisé en 3D, cette solution pourrait être utilisée lors de la subdivision des faces pour lisser en même temps les faces.



Référence de l'image : <http://www.rorydriscoll.com/2008/08/01/catmull-clark-subdivision-the-basics/>.

C'est sur ce site que l'on peut retrouver une explication de l'algorithme, très utile dans le cas où on voudrait l'implémenter.

Décision Final

L'algorithme de Catmull-Clark a été aperçu lors de l'écriture de la documentation, lorsque le projet arrivait à sa fin. C'est en cherchant pour des liens explicatifs que nous sommes tombés sur cette explication de l'algorithme. La volonté d'intégrer cet algorithme au lieu de séparer les arrêtes en deux reste très forte. Il faudra en tenir compte en cas de poursuite du projet.

Par contre, l'algorithme qui est présentement implémenté respecte le mandat du projet.
L'implémentation de l'algorithme Catmull-Clark aurait seulement été une valeur ajoutée.

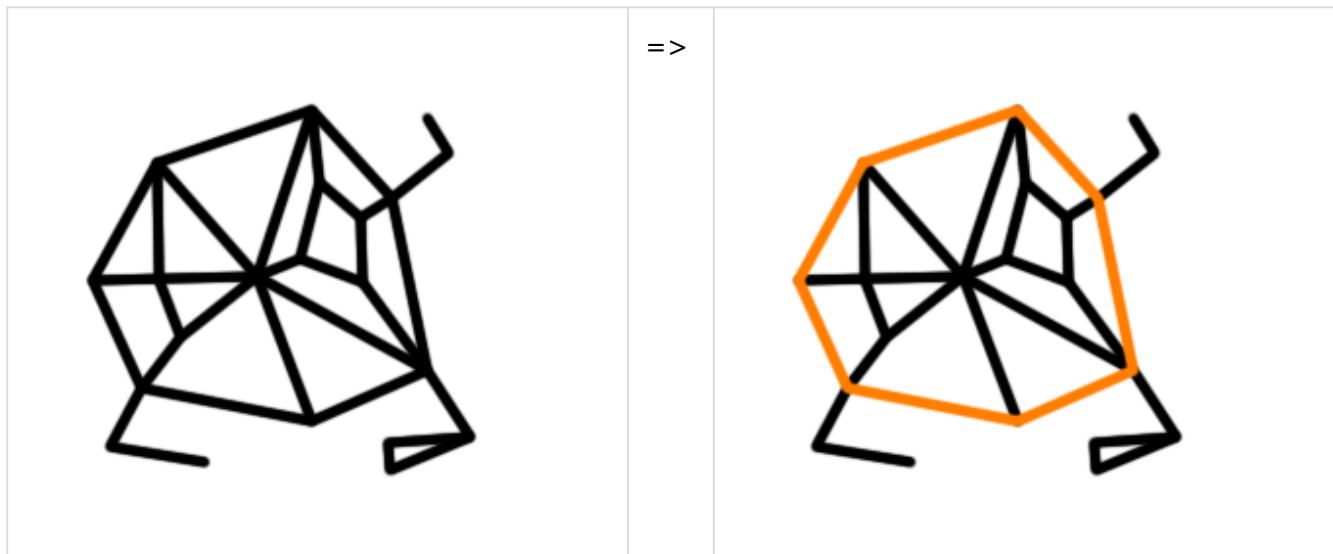
ANALYSE COMPLÉMENTAIRE - MINIMAL CYCLE BASIS

Introduction

Dans le [Curve Algorithm](#) on cherche à trouver des intersections entre un quantité N de **Path**, pour ensuite trouver la relation de voisin entre toutes ses intersections et enfin utiliser ces informations pour créer des faces d'un mesh. Dans cette page nous allons visiter les essais et les tests qui ont été réalisé pour résoudre la dernière partie de l'algorithme qui consiste à gérer la création des faces d'un mesh basée sur une relation de **Node**.

