Rapport de projet

GAUTHIER Silvère
LAMEIRA Yannick
PELADAN Cécile

12 février 2015

Table des matières

Chapitre 1

Remerciements

Un grand merci à Frédéric Boudon et Benjamin Gilles pour leur encadrement. Un remerciement particulier à Frédéric Boudon pour son idée d'outil de subdivision locale.

Chapitre 2

Introduction

2.1 Sujet initial

L'objectif de ce projet est de créer un logiciel de sculpture 3D. L'utilisateur aurait à disposition un maillage déformable, qu'il pourrait modeler avec différents outils tels que déplacement, ajout de matière ou lissage. L'interface devra permettre à l'utilisateur de facilement créer un maillage initial, le visualiser et interagir avec lui à l'aide des différents outils de modelage. Initialement, les premières fonctionnalités à implémenter seront la création d'une sphère ou d'un cube comme maillage de départ, la navigation dans l'espace 3D pour se positionner autour du maillage, puis les outils de déformation cités ci-dessus. Une interface graphique contenant les boutons d'outils sera définie pour que l'utilisateur puisse intuitivement appliquer les différentes opérations proposées.

Les principales difficultés ici seront d'abord de gérer correctement l'interaction 3D de l'utilisateur avec le maillage via le curseur et la fenêtre 2D. Ensuite viendra la mise en place d'une structure de données efficace et robuste de maillage avec l'implémentation des algorithmes de subdivision et de raffinement.

Si le temps le permet, d'autres outils et maillages de base pourront être implémentés, afin d'enrichir le logiciel. On pourra également réfléchir à une manière d'importer et exporter les maillages sous différents formats tels que le OBJ ou le STL par exemple.

Ce logiciel sera développé en C++ avec la bibliothèque OpenGL pour le rendu 3D et la bibliothèque Qt pour définir l'interface.

Chapitre 3

Cahier des charges

Ce chapitre détaille la phase de conception du projet.

3.1 Mécanismes

Cette section détaille succintement les fonctionnalités de l'application.

Les Modèles

Différents maillages prédéfinis, appelés ici modèles, seront mis à disposition de l'utilisateur.
Nous en prévoyons actuellement cinq, paramétrables par l'utilisateur :

- **Cube :** défini par une largeur, une hauteur et une profondeur.
- **Sphère :** défini par un rayon.
- **Cylindre :** défini par une hauteur et un rayon.
- **Cône :** défini par une hauteur et un rayon.
- **Tore :** défini par un rayon horizontal et un rayon vertical.

Chaque modèle sera aussi paramétré par un pas de discrétisation.

Le Rendu

Le rendu s'effectuera dans une classe spécifique, et permet de synchroniser la structure interne des maillages avec les buffer objects. Il existera également différentes options de rendu telles que l'affichage solide ou en fils de fer, les couleurs, les lumières...etc.

Les Outils de sculpture

Différents outils seront disponibles. Ils auront pour action de modifier le maillage dans la direction des normales de surface, et selon un schéma de modification propre à chaque outil. Nous avons défini deux catégories d'outils : modifications globales et locales des objets (une troisième est destinée au déplacement de la caméra dans la scène en trois dimensions).

— **Modifications Globales :**
  — Déplacement d'un objet dans le repère scène.
  — Rotation d'un objet dans le repère scène.
  — Mise à l'échelle d'un objet dans le repère scène.
— **Modifications Locales :**
  — Ajout de matière à la surface de l'objet.
  — Lissage de la surface de l'objet.
  — Déplacement d'une partie des points de la surface de l'objet.
  — Gonflement de la surface de l'objet.
  — Pincement de la surface de l'objet.

Les Algorithmes de maillage

Les différents algorithmes de modification de maillages seront regroupés dans une classe statique spécifique. Y seront présentes les fonctions de subdivision et de décimation globale, ainsi qu'une subdivision et une décimation automatique paramétrable avec une longueur (maximale ou minimale) d'arête.

Interface

L'interface de l'application devra être ergonomique et proposer de nombreux raccourcis clavier afin de permettre à l'utilisateur de travailler rapidement. Elle devra également être le plus intuitive possible, car un tel logiciel pourrait devenir très compliqué à appréhender. Cela concernera tant les dispositions que les icônes qui devront suivre une certaine logique.

