Université de Poitiers

Master d'informatique

# Projet Discret
# Rapport Pédagogique

*Auteurs :*
Thomas Benoist,
Zied Ben Othmane,
Adrien Bisutti,
Lydie Richaume

*Superviseur:*
Philippe Meseure

March 17, 2016


# Contents

# 1 Introduction

## 1.1 Contexte

Dans le cadre de notre formation de deuxième année de master d'informatique nous avons eu à réaliser un projet à destination de clients réels. Ce projet s'est déroulé sur six mois : depuis Octobre jusqu'en Mars. Il permet de mettre en pratique les compétences que nous avons acquises durant nos cours de gestion de projet, ainsi que de nous confronter au développement d'un projet de dimension moyenne. Il nous a donc été attribué des clients pour qui nous avons dû réaliser un projet.

Les clients qui nous ont été attribués sont Éric Andres et Gaëlle Largeteau-Skapin, tous deux faisant partie de l'équipe de géométrie discrète au laboratoire X-Lim. Ces derniers ont développé un algorithme de génération de surfaces de révolution discrètes. Ils souhaitaient pouvoir mettre à disposition un outil qui permettrait d'étayer leur publication, ce qui a donné le sujet de notre projet. Le public ciblé est alors aussi bien les scientifiques lisant la publication que n'importe quelle personne recherchant un outil de génération de surfaces de révolution discrètes.

Initialement, le projet devait aussi comprendre une dimension artistique. Pour cela, une troisième cliente, Aurélie Mourier, une artiste travaillant régulièrement avec des voxels, devait participer au projet. Son rôle aurait été de nous suggérer des fonctionnalités qui puissent intéresser d'autres artistes. L'évolution du projet n'a finalement pas permis d'intégrer Aurélie[1].

## 1.2 Définitions

Une surface de révolution est un objet mathématique en trois dimensions généré à l'aide de deux courbes en deux dimensions : la courbe de révolution et la méridienne. La courbe de révolution (le plus souvent un cercle) définit la transformation à appliquer à la méridienne afin de générer la surface de révolution. Un exemple de surface est visible sur la figure 1

Une méridienne peut être définie soit de façon explicite, avec une courbe définie par une fonction $y = f(x)$, soit de façon paramétrique avec une courbe définie par deux fonctions de la forme $x = f_x(t)$ et $y = f_y(t)$.

Une courbe de révolution doit être définie de façon implicite, à l'aide d'une équation de la forme $f(x, y) = 0$.

## 1.3 Problématique

Le but de ce projet est de construire une application web qui permette de créer des surfaces de révolution discrète à l'aide des algorithmes fournis par les clients. Cette application doit permettre à des utilisateurs mathématiciens comme à des utilisateurs

[1] voir risque "Ajout de clients" page xx

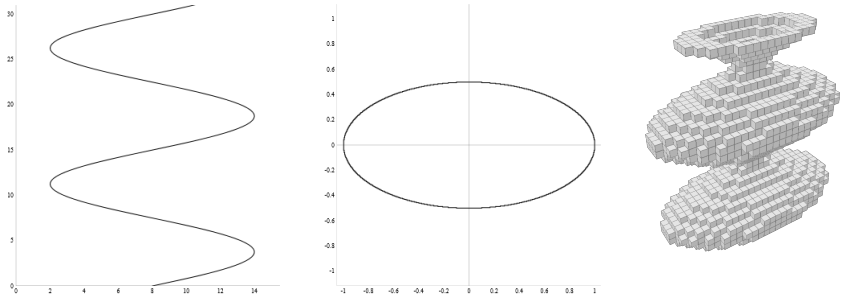


Figure 1: Une médiatrice, une courbe de révolution et la surface correspondante

lambda de générer facilement des surfaces de révolution et d'observer les résultats de l'algorithme.

La génération doit se faire à partir d'une méridienne et d'une courbe de révolution. Ces deux courbes peuvent être définies de différentes manières : choisies dans une liste de modèles proposés par l'application ou par une équation rentrée par l'utilisateur. La méridienne doit aussi pouvoir être dessinée à la main. Il est nécessaire que ces courbes soit visualisables dans un espace 2D.

Une fois la surface générée, celle-ci doit pouvoir être observée sous forme de coupes. L'application doit aussi proposer des solutions afin que la surface puisse être exportée à destination d'un logiciel de modelage ou d'une imprimante 3D.

## 1.4 Travail réalisé

Afin de répondre à cette problématique, nous avons réalisé une application web (cf. figure 2) se basant sur les technologies HTML5/CSS, Javascript et WebGL. Notre application se base sur le projet Bifurcations. Ce projet de master 1 (2014-2015), réalisé pour Aurélie Mourier, utilise lui aussi un espace 3D voxélisé.

Afin de rendre notre application accessible à tous, l'application propose des listes de courbes prédéfinies, parmi lesquelles l'utilisateur peut choisir sa méridienne ainsi que sa courbe de révolution. Chacune de ces courbes est associée à un ensemble de paramètres qui permettent de personnaliser simplement sa courbe (par exemple choisir la période d'une sinusoïde). Ceci permet de générer facilement et rapidement une surface de révolution. Afin de satisfaire également les personnes souhaitant un paramétrage plus approfondi de l'application, nous proposons également de définir les courbes par une équation que l'utilisateur peut rentrer au clavier.

L'application propose aussi d'observer les coupes liées à un voxel sélectionné ou de tronquer l'espace 3D à l'aide d'un jeu de sliders.

Enfin, l'application permet d'exporter la surface sous les formats X3D ou STL.

Le travail réalisé comprend également une documentation technique (architecture de l'application sous forme de diagrammes de classes et documentation du code générée

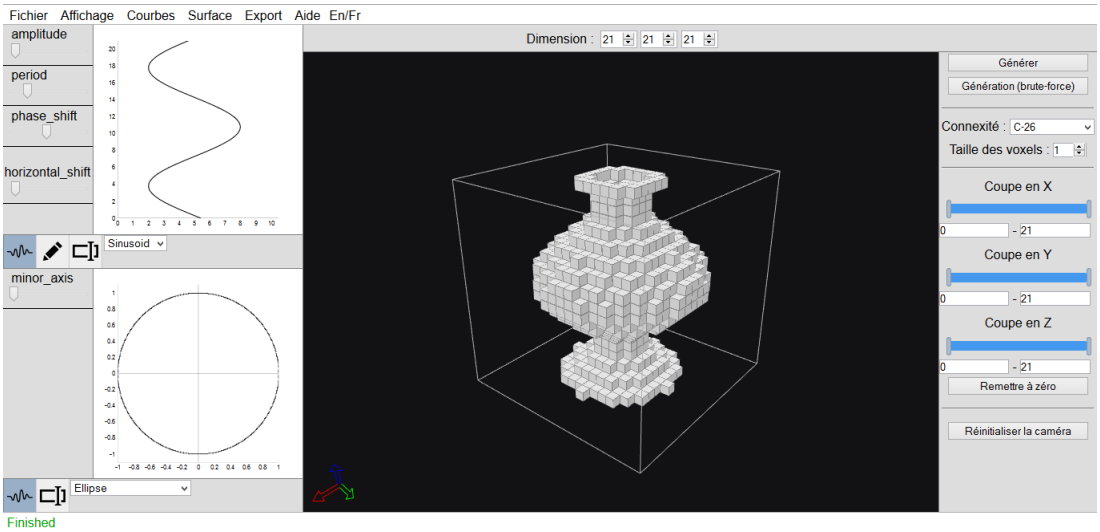



Figure 2: Application finale

avec JSDoc) qui permet de maintenir et de faire évoluer l'application.

## 1.5 Organisation du rapport

Dans ce rapport nous détaillons le déroulement du projet. Nous portons un intérêt particulier aux problèmes que nous avons rencontrés et aux solutions que nous avons envisagées pour y répondre. Pour cela, nous commençons par parler du travail technique qui a abouti à la version actuelle de l'application, puis nous expliquons la conduite de projet que nous avons mise en place tout au long de ces six mois pour organiser ce travail.

# 2 Travail réalisé

Dans cette section nous décrivons les fonctionnalités développées en lien avec le cahier des charges. Nous détaillons les difficultés que nous avons pu rencontrer ainsi que les solutions que nous avons mises en place.

## 2.1 Conception

Nous avons choisi d'utiliser une architecture de type Model-View-Controller pour notre application. Cela permet d'assurer une facilité d'évolution et de maintenance du code. Nous avons organisé l'ensemble des classes résultant en noyaux correspondant aux grandes fonctionnalités : le noyau 2D, le noyau 3D et le noyau de gestion de données.