Les premières idées

Il est possible d'extraire le plus grand [Graph Planaire Extérieur](#) d'un autre graphe comme le montre l'image ci-dessous. Pour réussir ceci, il faudrait suivre les étapes suivante :

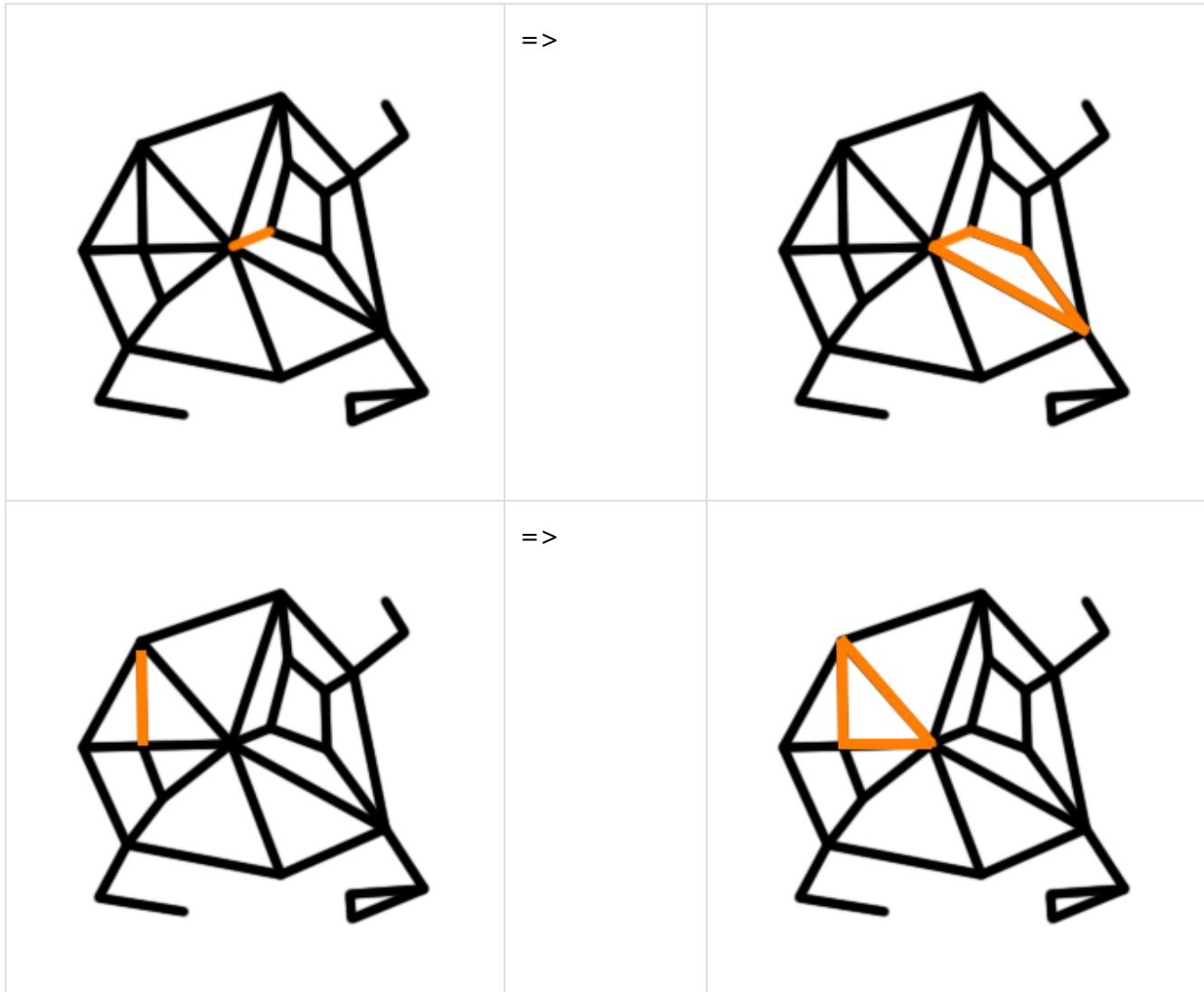


1. Choisir une **Node** qui est certain d'être à l'extérieur de tous cycles (ex. prendre le **Node** le plus à gauche)
2. Choisir le voisin de la **Node** qui est le plus à droit (ou le plus dans le sens des aiguilles d'une montre) en se basant sur un point plus loin que la **Node** (ex. si on a pris la **Node** la plus à gauche, on choisit un point un peu plus à gauche).
3. On continue de choisir les voisins les plus à droite jusqu'à ce qu'on revienne au point de départ.
 - a. Si on ne retrouve jamais le point de départ, il n'y a pas de cycle.
 - b. Si on se retrouve dans un cul de sac, on rebrousse chemin jusqu'à la **Node** précédente qui a encore des voisins que nous n'avons pas visité.
 - c. Si on se retrouve avec une liste de X dernières **Node** équivalente à l'inverse des X premières **Node**, ça veut dire que nous avons commencé sur un cul de sac et que nous pouvons la retirer.

L'expérience d'un précédent projet fut une bonne piste pour commencer à résoudre notre problème.

Il devrait être possible de modifier cet algorithme pour chercher pour des **Cycles Minimales** au lieu d'un **Graph Planaire Extérieur**. Un exemple rapide démontre que si on prend deux **Nodes**,

une Initiale et un Voisine, les deux "extérieures au graphe" on se retrouve toujours avec un **Cycle Minimal**.



Voici les problématiques avec cette idée au point où nous en sommes maintenant :

1. Si on choisit "mal" nos deux **Nodes**, on peut se retrouver à faire le tour du graphe en créant sans le vouloir un **Graphe Planaire Extérieur** ou quelque chose de similaire mais non désiré.