Les Options

nouveau, ouvrir, enregistrer, importer, exporter, annuler, refaire, paramètres, quitter

3.2 Structure du programme

Cette partie détaille succintement la structure globale du logiciel, ce qui couvre les aspects non maîtrisables par l'utilisateur.

L'API

L'application devra contenir une surcouche, ayant pour objectif de rendre notre logiciel facilement extensible par d'autres développeurs, sans qu'ils aient besoin de connaître tous les détails de la structure interne de celui-ci.

La structure interne des maillages retenue ici est la structure par demi-arêtes. Nous auront donc, pour chaque maillage, des listes de points, faces et demi-arêtes.
Chaque point contiendra des coordonnées en trois dimensions, un indice de position dans la liste afin d'améliorer la vitesse d'accès, ainsi qu'un pointeur vers une demi-arête sortante.
Chaque face contiendra un pointeur vers une demi-arête intérieure, et éventuellement

un vecteur normal.
Chaque demi-arête contiendra des pointeurs vers : une face, un point, les demi-arêtes suivantes et précédentes au sein de la même face, et la demi-arête opposé.

Plusieurs méthodes publiques devront être présentes, telles que "ajouter une face", "couper une arête en deux", "fusionner deux arêtes"... et cacher la structure interne (n'agir qu'à l'aide de coordonnées par exemple). Cela permettra dans le futur, si besoin, de changer la structure interne sans modifier le reste de l'application.

Chapitre 4

Gestion du projet

Cette partie traite globalement de tout ce qui concerne l'organisation du projet, que ce soit au niveau de la conception, du développement, de l'équipe ou encore de la gestion des fichiers.

4.1 Gestion de l'équipe

Tous les membres se connaissant et étant supposés être capable de travailler en équipe, nous n'avons fait aucune élection préalable de chef de projet.
Mais le déroulement du projet nous a imposé ce choix, GAUTHIER Silvère assurera donc ce rôle, afin de garder une cohésion de groupe et assurer la réalisation de nos objectifs.
Chaque membre peut tout de même participer activement au projet, autant lors de la conception que du développement. Toutes les décisions seront prises suivant la majorité lors de votes.

Pour ce qui est des réunions de projets, nous avons convenu avec nos tuteurs d'une réunion, allant d'environ trente minutes à une heure, toutes les semaines, afin de mettre au point l'avancement du projet. En parallèle, tous les membres de notre équipe se retrouvent une fois par semaine afin de discuter des points clés effectués ou à venir, donner lieu aux votes pour les prises de décisions, ou encore, lors de la phase de développement, travailler en collaboration afin d'optimiser notre travail.

Au niveau du travail collaboratif, nous avons mis en place un dépôt sur github (adresse à la page 15), contenant tant la documentation que les sources de notre programme. Par ailleurs, nous mettrons sur ce dépôt uniquement les fichiers sources et les images, mais en aucun cas les fichiers temporaires ou les exécutables. Les seuls fichiers binaires disponibles seront les PDF de la documentation, pour un soucis de facilité d'accès et de lecture.

4.2 Découpage en tâches

Afin de préparer le développement du programme, il était nécessaire de séparer les fonctionnalités les unes des autres. Nous avons abouti à ce dia-

gramme, qui résume notre choix de découpage :