## 2.2 Choix technologiques

La demande des clients est une application web, en prenant en compte que leur première version de l'algorithme fonctionne sous Mathematica, il s'offrait à nous les choix suivants :

- utiliser la version Mathematica déjà existante;
- choisir une autre technologie et retranscrire l'algorithme.

Dans un cas comme dans l'autre il nous fallait aussi choisir entre une application cliente ou une application serveur.

### Version Mathematica cliente

Une version de l'application cliente utilisant Mathematica implique que l'utilisateur possède une version du logiciel installée sur sa machine. Les licences de celui-ci étant payante, cette solution n'est pas envisageable.

### Version Mathematica serveur

Une version de l'application serveur implique une installation d'une infrastructure sur les serveurs de l'université de Poitiers. Une telle procédure aurait pris un temps excessif, de plus nous n'étions pas assuré que cela serait possible.

### Version cliente sans Mathematica

Une version cliente sans Mathematica implique que les calculs nécessaires sont effectués sur l'ordinateur de l'utilisateur. Ces calculs pouvant être intensifs, il est possible que certaines machines soient incapables de le gérer.

#### Version serveur sans Mathematica

Une version serveur sans Mathematica implique le transfert d'un grand nombre d'informations, ce qui ralentirait significativement l'application pour des connexions lentes. De plus il aurait fallu ajouter tout un aspect sécurité à l'application.

Parmi ces solutions, la version cliente sans Mathematica est la moins contraignante et nous permet de nous baser sur le projet Bifurcation[2]. C'est donc celle qui a été retenue. Ainsi le choix des technologies s'est porté sur du HTML5/CSS et Javascript pour la partie web ainsi que WebGL pour la partie 3D.

## 2.3 Fonctionnalités

Les fonctionnalités de l'application sont réparties en trois grands groupes :

- gestion et rendu des courbes 2D,
- gestion et rendu de la surface 3D,
- gestion de données (export, sauvegarde, chargement).

Dans cette section nous définissons le travail effectué pour chacune de ces sections.

### 2.3.1 Gestion et affichage des courbes 2D

La définition et l'affichage des courbes sont deux aspects importants de notre projet, les courbes étant fondamentale pour la génération de la surface.

#### Définition des courbes

Les courbes doivent pouvoir être définies de deux façons différentes : à l'aide de formules mathématiques (listes prédéfinies ou formules rentrées par l'utilisateur) ou par un dessin à la souris (qui est traduit en fonctions paramétriques dépendantes du temps).

Pour les courbes nécessitant une formule il était nécessaire de trouver un outil permettant de modéliser ces équations. Pour cela, nous avons utilisé la bibliothèque MathJS qui convient parfaitement à notre problème : à partir d'une équation entrée sous forme de chaîne de caractères, ce module nous permet d'obtenir un arbre. Cet arbre, composé de strings, d'entiers et d'opérateurs représente la formule. Cela facilite grandement l'exploitation de la formule sous forme de parcours d'arbre, etc.

Pour les courbes à main levée, il était nécessaire de définir le format de stockage mais aussi la technique de récupération des données. Nous stockons l'ensemble des points de la courbe sous forme de deux tableaux, chaque tableau correspondant respectivement aux coordonnées en $x$ et $y$ de chaque points. L'index des tableaux correspond à la coordonnées dans le temps. La technologie du canvas 2D native à HTML5 satisfaisait parfaitement ce dernier besoin. En effet, les canvas permettent d'obtenir les points de la courbe à partir des mouvements de la souris.

[2] cf. 1.4 Travail réalisé p4

### Paramètres des courbes

L'application devait proposer de modifier les paramètres des courbes. Les problèmes liés à ceux-ci sont le choix des paramètres proposés ainsi que la façon de les présenter. Afin de choisir une solution, nous avons discuté de cela avec les clients. Il a ainsi été décidé que seuls les méridiennes et le cercle, pour les courbes de révolution, présenterait des paramètres.

### Affichage des courbes

L'application devait pouvoir afficher des courbes explicites et implicites. Alors que le calcul des points des courbes explicites est trivial, les courbes implicites de par leur nature nécessitent un traitement particulier pour l'affichage. En effet, celles-ci sont représentées par une équation. Sans traitement, les points appartenant à la courbe ne sont pas connues. Une solution revient à tester chaque pixel.

Nous avons envisagé plusieurs solutions :

- parcourir l'espace en x et en y selon un pas prédéfini. Il faut alors tester chaque couple $(x, y)$. On obtient ainsi un ensemble de points appartenant à la courbe qu'il faut ensuite relier pour obtenir le graphe de la courbe.
- utiliser JSXGraph, une bibliothèque permettant d'afficher des courbes, qui fonctionne particulièrement bien avec MathJS.
- utiliser functionPlot, une bibliothèque permettant d'afficher des courbes.

La première solution s'est révélée trop longue, notamment lorsque le pas choisi est petit. En revanche, avec un pas plus élevé, il fallait relier les points entre eux alors que nous ne connaissions pas l'ordre de ceux-ci. Nous étions donc incapables d'afficher une courbe correcte dans un temps acceptable, c'est pourquoi cette solution n'a pas été retenue.

La solution utilisant JSXGraph propose de nombreuses fonctionnalités intéressantes, notamment la possibilité de modifier la fonction à partir de points de contrôles apparaissant sur le dessin de la courbe. Cependant ce module ne supporte pas les courbes implicites et a donc été abandonnée.

La bibliothèque functionPlot s'est donc imposée comme la seule solution viable. En effet elle nous a permis d'afficher les courbes définies par des formules avec un temps de calcul acceptable. Il suffit de préciser quelques paramètres tels que l'intervalle d'affichage, la fonction sous forme de chaîne de caractères et son type.

### 2.3.2 Génération et affichage de la surface 3D

#### Génération

Afin de générer une surface de révolution, nous utilisons les algorithmes fournis par les clients. Il en existe deux différents en fonction du type de médiatrice utilisée (fonction explicite ou dessinée).

Le premier algorithme nécessite de calculer les valeurs des fonctions définissant les courbes.
Pour cela, nous avons naturellement choisi d'utiliser MathJS, qui avait déjà été mis en place pour la gestion des courbes et qui convenait parfaitement à ce problème.

Les difficultés du second algorithme se trouvent au niveau de la génération de la liste de point correspondant au dessin : si les points sont trop proches les uns des autres, la génération devient trop lente, tandis que si ceux-ci sont trop éloignés, la surface générée n'est pas connexe.
Afin de régler le problème de lenteur, nous avons décidé d'ignorer chaque point qui serait trop proche du point précédent. La solution du second problème nous a été fournie avec l'algorithme : ajouter des points jusqu'à ce que la distance maximale entre deux points soient égale ou inférieure à un voxel. Les points ajoutés sont déterminés par interpolation linéaire.

La génération effectuant des calculs trop intensifs, et relativement longs, même après optimisation, le navigateur ne répondait plus.
Pour palier ce problème, la solution retenue a été celle des workers. En effet, les workers permettent de mettre en place des threads dans un processus Javascript. Nous avons implémenté l'algorithme sur un deuxième thread ce qui a permis de libérer le navigateur pendant la génération.

Afin de minimiser les temps de génération, nous avons cherché à accélérer au maximum les algorithmes. Les versions qui nous ont été fournies parcouraient chaque voxel de l'espace afin de déterminer son appartenance à la surface.
Nous avons cherché à améliorer cette technique en envisageant les solutions suivantes :

- accélérer la version brute force en divisant l'espace dans plusieurs threads à l'aide de workers;
- accélérer la version brute force en la parallélisant à l'aide de WebCL;
- mettre en place une génération incrémentale : on parcourt uniquement les voisins des voxels qui appartiennent à la surface. Le premier est trouvé par brute force.

La première solution est facile à mettre en place, les workers étant déjà utilisés pour d'autres raisons. Cependant, la parallélisation se déroulant sur CPU et non sur GPU, la différence dans le temps de calcul est peu importante.

L'utilisation de WebCL aurait permis de paralléliser l'algorithme sur une carte graphique. Cela aurait permis de traiter un grand nombre de voxels en même temps, divisant le temps d'exécution d'autant. Cette solution aurait permis une accélération importante de la génération, cependant le module WebCL nécessite d'être installée sur le navigateur client pour fonctionner. Cela nuisait grandement à la facilité d'utilisation de l'application et cette solution a donc été abandonnée.

La dernière solution permet d'éviter un nombre important de calculs inutiles. Cependant, elle se base sur l'hypothèse que la surface générée est connexe, si ce n'est pas le cas, la génération sera incomplète.

