Projet : Génération de dégradations physiques sur les modèles 3D

Rapport de projet

Tuteurs de projet :
- Christophe Renaud
- François Rousselle

Remerciements

Je souhaite avant tout remercier M. Rousselle et M. Renaud, qui ont accepté que je travaille avec eux sur ce projet, et qui m'ont beaucoup aidé à sa réalisation. Je remercie aussi M. Verel qui, avec mes collègues, nous a donné de nombreux conseils quant à la réalisation de ce rapport, autant sur le fond que sur la forme.

Table des matières

1 - Positionnement du projet

1.1 Rappel : Contexte et définition du problème

Dans le cadre de reconstitutions virtuelles historiques, le challenge de l'équipe pluridisciplinaire réunie (informaticiens et historiens) consiste à s'approcher au plus près de la réalité historique de l'environnement reconstruit. Au travers de recherches approfondies dans divers fonds d'archives, les historiens apportent ainsi des connaissances fondamentales à la reconstruction 3D des environnements ciblés, permettant ensuite une visualisation plus ou moins réaliste et/ou plus ou moins interactive de ces environnements, pour des applications diverses à destination des chercheurs ou du grand public..

Un défaut fréquent de ces reconstitutions réside dans le fait qu'ils apparaissent comme neufs, ce qui est clairement préjudiciable à leur interprétation dans leur contexte historique. Un travail complémentaire, effectué par des infographistes, consiste alors à introduire des dégradations dans ces environnements, provenant tant de causes naturelles (vent, pluie, etc.) que humaines (usures dues aux passages répétés, chocs, déprédations volontaires, etc.).

L'objectif est de se focaliser sur un type particulier de dégradation, lié à l'usure et aux chocs reçus par les pierres qui apparaissent dans le bâti reconstruit, qu'il s'agisse de marches d'escalier ou des pierres des murs. Le travail à réaliser consistera à développer le code d'une méthode existante et à tester son utilisation dans le cadre d'une reconstitution du Pont Notre Dame en 1720.

1.2 Encadrants

Je suis encadré pendant ce projet par M. Christophe Renaud, directeur du LISIC et responsable de l'équipe IMAP (Images et Apprentissage) du laboratoire de recherche, et par M. Rousselle, maître de conférences, enseignant-chercheur à l'IUT de Calais-Boulogne, et membre de l'équipe IMAP du laboratoire de recherche.

1.3 L'état initial

Le pont Notre Dame a déjà été intégralement modélisé sous 3DsMax par M. Rousselle, et m'a été fourni au format Wavefront obj à des fins de tests.

1.4 Durée du projet

La période libre de projet s'étale du 18 avril au 05 juin 2016. Toutefois, comme mentionné par M. Fabien Teytaud, responsable du Master 1 Informatique ISIDIS, il est fort recommandé de commencer le projet dès le mois de janvier.

1.5 Les outils mis en place

Une boite de dépôt github a été créée pour l'occasion, à l'adresse suivante : https ://github.com/guil-prin/notreDame . Régulièrement mise à jour, vous pouvez y trouver l'évolution du développement (dossier "dev"), les documents produits pour le projet (dont ce rapport, dans le dossier "documentation"), et les sources finales (dossier "sources"). Les descriptions fonctionnelles du projet sont d'ailleurs dans le cahier des charges fourni le 25 avril.

Un disque dur avec un système d'exploitation Debian vierge installé m'a été gracieusement prêté par Nathalie Ramat, responsable du parc informatique de l'ULCO, afin que je puisse bénéficier du statut d'administrateur sur mon pc, et que je puisse donc installer les outils nécessaires au développement du projet.

Une bibliothèque c++ a aussi été mise en place : CGAL. Disponible sur le site http ://www.cgal.org/, elle permet de travailler sur une structure 3D avec des outils fournis pour définir un objet 3D et ses composants (arêtes, sommets, faces). La recherche de cette bibliothèque a été une partie du travail préliminaire au projet.