2. Il est très possible de trouver plus qu'une fois chaque **Cycle Minimal**.

La problématique numéro **2** pourrait se régler avec un **Graphe Orienté**, duquel on enlèverait des branches à chaque fois qu'un **Cycle Minimal** est trouvé.

Par contre, après discussion avec mes collègues, nous n'avons pas trouvé une solution au premier problème.

A*

C'est à ce moment qu'un collègue offre l'idée d'utiliser l'algorithme de A Star. L'algorithme de A Star permet de chercher le chemin le plus petit entre deux points basé sur une heuristique (qui est souvent la distance entre les deux points). Alors on pourrait utiliser cet algorithme pour chercher le chemin le plus court entre deux **Nodes** voisines en considérant que ces **Nodes** n'ont plus d'arrêté entre les deux.



Le point rouge est le point de départ de la recherche et le point de l'autre bord de l'arrête orange est le point d'arrivée.

On enlève l'arrête entre les deux **Nodes**.



Avec A Star on trouve le chemin le plus court entre ces points.

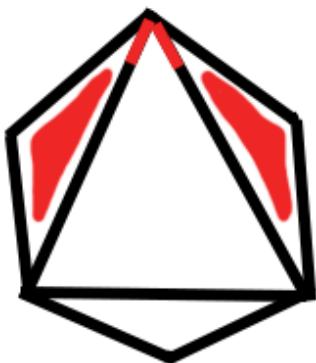
Une fois un chemin trouvé on peut compléter la face.

Vu qu'on ne veut pas trouver plus qu'une fois la même face, on enlève l'arrête entre nos deux **Nodes**, mais seulement dans une seul direction.

De plus, on applique cet algorithme pour chaque voisins de chaque **Node**. Vu qu'on enlève l'arrête Voisin → **Node** à chaque itération, on se retrouve des fois avec des **Nodes** qui n'ont plus de voisin et dans ce cas on sait qu'on a complété les faces autour de cette **Node** et que l'on peut passer au prochain. Le choix de **Nodes** et de voisins se fait semi-aléatoirement, l'ordre n'est pas établi alors on considère que l'on ne sais pas laquel sera sélectionnée.