La plupart des surfaces traitées étant connexes, nous avons gardé la dernière solution. Cependant, afin de pouvoir traiter tous les cas, nous avons aussi mis en place la première solution, en conseillant de ne l'utiliser que si la surface générée ne semble pas être celle attendue.

### Rendu

Deux problèmes importants se sont présentés lors de la mise en place du rendu 3D : la gestion de mémoire liée à WebGL et la vitesse de rendu.

Les buffers de WebGl possèdent une taille limitée. En conséquence, en utilisant un seul buffer, l'application ne permettait pas d'afficher tous les voxels lorsque la surface était trop grande.

Pour palier ce problème l'application utilise un nombre de buffers dépendant directement du nombre de voxels à afficher. Il est ainsi possible d'afficher les surfaces complètes quelle que soit leur taille.

Le rendu coûte beaucoup de ressources, notamment au niveau du calcul de visibilité et est donc lente. Cela vient du fait que les surfaces sont composées de voxels et comprennent donc de nombreuses faces qu'il est inutile d'afficher (ces faces sont invisibles lorsque deux voxels sont collés). Les solutions envisagées pour résoudre ce problème sont :

- Laisser tous les calculs de visibilité à la carte graphique. Ils sont alors gérés nativement par WebGL.
- Calculer la visibilité de chaque face au moment de la préparation du rendu. C'est la solution qui était déjà proposé dans l'ancien projet.
- Calculer la visibilité des faces concernées lors de l'ajout d'un voxel.

La première solution est particulièrement lente. En effet, WebGL n'est pas fait pour gérer des objets dont un nombre de triangle aussi élevé ne sera pas affiché.

La seconde solution rend la préparation lente. La surface demandant à être préparée dans plusieurs situation (premier affichage, modification des coupes, ...), l'application perdait de son interactivité.

La dernière solution ralentit légèrement la génération, cependant, grâce à cela, le calcul de visibilité n'est effectué qu'une seule fois par surface. Nous avons donc choisi la dernière solution afin de privilégier l'intéractivité de l'application.

### 2.3.3 Gestion de données

L'application propose d'exporter les résultats obtenus à destination de différentes utilisations. Pour cela, elle autorise différents format d'export : PNG, X3D et STL.

#### Export PNG

La difficulté de l'export PNG réside dans le fait que les courbes ne sont pas affichées de la même façon selon leur type. En effet, les courbes dessinées utilisent les canvas HTML5 alors que les courbes définies par des formules sont affichées en SVG.

Dans l'optique d'enregistrer les courbes au format SVG, l'application utilise la bibliothèque saveSvgAsPng qui transforme le SVG en canvas ce qui simplifie l'export PNG.

L'enregistrement des canvas quant-à-lui s'effectue à l'aide de la bibliothèque FileSaver qui permet de proposer à l'utilisateur un fichier PNG représentant la courbe qu'il veut enregistrer.

#### Export 3D

Pour exporter la surface dans un format 3D (X3D ou STL), il faut que ces fichiers respectent les standards de ces formats.

Nous avons envisagé d'utiliser des bibliothèques permettant de convertir une surface WebGL en objet X3D ou STL, cependant nous n'avons trouvé aucune bibliothèque gérant cela. La solution choisie est donc d'utiliser notre propre algorithme permettant de retranscrire la surface dans un fichier XML.

# 3 Conduite de projet

## 3.1 Rôles

Notre équipe est composée de quatre personnes :

- Thomas Benoist - Chef de projet
- Zied Ben Othmane - Responsable qualité
- Adrien Bisutti - Responsable des risques
- Lydie Richaume - Responsable des tâches

## 3.2 Méthode de développement

Nous avons choisi un cycle de développement en spirale. En effet, cela nous permet de travailler de manière incrémentale et de garder un contact constant avec les clients afin d'être certains de leur satisfaction, tout en gardant une certaine souplesse quant à la durée de chaque cycle.

Cette méthode de développement s'est traduite par la mise en place de cinq cycles de durées variables, chacun d'entre eux se soldant par un livrable. Chacun de ces livrables a donné lieu à une réunion avec les clients. Ces réunions nous ont permis d'obtenir des retours sur l'état actuel de l'application. En prenant en compte les remarques des clients dans les cycles suivants, nous avons pu nous assurer que la version finale leur convienne.

En plus de ces réunions régulières avec les clients, nous avons mis en place des discussions hebdomadaires avec notre encadrant pédagogique. Cela nous a permis de faire le point chaque semaine sur l'avancement du projet, les derniers problèmes rencontrés, ainsi que l'organisation de la semaine suivante.

De façon plus formelle, nous avons aussi participé à plusieurs évaluation, de notre avancement et de notre gestion de projet, au cours des audits suivants :

- Le Go/No go : Il nous a permis de prendre connaissance du projet et d'expliquer aux clients ce que nous pouvions leur proposer, avec une première approche technique et de gestion de projet.
- La réunion de lancement : Au cours de cette réunion, nous avons eu à démontrer que notre projet reposait sur une base solide (planification, gestion des risques, gestion des coûts...). Cela correspond aussi à la signature du cahier des charges.
- La réunion de suivi[3] : Cette réunion interne a permis d'illustrer nos discussions avec l'encadrant pédagogique aux yeux de l'auditeur du projet.
- La réunion de revue : Cette réunion avec les clients permet de faire le point sur l'état d'avancement du projet, sur le reste à faire et les informer de ce qui pourra ou ne pourra pas être réalisé.

[3] La fiche de reporting de cette réunion est disponible en annexe 1. Une version plus lisible peut être trouvée dans le CD de mémoire

## 3.3 Planification

Lors de la planification, nous avons défini des tâches en lien avec les fonctionnalités du cahier des charges. Les tâches concernant une même fonctionnalité ont été regroupées dans des macro-tâches, dont la liste est fournie dans la figure 3.

| | |
|---|---|
| 1 - Documentation, test et aide utilisateur | |
| 2 - Conception | |
| 3 - Noyau fonctionnel | 4 - Interface minimale |
| 6 - Ajout de fonctionnalités | 5 - Amélioration IHM Choix des courbes |
| 8 - Dessin à main levée méridienne | 7 - Amélioration IHM Paramètres |
| 9 - Gestion des données | 10 - Amélioration IHM Rentrer des formules |
| 11 - Ajout courbes utilisateur | |
| 12 - Rédaction rapport technique | |

Figure 3: Liste des tâches

Dans cette liste, on peut distinguer deux types de tâches : celles liées aux fonctionnalités et celles liées à l'interface. Comme nous l'expliquons ci-après, cette distinction permet de paralléliser le travail.

Pour optimiser le temps de travail, nous avons parallélisé les tâches portant sur les mêmes fonctionnalités. Le résultat de cette élaboration est le diagramme de Gantt de la figure 4.

Naturellement, notre projet a évolué, et nous avons modifié nos diagrammes de Gantt. Ces modifications ont été effectuées à plusieurs moments :

- Annulation de la tâche 11 : Nous avons accumulé du retard durant la tâche de conception dans l'optique de régler les problèmes identifiés en amont. Comme nous pouvons le voir sur le diagramme d'avancement (figure 5), ce retard s'est répercuté sur le reste du projet et il nous a fallu annuler la dernière tâche (spécifiée dès le début comme optionnelle) afin de respecter les délais.

- Modification du diagramme de référence :

  Au cours du refactoring et de la création des classes, nous nous sommes rendus compte de l'incohérence d'une tâche : 3.3 Implémentation des classes. Cette tâche impliquait que nous développions au minimum le squelette de toutes les classes prévues. Cependant nous avons supposé (à raison) que lors du développement, nous serions amenés à modifier les classes déjà existantes ou à en ajouter de nouvelles.