2 - Travaux préliminaires

2.1 Étude documentaire

Deux notes de recherche m'ont été confiés par M. Renaud lors de notre premier entretien :

— Surface Aging by Impacts [PPD01]

— Modeling cracks and fractures [DGA05]

Il m'a aussi expliqué le but du projet : le modèle 3D déjà créé est terminé, mais il s'avère être "trop beau" par rapport à la réalité. En effet, les bâtiments modélisés s'avèrent en réalité avoir environ une centaine d'années d'après les historiens avec qui ils travaillent. Le but est donc de vieillir le modèle 3D afin de le prendre plus proche de la réalité.

Il m'a donc été demandé à ce moment de consulter ces papiers de recherche, et plus particulièrement Surface Aging by Impacts [PPD01]. J'ai eu plusieurs réunions avec M. Rousselle pour discuter de ce papier en détail. J'ai eu, à la demande de ce dernier, à réaliser une synthèse personnelle de ce papier afin de me faire comprendre la démarche d'un document scientifique (un cours d'initiation à la recherche a été enseigné par M. Verel à ce propos, mais n'a eu lieu que après cette réunion). Cette synthèse est disponible dans le dossier documentation du projet.

Une troisième note de recherche me sera fournie au début de la période de projet, Real-Time Relief Mapping on Arbitrary Polygonal Surfaces [POC05]. Bien que consultée, elle ne sera pas intéressante pour cette période de projet.

2.2 Recherche de bibliothèques

Au cours de ces entretiens, il m'a aussi été demandé de rechercher des bibliothèques c++ afin d'éviter à devoir tout coder. En effet, la modélisation 3D est un domaine extrêmement en vogue à l'heure actuelle (que ce soit pour des simulations historiques, des modélisations de prototypes, des créations de jeux vidéos, ...), et de nombreuses personnes ont déjà travaillé dans ce domaine. Deux bibliothèques ont ainsi été trouvées par mes soins : les bibliothèques Assimp (http ://www.assimp.org/) et Magnum (http ://mosra.cz/blog/magnum.php), don la dernière n'a pas retenu notre attention. par la suite, M. Rousselle s'est rappelé d'un projet graphique réalisé quelques années auparavant par deux

anciens étudiants, et ils avaient utilisé la bibliothèque CGAL. Cette dernière répondant à nos besoins, elle a été gardée pour le développement, tout en mettant de côté Assimp s'il s'avère que CGAL montre des limites.

3 - Recherche et Développement

Evoquer papiers + photos pour illustrer le but recherché (photo réelle, synthèse, ...)

3.1 Comment vieillir des pierres ?

L'idée générale du vieillissement dans ce projet est de réaliser de nombreux petits impacts sur la surface à vieillir, afin que le cumul de ces impacts donne une sensation d'usure sur l'objet 3D. Surface Aging by Impacts [PPD01] explique en détail une théorie de l'idée. Dans le cadre de ce projet, les étapes de la réalisation d'un impact sont les suivantes :

Algorithm 1 Réaliser un impact

```
1: Charger l'objet 3D sur lequel travailler
2: repeat
3:    repeat
4:       Chercher la/les face(s) sur laquelle/lesquelles le point d'impact aura lieu
5:       if le point est sur une face then
6:          Affiner le maillage de la face qui sera impactée
7:       else if le point est sur une arête then
8:          Affiner le maillage de l'arête et des faces adjacentes qui seront impactées
9:       else if le point est sur un sommet then
10:         Affiner le maillage des arêtes et des faces adjacentes qui seront impactées
11:      else
12:         Arrêter le processus
13:      end if
14:   until surface de la face à impacter en dessous d'un certain seuil
15:   Réaliser l'impact
16: until impact souhaité obtenu
17: Enregistrer l'objet 3D réalisé
```

Annoncer plan par la suite par rapport à l'algo

3.2 Lire et écrire un objet 3D

3.2.1 Structure d'un objet 3D au format OBJ

Définition (Wikipédia)

OBJ est un format de fichier contenant la description d'une géométrie 3D. Il a été défini par la société Wavefront Technologies dans le cadre du développement de son logiciel d'animation Advanced Visualizer. Ce format de fichier est ouvert

et a été adopté par d'autres logiciels 3D pour des traitements d'import / export de données.

Pourquoi le format Wavefront OBJ ?

Le pont Notre Dame a été entièrement modélisé sous Unreal Engine, un moteur de jeu vidéo développé par la société Epic Games. Le format des objets créés étant propriétaires, je ne pouvais pas travailler directement dessus. Cependant, Unreal Engine propose un export des objets créés dans divers formats libres, dont le format OBJ, facile à comprendre et sur lequel il est aisé de travailler.