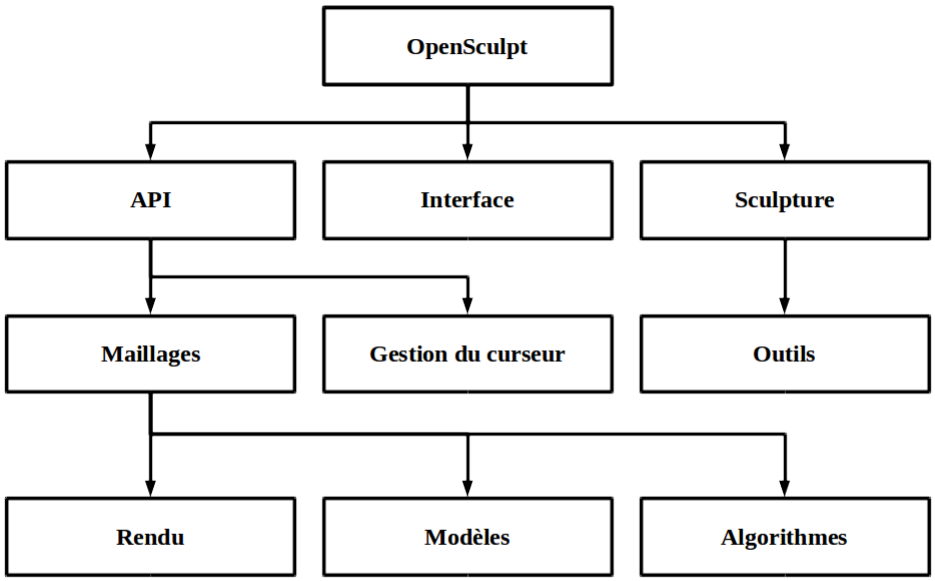

FIGURE 4.1: Diagramme des tâches du projet

4.3 Assignation

Le projet étant découpé en un certain nombre de modules, il ne restait plus qu'à assigner chaque tâche à un ou plusieurs membres de l'équipe. Nous nous sommes organisés comme ceci :

- **Conception, implémentation et vérifications de l'API** : GAUTHIER Silvère.
- **Mise en place de l'interface** : LAMEIRA Yannick.
- **Implémentation des maillages modèles** : PELADAN Cécile.
- **Conception et implémentation des outils** : GAUTHIER Silvère, LAMEIRA Yannick, PELADAN Cécile.
- **Tests et vérifications de l'application** : GAUTHIER Silvère, LAMEIRA Yannick, PELADAN Cécile.

Bien entendu, les membres pourront évidemment faire appel aux autres pour trouver une solution à un problème par exemple.

Le détail complet des tâches et assignations se situe dans la section Gestion du temps, page 13.

4.4 Gestion du temps

Afin de clarifier notre gestion du temps, deux diagrammes de Gantt (prédictif et final) sont disponibles en annexe (cf page 23) et dans la documentation de notre projet.

4.5 Choix technologiques

Afin de pouvoir développer correctement notre logiciel, il a fallu définir tout ce que nous allions utiliser en terme de langages et bibliothèques selon notre logique de conception.

Langages de programmation

Pour des besoins de performances, nous avons comparé différents langages. Pour réduire le temps de recherche et de comparaison, nous nous sommes appuyé sur des tests déjà effectués par d'autre.
Des tests de performances concernant un large panel de langages, comparés dans quatre contextes différents, sont fournis en annexe, page 24.
Nous pouvons observer que globalement, le langage le plus rapide est ici C++. L'utilisation de ce langage étant très fréquente dans les applications en temps réel, de part sa réputation d'un des langages les plus performants, et tous les membres de notre équipe sachant l'utiliser, nous avons fait le choix de programmer le logiciel en C++.

Bibliothèques

Pour la gestion graphique de l'interface et de l'affichage de l'objet, nous avons cherché une bibliothèque relativement simple d'utilisation mais surtout performante afin de garder la fluidité gagnée avec le choix des langages de programmation.
Connaissant la bibliothèque OpenGL, qui est bas niveau et performante dans les affichages deux et trois dimensions, nous nous sommes tournés vers une bibliothèque utilisant OpenGL : Qt.

Représentation du maillage

Nous cherchons ici à comparer différentes techniques permettant de représenter un maillage, afin de choisir celle qui sera la plus adaptée à nos opérations. Voici un tableau récapitulatif de cette étude comparative :

Représentation	Avantages	Inconvénients
Octree ou KDTree	- Hiérarchie des résolutions - Rendu volumique possible - Construction et parcours simples	- Visualisation surfacique difficile - Coût de stockage excessif - Recalculer à chaque modification
Arbre CSG	- Historique de construction - Approche fonctionnelle	- Non unicité - Opérations complexes - Domaine insuffisant
G-maps	- Opérations de topologie simples - Plongements multiples	- Séparation topologie / plongement
Liste de triangles	- Opérations simples	- Stockage non optimisé
Sommets partagés	- Opérations simples - Stockage correct	
Bandes de triangles	- Stockage correct	- Chaque sommet est visité deux fois - Opérations de déplacement délicates
Structure par faces	- Chaque face pointe sur ses sommets - Une face connaît les faces adjacentes	- Pas d'accès direct aux arêtes
Structure par demi-arêtes	- Parcours de maillage très pratiques	- Coût de stockage excessif
Vertex Array (VAO)	- Optimisé pour le rendu OpenGL - Simple d'utilisation	- Utilise le CPU et la RAM
Vertex Buffer Object (VBO)	- Optimisé pour le rendu OpenGL - Utilise le GPU et la VRAM	

TABLE 4.1: Tableau comparatif de méthodes de représentation de maillage 3D

L'étude comparative montre qu'il serait judicieux d'utiliser conjointement des Vertex Buffer Objects (VBOs) afin d'optimiser les performances d'affichage et une classe personnalisée pour aisément gérer nos données par une surcouche de méthodes. La structure interne des maillages sera donc une structure de type demi-arêtes, afin de faciliter les opérations de voisinage.