  Il a alors été décidé de fractionner la tâche 3.3 pour incorporer de nouvelles tâches dans chaque macro-tâche où il serait nécessaire de créer des classes. On peut

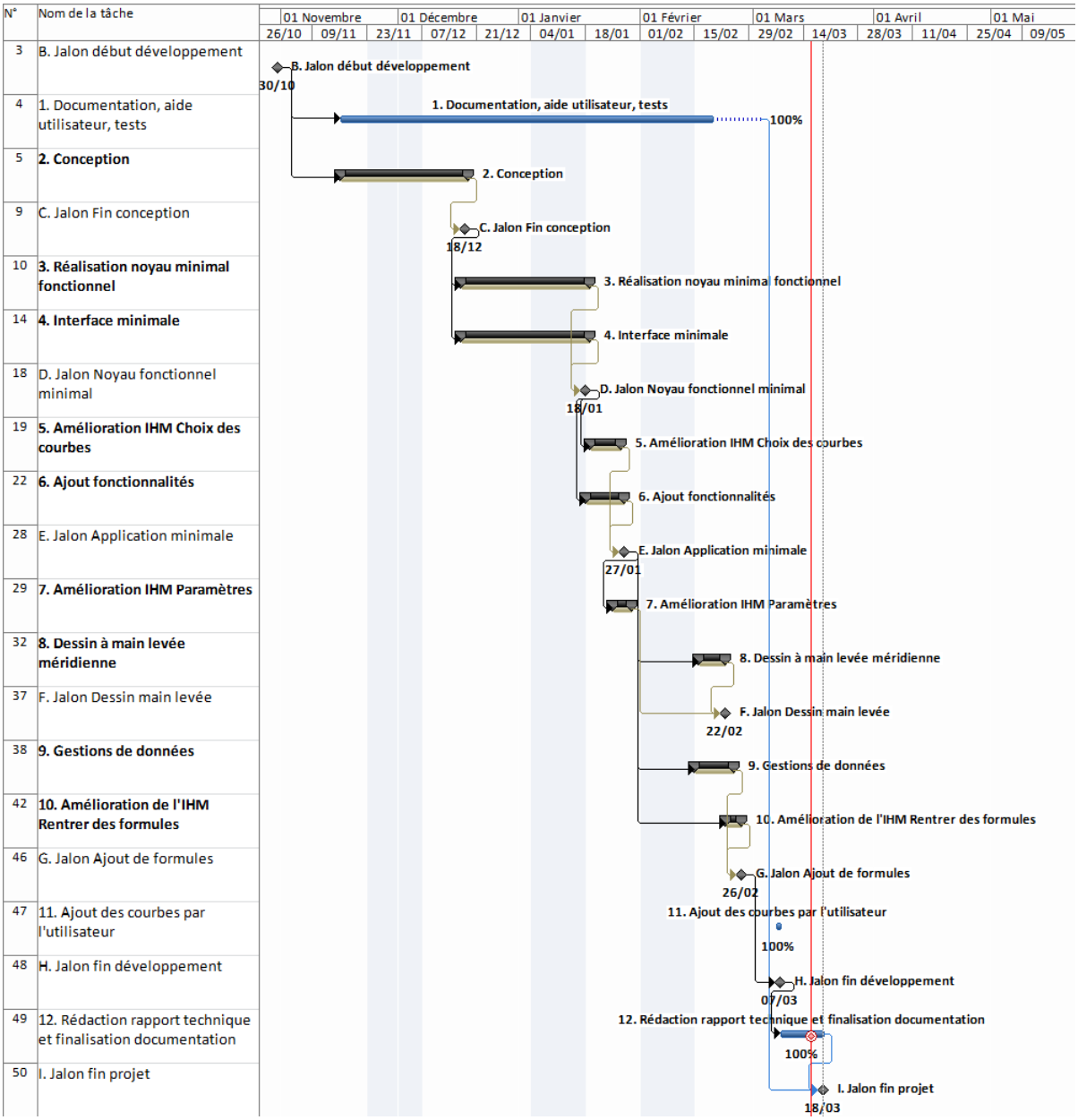



Figure 4: Gantt : Projet complet

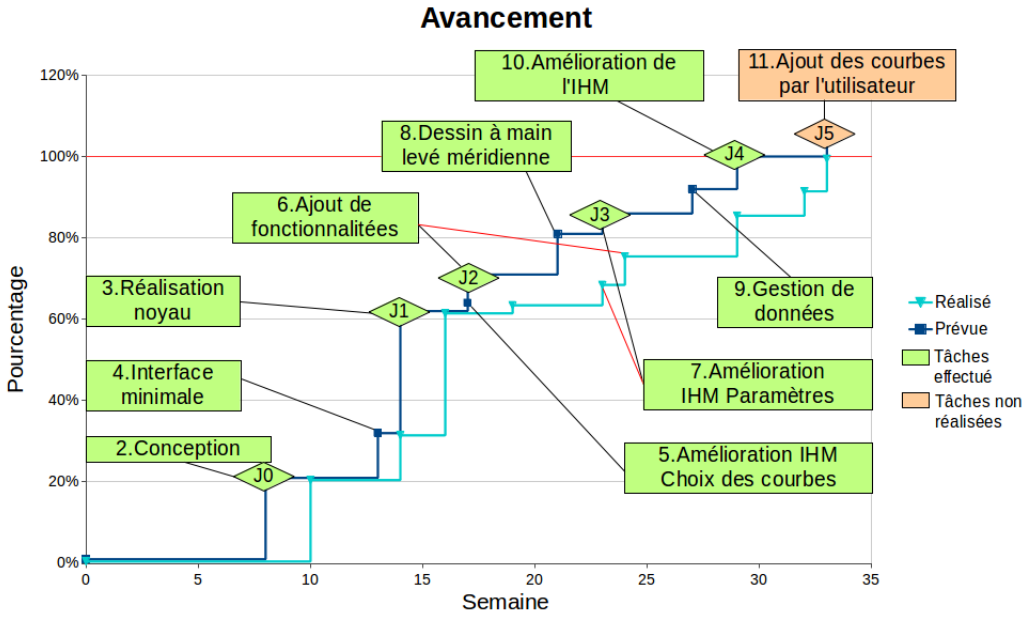



Figure 5: Courbe d'avancement

observer un exemple de tâche ajoutée sur la figure 6 : 6.1 Implémentation des classes courbes

- Modification du diagramme de suivi :

  Pour pouvoir livrer la version finale de l'application à la date prévue, nous avons été amené à modifier l'organisation des tâches restantes entre-elles (cf. figure 7). Les macro-tâches 9 et 10, initialement séquentielles à la tâche 8 ont finalement été développées en parallèles de cette dernière.

Le développement de chacune des grandes fonctionnalités a engendré un livrable. Ces grandes fonctionnalités étant aux nombre de cinq, nous avons initialement prévus cinq livrables à destination des clients. La liste de ces livrables peut être consultée sur la figure 8.

Suite aux retards et aux modifications du diagramme de Gantt, nous avons modifié le planning de certains livrables. De plus, certaines des tâches n'ayant pas été effectuées dans l'ordre prévu, nous avons ajouté des livrables intermédiaires. Ainsi, le livrable 2 bis est un livrable intermédiaire faisant le lien entre les tâches 5 et 6 et les tâches 7 et 8. Les livrables 3 et 4 ont quant à eux été fusionnés en un seul.

L'application livré lors de l'audit de livraison, ne contenait pas la documentation complète. Il a donc été convenu qu'elle serait livrée le 11 Mars. Suite à une panne de réseau, nous avons dû reporter sa livraison au 14 Mars.