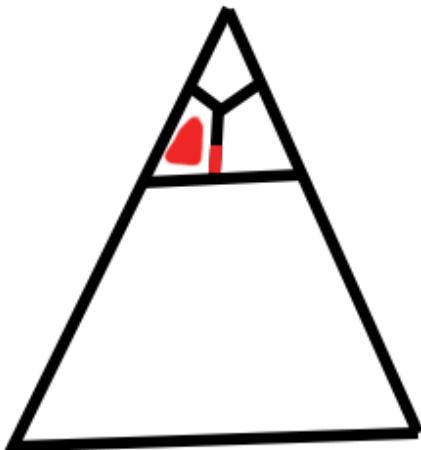
Voici les problématiques avec cette idée au point où nous en sommes maintenant :

1. On peut bloquer l'accès à une face à cause de notre choix de **Node** / Voisin.



- a. (Les lignes rouges représentent qu'une arrête noire à rouge est bloqué et les zones rouges représentent le fait que ces faces aient été trouvé).
- b. Dans le cas du point a), on peut voir qu'on ne peut plus compléter la face du centre à cause que la face a deux arrêtes qui sont bloquées dans des sens différents.

2. On risque encore de trouver des faces qu'on a trouvé précédemment avec des cas de bord.



- a. (La ligne rouge représente que l'arrête Noir à Rouge est bloqué, et la zone rouge représente que cette face a été trouvé)
- b. Dans ce cas, si on était pour commencer à chercher sur l'arrête en bas de la face rouge, on ne serait pas bloqué par l'arrête rouge parce qu'il est bloqué dans un sens et non dans l'autre. Donc on compléterait une deuxième fois cette face.

Pour régler le premier point on pourrait forcer la sélection des arrêtes pour qu'elles soient toujours cyclique. Donc au lieu d'enlever l'arrête **Node** → Voisin, on enlèverait plutôt l'arrête qui serait dans le sens horaire de la face. On pourrait la trouver grâce à des calculs de direction de polygone. (https://en.wikipedia.org/wiki/Curve_orientation).

Pour régler le deuxième point on pourrait enlever le dernier chemin de la face en sens inverse. Donc si on avait la face A → B → C, on enlèverait le chemin A → B et C → B. Par contre, cette idée viendrait s'opposer au premier point et on se retrouverait avec des cas de bord de face inaccessible.

Il faut donc chercher un peu plus pour trouver ce qui manque à cet algorithme.

Minimal Cycle Basis

L'utilisation de A star ne fut pas aussi simple que prévu, heureusement mon collègue c'est souvenu de l'algorithme de **Minimal Cycle Basis**,