4.6 Gestion des fichiers

Nous avons beaucoup de fichiers à gérer dans ce projet, et nous devions établir des conventions ou des moyens afin de les gérer correctement.

Format des Fichiers

Le code étant écrit en C++, nous utiliserons des fichiers d'en-tête au format H et des fichiers de définition au format CPP.
Toutes les images nécessaires au logiciel seront au format PNG afin de pouvoir utiliser la transparence et garder la pleine qualité d'image (contrairement à JPEG qui perd de l'information à la compression).

Sauvegarde

Une façon judicieuse de sauvegarder nos données serait d'avoir un format spécial pour enregistrer un projet complet contenant une liste d'objets, ainsi qu'une possibilité d'exportation de chaque objet ou scène dans un format connu tel que STL ou OBJ.

Commentaires

Si une méthode ou fonction (voir même un bloc) dépasse une certaine taille ou devient trop compliquée, un commentaire sera ajouté avant celle-ci expliquant brièvement son processus :

```
/** Description :
*** Entrée : ...
*** Sortie : ...
**/
```

Quelque commentaires précieux pour le travail collaboratif seront également présents :

Marqueur spécifique	Signification
TODO	A mettre à la place du code d'une fonctionnalité à implémenter
RECODE	A mettre au dessus du bloc d'une fonctionnalité à refaire ou à optimiser
FIXME	A mettre au dessus du bloc d'une fonctionnalité contenant un bug

Table 4.2: Forme et usage des commentaires

Conventions de Nommage

Pour la lisibilité et la bonne pratique du développement de l'application, il est nécessaire de suivre des règles établies au sein de l'équipe de projet, appelées conventions. Ainsi, nous avons choisi d'écrire le code en anglais uniquement, mis à part pour les commentaires utiles aux développeurs préférant le français. De même, au moins une ligne de commentaire est requise avant chaque déclaration de classe ou de fonction, afin d'en expliquer brièvement son fonctionnement (sauf dans le cas de méthodes simples avec des noms explicites).
Voici un tableau récapitulatif des conventions de nommages pour les noms de variables, classes et méthodes :

Type de variable	Format du nom
Classe	Majuscule suivit de minuscules
Méthode et Fonction	Minuscules (pour les mots composés, chaque mots suivant est une majuscule suivit de minuscules)
Attribut de classe	Précédé par m_
Variable globale	Précédé par g_
Variable statique	Précédé par s_

Table 4.3: Conventions de nommage

Gestion du code source

Afin de faciliter le travail collaboratif, nous utilisons un dépôt utilisant le gestionnaire de version GIT, hébergé sur le site :

`https://github.com/slvrgauthier/OpenSculpt`

Sur ce dépôt seront présents tous les fichiers sources nécessaires au développement du programme ainsi que les documentations au format LaTeX, ODT et PDF (même si aucun fichier binaire ne devrait être présent, il est plus pratique de récupérer directement un tel fichier que de le compiler soit-même). De plus, y seront stockées toutes les données utilisées par le programme telles que les images et autres ressources. Seuls les fichiers temporaires, exécutables et fichiers de sauvegarde ne seront pas stockés.

Chapitre 5

Développement

Ce chapitre est une sorte de carnet de bord. Il détaille tout ce qui concerne le développement de l'application.

5.1 API

5.2 ...

...

Chapitre 6

Manuel

Ce chapitre détaille tout ce qui concerne l'utilisation de l'application, autant pour les utilisateurs que les développeurs souhaitant étendre les possibilités du logiciel.

6.1 Utilisateur

6.2 Développeur

Toute personne avec des bases de programmation pourra étendre les fonctionnalités de celle-ci grâce à notre API.