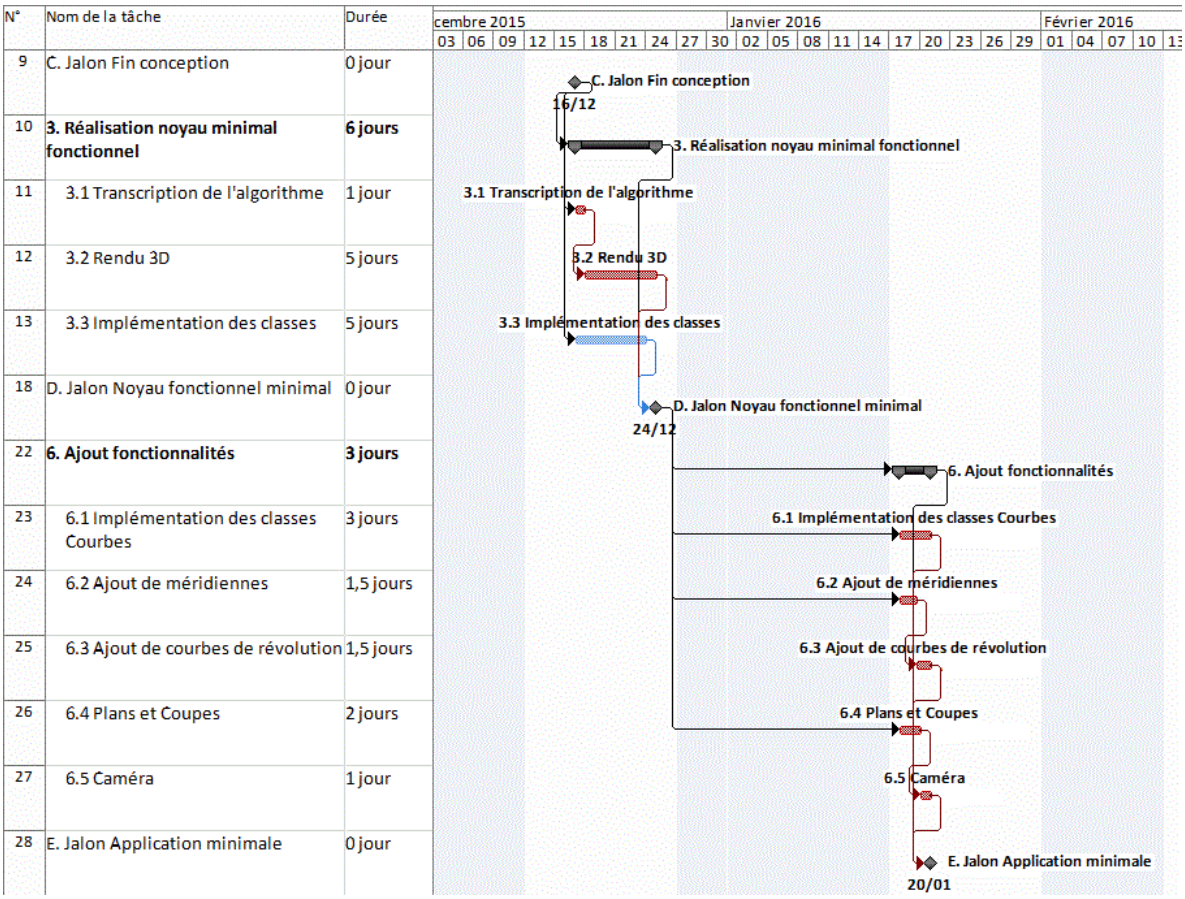



Figure 6: Gantt : Ajout de tâches

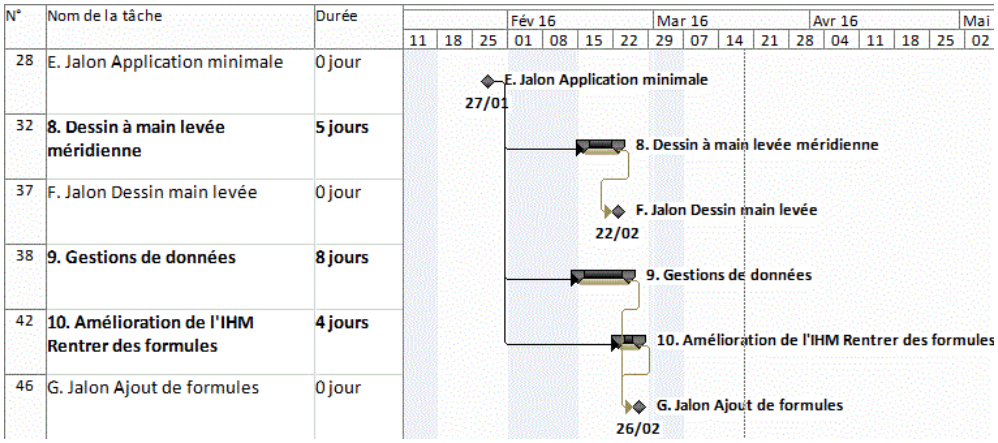



Figure 7: Gantt : Parallélisation des tâches

| N. | Livrable | Tâches | Date prévue | Date effective |
|---|---|---|---|---|
| 1 | Résultat de l'algorithme et interface | 2, 3, 4 | 23/12 | 18/01 |
| 2 | Application minimale | 5, 6 | 21/01 | 25/01 |
| $2^{\text{bis}}$ | Multicoupe et paramètres | 7 | — | 29/01 |
| 3 | Courbes avec paramètres modifiables et tracé à main levée | 7, 8 | 29/01 | 24/02 |
| 4 | Équations et export | 9, 10 | 19/02 | 24/02 |
| 5 | Application finale ~~et documentation~~ | 1 - 11 | 02/03 | 02/03 |
| $5^{\text{bis}}$ | Documentation technique | 1 | 02/03 | 14/03 |

Figure 8: Tableau des livrables

## 3.4 Gestion des risques

Nous avons identifié une vingtaine de risques. La liste complète est fournie en annexe 2. Les plus importants sont :

- la possibilité d'un nouveau client ;
- la possibilité d'une évolution de l'algorithme fourni par le client ;
- la possibilité que le rendu 3D demande trop de ressources .

### 3.4.1 Nouveau client

Nous avons été prévenus dès le début du projet que celui-ci avait été proposé comme projet transdisciplinaire au sein de l'université. Cela pouvait se traduire par un ajout de clients menant possiblement à une modification du cahier des charges. Cela aurait donc pu impacter fortement notre projet, notamment au niveau des délais : en effet, la conception réalisée en début de projet aurait pu ne plus convenir et nous aurions pu avoir à modifier des fonctionnalités déjà développées.

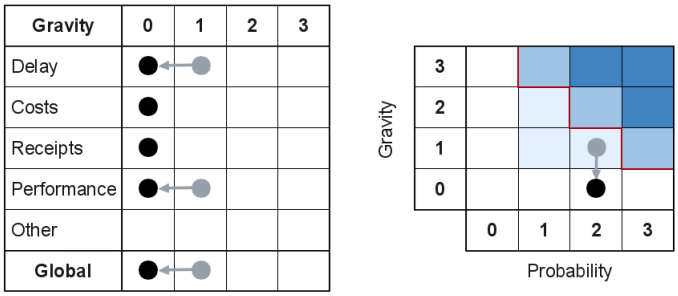



Figure 9: Risque de nouveau client

Afin de pouvoir anticiper ces problèmes, il a été décidé que nous demandions un maximum d'informations concernant les changements potentiels à nos clients et que nous nous tenions régulièrement informés des évolutions du dossier.

Ce risque s'est avéré (le 1er décembre 2015) : le projet de l'université a été accepté et nos clients ont assisté à une réunion avec les nouveaux clients. Il en est rapidement ressorti que l'université n'interviendrait pas directement dans notre projet. Ce risque n'a donc eu aucun impact et son degré de gravité a diminué.

### 3.4.2 Évolution de l'algorithme

Il a été annoncé dès le début du projet que nos clients travaillaient encore sur de nouvelles versions de leur algorithme. Il était donc possible que nous ayons à inclure des fonctionnalités supplémentaires à celles spécifiées au départ du projet. Cela pouvait donc impacter nos délais par l'ajout de tâches en résultant.

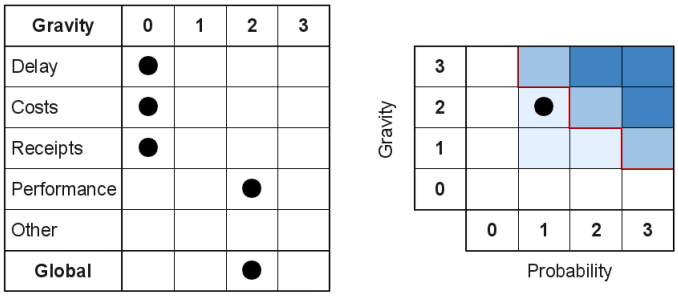



Figure 10: Risque d'un nouvel algorithme

Afin d'éviter que cela nous fasse prendre un retard trop grand, il a été décidé que nous n'accepterions pas de modifications annoncées moins d'un mois avant la fin du projet, et que nous nous informerions régulièrement auprès des clients de l'avancement de leurs recherches afin d'anticiper toute modification de planning.

Ce risque ne s'est finalement pas avéré et les discussions avec les clients nous ont permis de diminuer la probabilité du risque au fil du temps.

### 3.4.3 Lenteur du rendu

Notre projet reprenait le squelette d'un projet précédent et un problème connu était la lenteur du rendu 3D lors de l'affichage d'une quantité trop importante de voxels. Cela pouvait donc avoir un impact important sur les performances de l'application. Afin de palier ce problème, nous avons prévu une tâche visant à améliorer l'algorithme de génération.

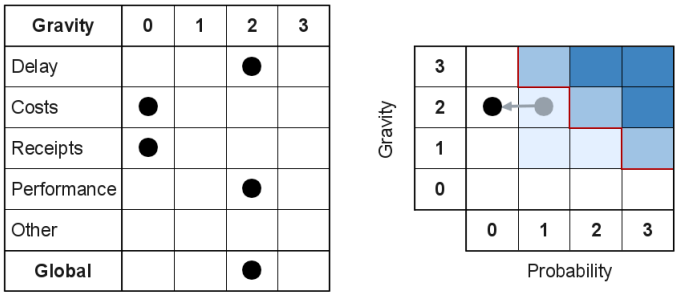



Figure 11: Risque de lenteur du rendu

Cette tâche a effectivement permis d'obtenir un temps de rendu acceptable sur la plupart des machines, mais le temps de rendu peut rester long lorsque l'on choisit les dimensions maximales pour l'espace 3D.