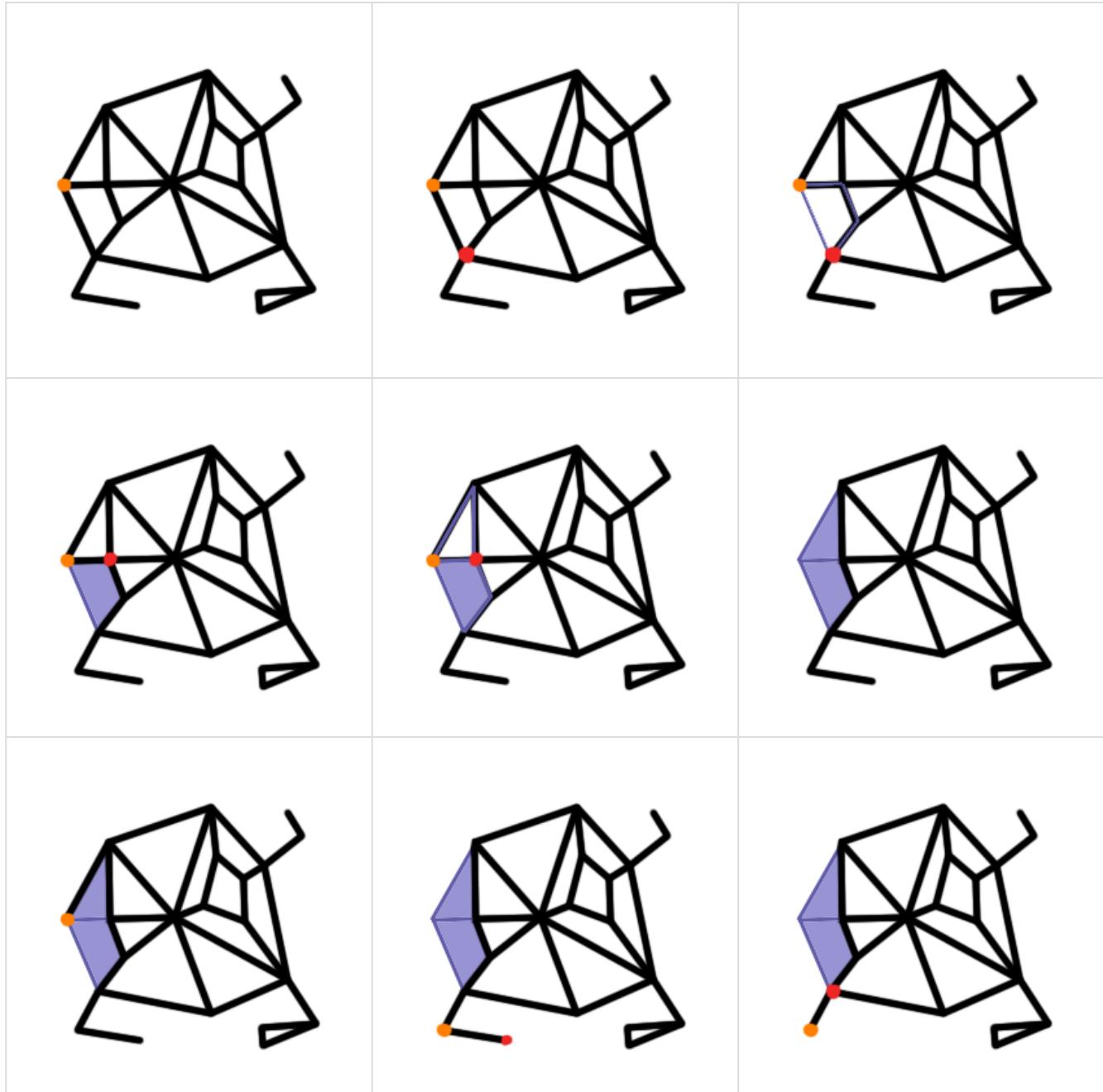
Un **Minimal Cycle Basis** répond exactement à notre besoin de trouver les faces dans un **Graphe Orienté**. Ce terme décrit l'ensemble des **Cycles Minimale** dans un graphe et il a déjà été résolu.