Ajout d'un maillage modèle

Il est possible d'ajouter un maillage modèle afin d'enrichir les objets de base proposés à l'utilisateur.

Ajout d'un outil

Il est possible d'ajouter un maillage modèle afin d'enrichir les actions de sculpture proposéés à l'utilisateur.

Pour aller plus loin...

Si un développeur souhaite modifier plus profondément l'application, il faudra pour cela se référer à la partie **Développement** (cf page 17), notamment les explications détaillant l'**API** (cf page 17).

Chapitre 7

Post-Mortem

Cette section liste toutes les étapes de la conception que nous n'avons pas réalisées (selon notre ordre de priorités), ainsi que toutes les améliorations auxquelles nous avons pensé lors de la phase de développement.

7.1 Fonctionnalités non implémentées

7.2 Améliorations réalisables

Annexe A

Dans cette partie seront placés les éléments trop volumineux pour être inclus directement dans le texte, tels que les images ou graphiques.

A.1 Diagramme de Gantt

Les diagrammes sont fournis page 23.

A.2 Comparatif de performances

Le graphique est situé page 24.

FIGURE A.1: Diagrammes de Gantt prédictif et final

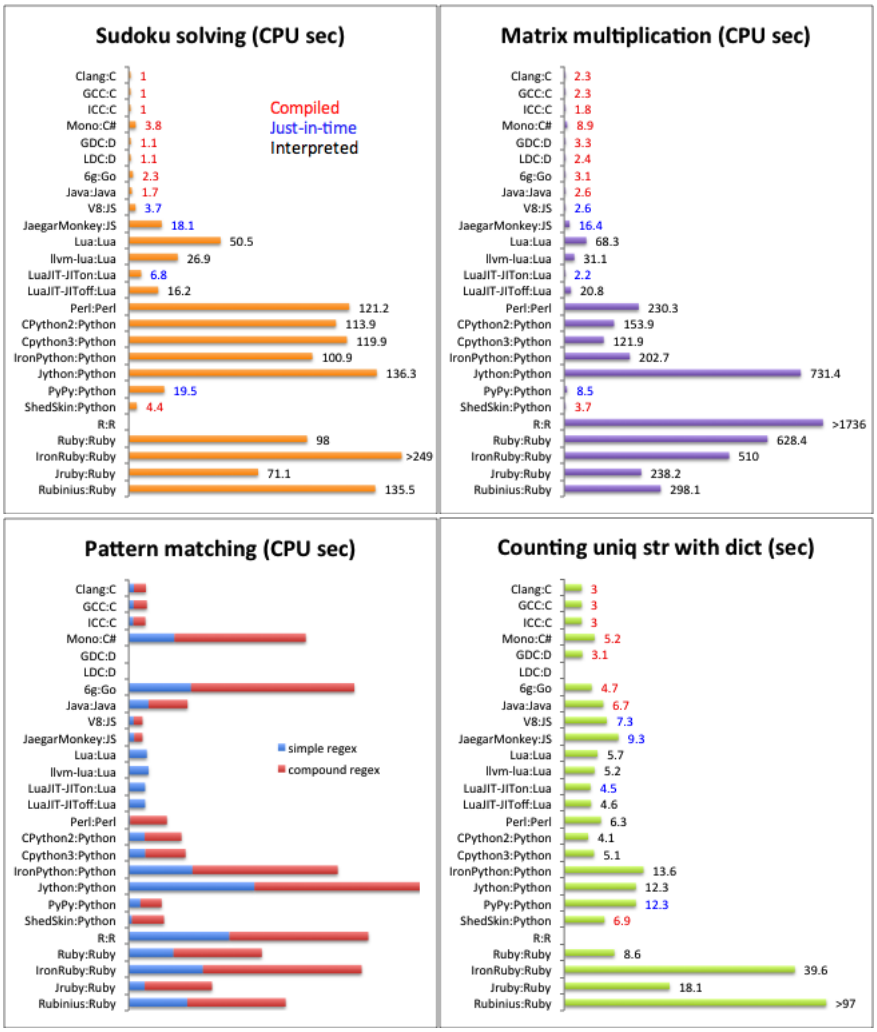

FIGURE A.2: Comparaisons de performances de divers langages dans des cas donnés

Source : `http://attractivechaos.wordpress.com/2011/06/22/my-programming-language-benchmark-analyses/`