### 3.4.4 Problèmes liés au serveur

Bien que nous développions une application web, nous n'avons pas eu à gérer les risques liés à l'utilisation d'un serveur. En effet, notre projet est développé uniquement en HTML et Javascript donc s'exécute uniquement chez le client. Il était par ailleurs mentionné dans le cahier des charges que l'hébergement de l'application serait laissé à la charge du client.

| Gravity | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Delay | ● | | | |
| Costs | ● | | | |
| Receipts | ● | | | |
| Performance | ● | | | |
| Other | | | | |
| **Global** | ● | | | |

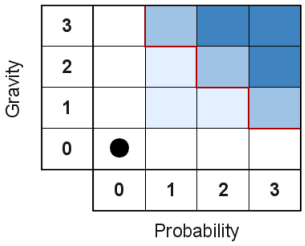



Figure 12: Risques liés au serveur

## 3.5 Plan qualité logiciel

Dans le cadre de notre gestion de projet, nous avons choisis de mettre en place une évaluation de la qualité de notre logiciel. Cette évaluation nous permet de nous assurer que le code que nous produisons respecte les normes de développement en place et que l'application est conforme aux besoins des client.

### 3.5.1 La norme ISO 9126

Nous avons mesuré la qualité logicielle en nous basant sur la norme ISO 9126 qui est une norme standard internationale. Elle normalise et classifie un certain nombre de principes qualité. Elle est composée de six caractéristiques générales qui définissent la qualité globale d'une application : la capacité fonctionnelle, la fiabilité, l'efficacité, la maintenabilité, la facilité d'usage et la portabilité. Chacune de ces caractéristiques est décomposée en sous caractéristiques.

Cette norme semble être une bonne approche pour déterminer la qualité de notre logiciel dans son ensemble et fournir une vue globale satisfaisante.

### 3.5.2 Démarche générale

Pour mesurer la qualité de notre logiciel nous avons commencé par déterminer son adéquation par rapport aux objectifs de départ et aux standards de programmation. Nous avons ensuite défini précisément ce que l'application doit faire et comment elle doit le faire, tant d'un point de vue fonctionnel que d'un point de vue technique. Après

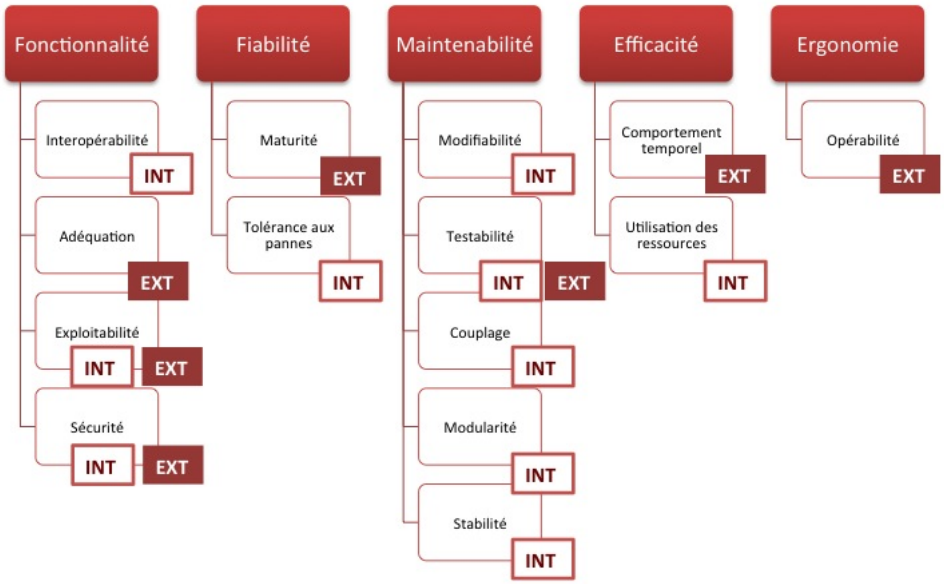



Figure 13: Résumé des attributs qualité ISO 9126

avoir fixé ces objectifs, nous avons appliqué un ensemble de règles et de mesures afin de calculer la différence entre les objectifs attendus et la réalisation obtenue .

Nous avons suivi le schéma de la figure 14 pour mesurer la qualité de notre application. Une première étape de l'évaluation est réalisée par les développeurs. Cette étape consiste à vérifier que les modules qu'ils ont programmés ne comporte pas de bugs. La deuxième étape est la validation de l'application par le responsable qualité. Celui-ci va tester l'application dans son ensemble et ainsi attribuer des notes selon certains critères décrit dans la section ci-après. Cela permet de garantir non seulement la qualité du code mais aussi de détecter les bugs liés à l'intégration de nouvelles fonctionnalités. Le responsable qualité va ensuite fournir un questionnaire aux clients qui vont évaluer la qualité du livrable fourni. L'ensemble de ces deux évaluations va permettre d'attribuer une note globale de qualité à l'application. Selon les remarques faites par les clients, le responsable qualité peut décider de rajouter des tâches pour le prochain cycle pour prendre en compte les remarques des clients et effectuer les modifications nécessaires avant le livrable suivant.

### 3.5.3 Les métriques de mesure de la qualité

Nous identifions six principes fondamentaux pour évaluer notre travail en suivant les indicateurs de qualité de la norme ISO 9126. Ces principes sont illustrés dans la mesure des critères de la norme dont les détails sont expliqués dans le rapport PAQL:

1. La capacité fonctionnelle, qui mesure la capacité qu'ont les fonctionnalités d'un logiciel à répondre aux exigences et besoins explicites ou implicites des usagers.

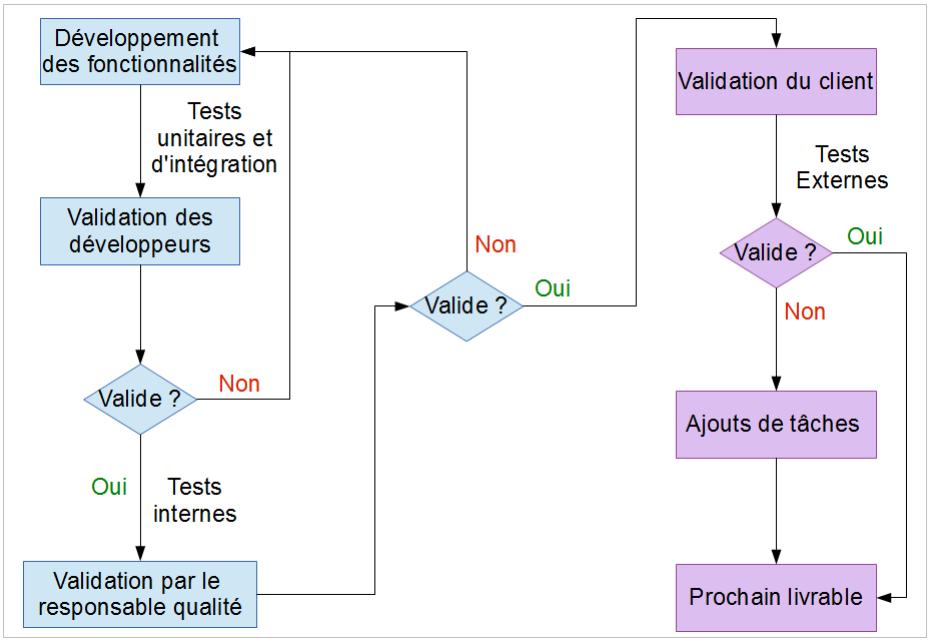



Figure 14: Schéma de mise en place de la qualité logicielle

2. La facilité d'utilisation, qui mesure l'effort nécessaire pour apprendre à manipuler le logiciel.

3. La fiabilité, qui mesure la capacité d'un logiciel de rendre des résultats corrects quelles que soient les conditions d'exploitation.

4. La performance, qui mesure le rapport entre la quantité de ressources utilisées (moyens matériels, temps, personnel), et la quantité de résultats délivrés.

5. La maintenabilité, qui mesure l'effort nécessaire à corriger ou transformer le logiciel.

6. La portabilité, qui mesure l'aptitude d'un logiciel de fonctionner dans un environnement matériel ou logiciel différent de son environnement initial.

### 3.5.4 Résultats obtenus

Nous avons généré des modèles statistiques pour mettre en évidence toute les mesures de qualité des différentes versions de notre application, ces modèles fournissent des indications précises sur l'utilisation et l'évaluation de la qualité logicielle. Il s'agit de donner du sens à des informations brutes.