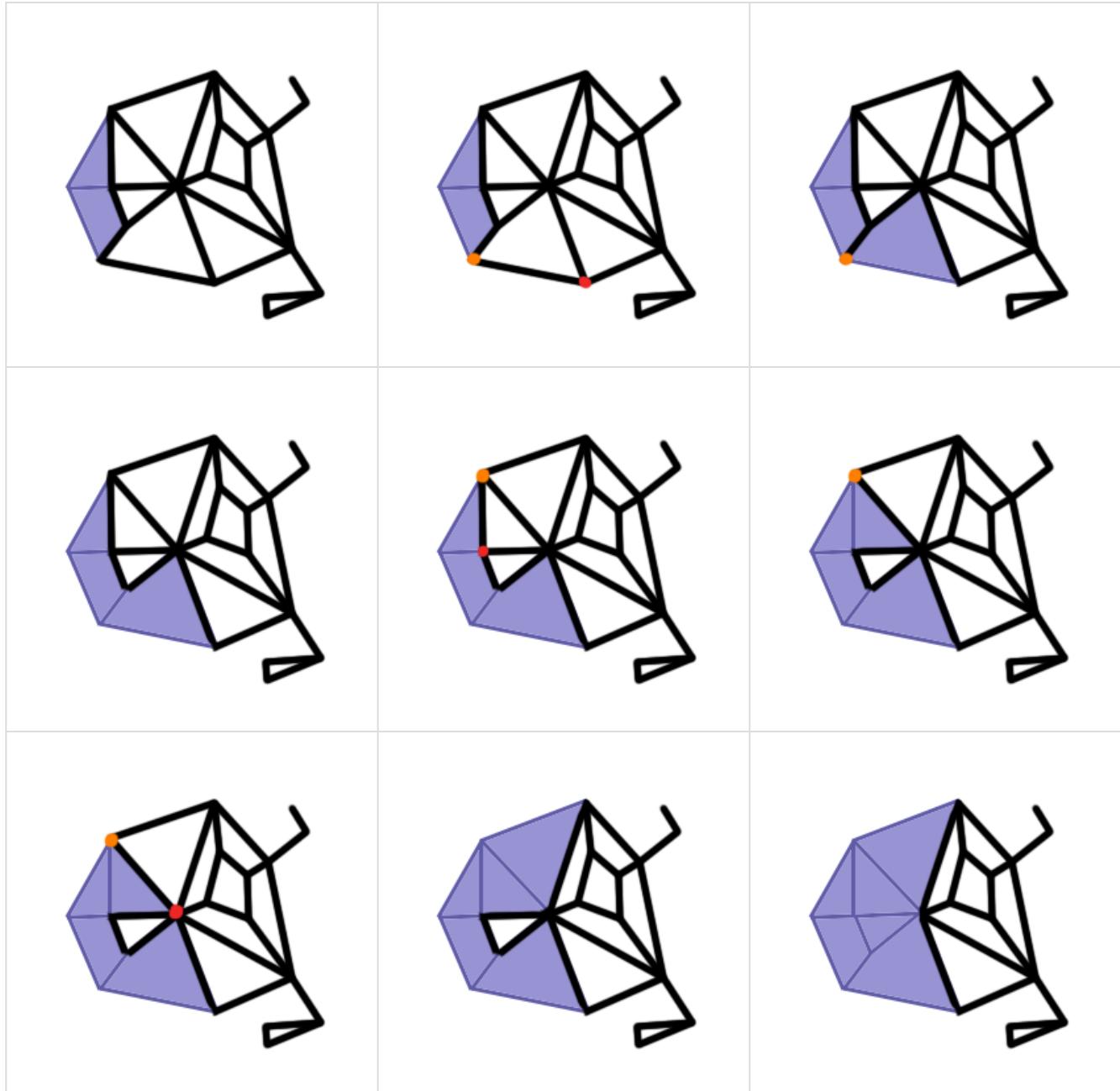
En lisant [cet article](#), je me rend compte de l'erreur que j'ai fait au tout début. L'idée de choisir un voisin le plus à droite pour compléter une face était la bonne piste, mais il manquait une seul étape, celle de tourner à gauche après avoir choisi un voisin de droite.

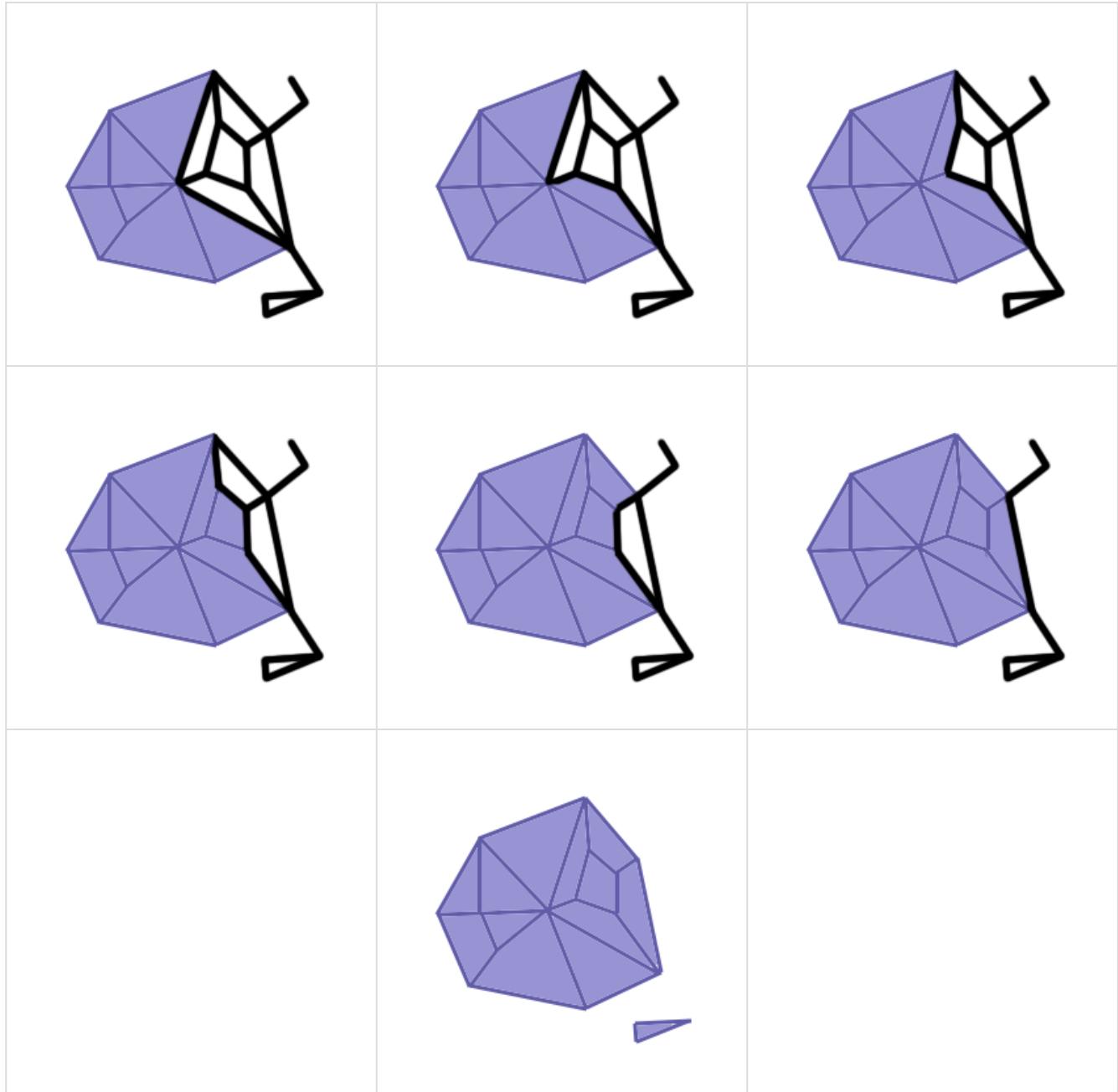
1. Choisir une **Node** qui est certaine d'être à l'extérieur de tous cycles (ex. prendre la **Node** la plus à gauche)
2. Choisir le **Voisin** de la **Node** qui est la plus à droit (ou la plus dans le sens des aiguilles d'une montre) en se basant sur un point plus loin que la **Node** (ex. si on a pris la **Node** la plus à gauche, on choisit un point un peu plus à gauche).
3. Choisir un voisin du **Voisin** qui est le plus à gauche et continuer ainsi jusqu'à ce qu'on ait retrouvé le **Node** du départ.
 - a. Si on ne trouve pas le **Node** de départ, ça veut dire qu'on est dans un cul de sac et que l'on peut abandonner ce voisin.
 - b. Il est encore possible de se trouver avec une face qui ne fonctionne pas (comme avoir une **Node** plus qu'une fois dans la liste) à cause des cas de bord. Dans ces cas, on abandonne la face et le voisin.
4. Après avoir trouvé une face on enlève complètement l'arrête entre la **Node** et le **Voisin**.
5. On continue au point 2. jusqu'à ce qu'il ne reste qu'un seul voisin à la Node originale. Lorsqu'on arrive à ce point on passe au prochain **Node** le plus à gauche.

Les images suivantes expliquent visuellement l'algorithme,

- Les points **orange** sont les **Nodes**,
- Les points **rouge** sont les **Voisins**.
- Les faces **mauvaises** sont les faces nouvellement rajoutées.







C'est cet algorithme qui est présentement implémenté dans le projet. Il est visiblement très rapide, surtout comparativement à [la section du Curve Algorithm qui s'occupe de trouver les intersections entre les courbes bézier](#). L'article sur lequel je me suis basé n'aborde pas la [comparaison asymptotique](#) de l'algorithme, donc pour la cause j'estime que cet algorithme est en $O(2N)$ ce qui veut dire que le temps pris par l'algorithme va augmenter d'une façon linéaire avec un plus grand nombre d'intersection.