- Fonctionnalité : ce critère vérifie que le livrable à évaluer comporte les fonctionnalités attendues, que le code développé respecte les standards et qu'il ne comporte

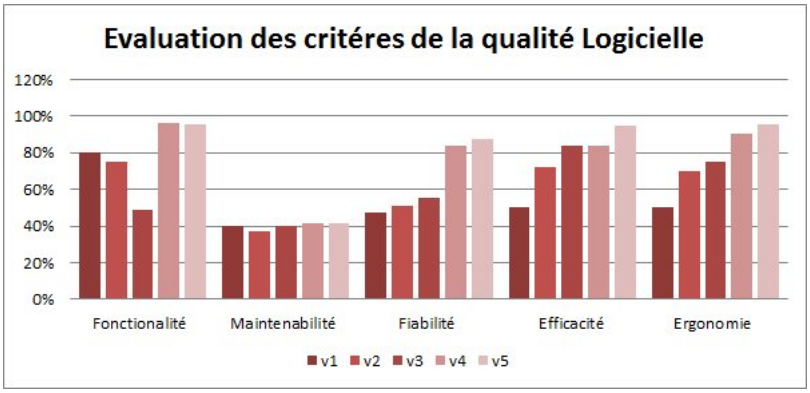



Figure 15: Résumé des critères de qualité logicielle

pas de bugs. L'évolution de ce critère pour les versions de notre application sur l'ensemble des évaluations a augmenté pour atteindre environ 95%. On dénote cependant une baisse de l'évaluation sur la troisième version puisque cette version était un livrable ne contenant qu'une partie des fonctionnalités attendues.

- Maintenabilité : ce critère s'évalue en testant si les fonctionnalités du livrables peuvent être modifiées sans impacter le fonctionnement de l'application. Nous évaluons également la testabilité de nos composants, c'est à dire si il est facile d'établir des plans de tests pour vérifier leur fonctionnement. Il faut aussi mesurer les relations entre les composants de l'application. Enfin, nous évaluons l'importance de l'existance des composants dans le fonctionnement du livrable en supprimant certaines fonctionnalités. De par le fort couplage existant au sein de l'application tout au long du développement, ce critère n'a que faiblement évolué :de 40% à 41%.

- Fiabilité : ce critère évalue la maturité du logiciel, c'est à dire l'évaluation du fonctionnement du livrable avec des erreurs. Il évalue aussi la tolérance du livrable aux pannes : à quel niveau le livrable reste opérationnel en cas d'erreurs. On peut observer sur la figure 15 qu'au fil de chaque livrable, la fiabilité augmente jusqu'à un niveau d'environ 90%, cette évolution s'explique par la modularité de l'application dans ses fonctionnalités.

- Efficacité : ce critère évalue le temps de réponse de l'application selon les navigateurs. Puisqu'au cours du temps, la vitesse de rendu et la vitesse de génération (qui sont les points critiques) ont été accélérés, ce critère a évolué jusqu'à atteindre 95% sur la dernière version.

- Ergonomie : ce critère s'évalue avec l'aide des clients, ils répondent à un questionnaire et donne leur avis sur l'interface que propose l'application. L'interface proposée à chaque livrable a été modifiée en accord avec les remarques des clients ce qui explique que ce critère augmente jusqu'à atteindre 95% pour le dernier livrable.

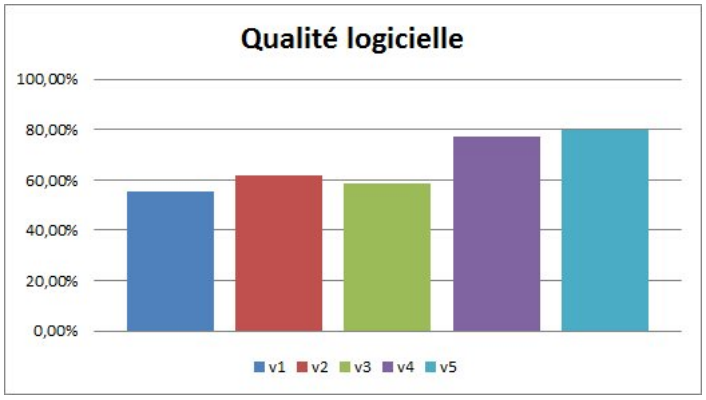



Figure 16: Evolution de la qualité logicielle

La qualité globale des livrables a été évaluée en faisant la moyenne des coefficients décrits précédemment. Nous pouvons observer sur la figure 16 qu'au fur et à mesure de l'avancement du projet, les livrables proposent une qualité croissante, jusqu'à une qualité de 80% pour la version finale. Nous dénotons une décroissance au niveau du troisième livrable qui s'explique par le fait que celui-ci était un livrable ne contenant pas la totalité des fonctionnalités attendues.

## 3.6 Coûts

La dimension coût du projet s'est basée sur un salaire de sept cents cinquante euros par semaine pour un jeune ingénieur. Nous avons identifié dix semaines de travail et nos seules dépenses correspondent aux salaires. Nous en déduisons une dépense totale de trente mille euros pour l'ensemble de projet. En appliquant une marge de trente pourcent, nous avons facturé le projet pour quarante mille euros.

Globalement le temps de travail réel s'est révélé inférieur au temps prévu. Cela s'explique par les cours qui ont été ajoutés durant les périodes de projet ainsi que des dysfonctionnements du réseau. Comme on peut le voir sur le diagramme de coût (figure 17) ces problèmes réduisent les dépenses réalisées de plusieurs jours-hommes. On peut également observer une dépense supérieure à celle prévue en Janvier : cela s'explique par une mauvaise allocation du temps de travail par tâche lors de la planification initiale. En effet, une personne du groupe n'avait pas de tâche attitrée sur cette période et n'a donc pas été comptabilisée dans le calcul des coûts.

L'échelonnement des paiements dont il a été convenu avec les clients est le suivant :

- à la signature du cahier des charges, un premier versement de douze mille euros, correspondant à trente pourcent de la somme totale;
- à chaque livrable, un versement intermédiaire de quatre mille euros, correspondant à dix pourcent du total;

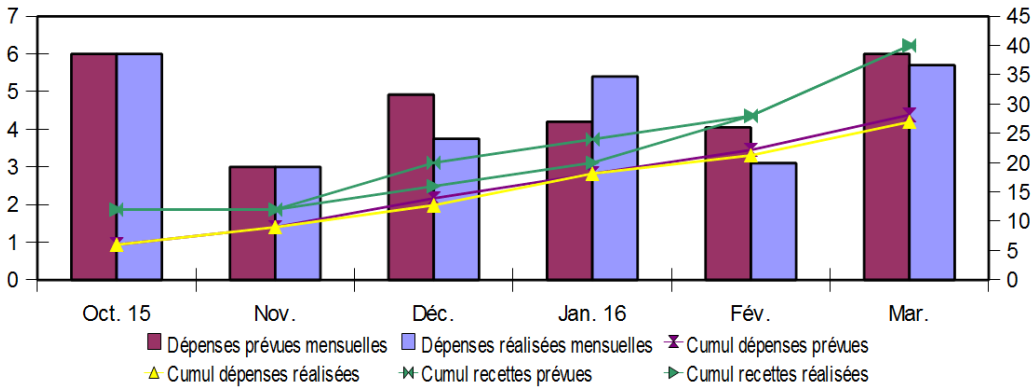



Figure 17: Diagramme de coûts

- à la livraison finale, un versement de douze mille euros, correspondant à trente pourcent de la somme totale.

Les livrables qui n'étaient pas initialement prévus n'ont pas donné lieu à un versement supplémentaire. En revanche, les livrables 3 et 4 qui ont été fusionnés, ont donné lieu à un paiement de huit mille euros, soit la somme du montant lié à chacun de ces livrables.

# 4 Conclusion

## 4.1 Contexte

L'origine du projet est l'élaboration d'algorithmes par Eric Andres et Gaëlle Largeteau-Skapin. Ces algorithmes permettent la modélisation de surface de révolution discrètes.

## 4.2 Problématique

L'objectif du projet était donc de fournir une application web illustrant ces algorithmes et que des utilisateurs scientifique ou non peuvent manipuler. Via cette application il doit être possible de générer une surface de révolution à partir d'une méridienne et d'une courbe de révolution. Les courbes peuvent alors être proposées par l'application, rentrée par l'utilisateur ou dessinée à la souris pour la méridienne. La surface de révolution doit être visible dans un espace 3D et les courbes dans un espace 2D.

## 4.3 Travail effectué

Le travail effectué répond à la problématique posée au début du projet et reprise dans le cahier des charges. Ainsi, il est possible de visualiser la méridienne et la courbe de révolution dans un espace 2D. L'application propose à l'utilisateur une liste de courbes prédéfinies ou de rentrer une équation. Elle intègre également l'algorithme fournie par les clients qui permet de dessiner la méridienne à la souris. L'affichage des courbes définies par une équation est réalisé avec la bibliothèque functionPlot tandis que celui des courbes dessinées à la souris est géré avec les canvas de HTML5.

La surface de révolution discrète peut être visualisée dans un espace 3D une fois la méridienne et la courbe de révolution choisies. La génération de cette surface utilise la bibliothèque MathJS et le rendu 3D à l'aide de la technologie WebGL.

## 4.4 Critique des résultat

Dans l'ensemble, les clients sont satisfaits de l'application. On peut cependant critiquer certains points :

- L'utilisation d'un espace 3D de trop grande taille peut provoquer des lenteurs. Dans ce cas, la génération et le rendu sont gourmands en ressources et prennent beaucoup de temps à s'effectuer. Puisque l'application est une application cliente, ce problème peut s'accentuer sur les machines possédant une petite configuration.

- La gestion des courbes que nous avons choisie pour l'application fait que certaines courbes (définies différemment selon les intervalles par exemple) ne seront pas affichées et mal générées.

- Cette même gestion des courbes implique que les paramètres permettant de manipuler la méridienne ou la courbe de révolution ne sont pas aussi étoffés que ce

qui était initialement prévus. Cela s'explique par les limitations mises en places pour que l'utilisateur ne fasse pas n'importe quoi.

- La définition d'une courbe par une chaîne de caractère entrée par l'utilisateur peut être difficile d'utilisation. En effet, elle ne propose aucun retour, que ce soit pour des opérateurs non compris par le parseur ou pour des erreurs de syntaxe.

- L'export de la surface au format X3D n'a pas pu être décrit à l'aide des indices et des positions des vertex. La solution mise en place pour pallier ce problème est de décrire la surface à l'aide de boites. Chaque boite correspond à un cube de la surface, ce qui fait que la structure X3D est extrêmement lourde et certains logiciels 3D mettent beaucoup de temps pour l'ouvrir.

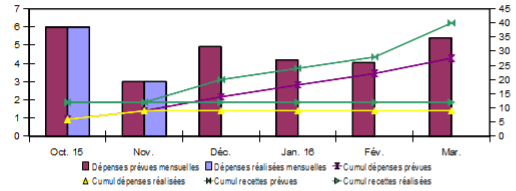
## 4.5 Perspectives

Des évolutions possibles de l'application sont d'ajouter la gestion des courbes de révolutions dessinées à la souris et l'amélioration des points soulevés précédemment. Ces améliorations passent par une optimisation du rendu 3D pour faciliter l'utilisation de l'application pour les machines avec une petite configuration, une modification de la gestion des courbes permettant alors un plus large ensemble de courbes disponibles, plus de paramètres liés aux courbes pour plus d'interactions, une meilleur communication de possibles erreurs avec l'utilisateur et une nouvelle gestion de l'export au format X3D voire un ajout d'export vers d'autres formats 3D.

# 5 Annexes

| N° projet | Chef de projet (Cdp) | Reporting projet | Libellé du projet | | | | | N° d'identification |
|---|---|---|---|---|---|---|---|---|
| | T Benoist | | Surface de révolution discrète | | | | | D060-QG8-rév0 |
| | Cahier des charges projet | Dir. du Cdp | Fréquence Reporting | | | Nom Fichier | Précédent / Mise à jour | Niv. Confid. / Edition |
| | | | | | | RP.xls | | 2 / 17/03/2016 14:15 |

*Evolution des dépenses et des recettes (k€)*




*Principaux résultats / livrables / faits marquants de la période et prochaines étapes*

Problèmes avec les fonctions implicites

Bibliothèque ne remplit pas les fonctionnalités attendues

Ajout de nouveaux clients

Prise de retard sur la conception

Gain de temps sur une tâche (étude du code)

| Reporting tâches-clés de la période | Fin prévue | Taux d'avct | Fin actual. |
|---|---|---|---|
| 2.1 – Etude du code Projet bifurcation | 17/11/15 | 100,00% | 17/11/15 |
| 2.2 – Conception de l'architecture | 14/12/15 | 50,00% | 16/12/15 |
| | | | |
| | | | |
| | | | |
| Global Projet | | | |

| Points positifs de la période | Préoccupations de la période (dont risques) |
|---|---|
| Conception a soulevé des problèmes qu'on aurait eu plus tard. Mise en œuvre de solutions<br>Evolution du projet => Ajout d'Aurélie en tant que Cliente, toujours à définir, mais sûr<br><br>Gain de temps sur la tâche Etude du code | Augmentation de la durée de la conception<br>Problèmes avec les fonctions implicites (risque 09 avéré)<br>Ajout de nouveaux clients, Aurélie Mourier et Labo (risque 10 avéré)<br>Problème de locaux pendant la pause pédagogique |

| Actions à mettre en œuvre | Libellé de l'action | Qui | Quand |
|---|---|---|---|
| | Modifier le temps alloué à la tache optionnelle (11 : Ajout de courbes utilisateur) pour en rajouter sur la tâche conception | Chef de projet | 14/12/15 |
| | Organiser une réunion avec les clients pour discuter des nouveautés qu'implique leur participation au projet | Chef de projet | avant le 14/12/15 |
| | Reconsidérer la numérotation des tâches ; corriger gantt, avancement, ... | Tout le groupe | 03/12/15 |
| | Mettre en place une solution pour travailler ensemble durant cette période (voir avec la BU, le secrétariat, les salle de réunion du labo) | Tout le groupe | avant le 21/12/15 |
| | Planifier les réunions de suivi ; prévoir la prochaine et rythme (au moins une par mois) | Chef de projet | 03/12/15 |
| | Discrétiser la courbe de progression technique ; enrichir la courbe d'annotations | Adrien | 03/12/15 |

Figure 1: Reporting de la réunion de suivi

| N° | Thèmes | Description | Gravité | Probabilité | Criticité |
|---|---|---|---|---|---|
| 1 | Réalisation | Non accès aux programmes informatiques extérieurs nécessaires | 3 | 0 | 0 |
| 2 | Performance | Erreur de conception | 2 | 0 | 0 |
| 3 | Performance | Performance insuffisante du produit développé | 1 | 2 | 1 |
| 4 | Performance | Détection de bugs (logiciel) | 1 | 1 | 1 |
| 5 | Performance | Non respect du cahier des charges | 1 | 2 | 1 |
| 6 | Performance | La validation met en évidence un grave problème technique | 2 | 1 | 1 |
| 7 | Moyens techniques | Panne du réseau électrique | 1 | 1 | 1 |
| 8 | Moyens techniques | Panne ou dysfonctionnement des appareils | 1 | 1 | 1 |
| 9 | Moyens techniques | Non adéquation d'un outil prévu, matériel ou logiciel (IFP ou extérieur) | 2 | 1 | 1 |
| 10 | Client / Partenaire (Cl./P.) | Nouveau Cl./P. : prudence | 1 | 2 | 1 |
| 11 | Partenaire | Mauvaise définition des rôles et responsabilités respectifs | 2 | 1 | 1 |
| 12 | Contractuel | Lacune au niveau des clauses de responsabilité | 2 | 1 | 1 |
| 13 | Financier | Mauvaise évaluation des besoins (moyens, coûts, délais) | 1 | 2 | 1 |
| 14 | Humain / Organisation | Démission du CDD / stagiaire | 2 | 0 | 0 |
| 15 | Humain / Organisation | Fiabilité de l'estimation des moyens nécessaires | 1 | 1 | 1 |
| 16 | Humain / Organisation | Information interne insuffisante conduisant à une non-anticipation de divisions ou directions actrices du projet | 2 | 1 | 1 |
| 17 | Humain / Organisation | Non prise en compte des risques dans l'évaluation de la durée des différentes tâches | 2 | 0 | 0 |
| 18 | Performance | Rendu 3D demandant trop de ressources | 2 | 1 | 1 |
| 19 | Réalisation | Interface à réaliser pour deux catégories d'utilisateurs | 2 | 1 | 1 |
| 20 | Réalisation / Délai | Évolution de l'algorithme de génération | 1 | 3 | 2 |
| 21 | Réalisation | Transcription de l'algorithme difficile | 2 | 1 | 1 |
| 22 | Moyens techniques | Problème lié au serveur | 1 | 0 | 0 |

| Légende des risques | |
|---|---|
| | Très faible |
| | Moyen |
| | Moyen mais plus probable |
| | Moyen mais plus grave |
| | Majeur |

Figure 2: Liste complète des risques