Structure d'un obj

Un objet 3D au format obj est un ensemble de sommets avec leurs coordonnées, et un ensemble de faces avec les sommets composants ces faces. Un sommet (ou vertice) se définit en commençant une ligne par la lettre v, et est suivi des coordonnées du point. Par exemple, les sommets d'un cube simple seraient :

```
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000000
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
```

Chaque sommet défini dans un fichier .obj est implicitement numéroté, en commençant à l'indice 1.

On peut aussi de manière facultative (mais conseillé) d'indiquer les coordonnées de texture (la ligne commence ainsi par vt) et les coordonnées des normales (la ligne commence ainsi par vn).

Ensuite, les faces (ou facets) sont définies en commençant chaque ligne par la lettre f, suivie d'une suite de numéros des sommets — définis précédemment dans le fichier — qui composent la face. Pour ce cube, avec des faces de type quadrilatère, elles seraient :

```
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
```

Enfin, un fichier obj peut être composé de plusieurs objets. Chaque objet est ainsi délimité par un nom, avant la déclaration du premier sommet. Le nom peut commencer par la lettre o ou par la lettre g, selon si on veut définir un objet simple ou un groupe. Certains logiciels 3D choisiront d'utiliser malgré tout la lettre g pour définir un objet simple.

3.2.2 Structure d'un objet 3D sous CGAL

Un objet 3D est défini via la bibliothèque CGAL comme un Polyhedron_3 composé de faces, arêtes, sommets, et de relations entre ces composants. Chacune de ses arêtes est composé de deux demi-arêtes (halfedge), ayant chacun un sens opposé à l'autre. Les relations entre ces composants peut se définir de la manière comme représenté dans la Figure 1.

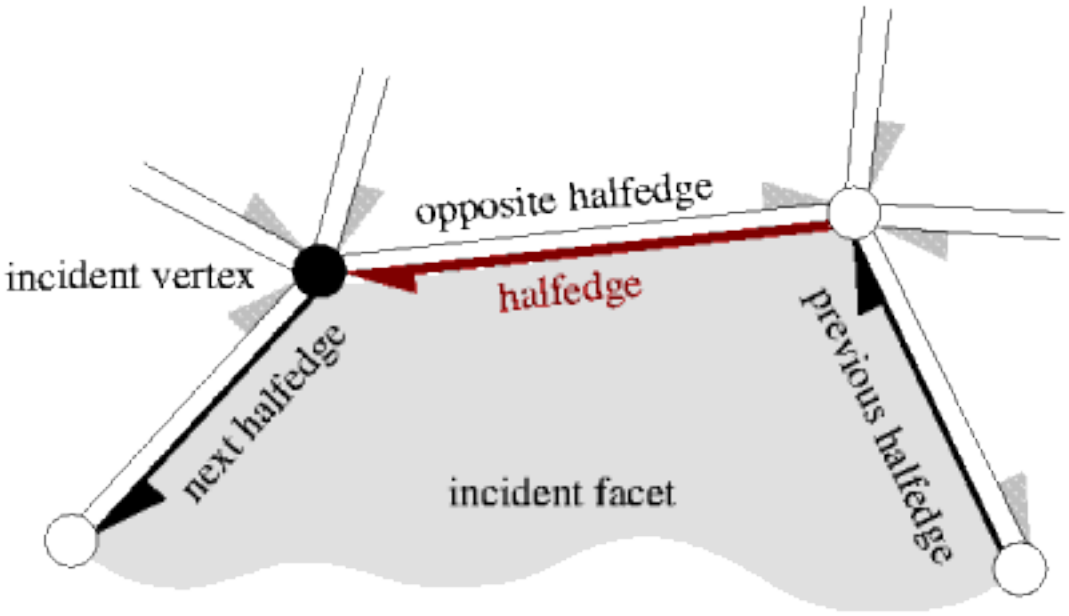

FIGURE 3.1 – Structure d'un Polyhedron_3 - Source : http ://www.cgal.org

Il est ainsi possible d'accéder à partir d'un composant à tous les autres composants adjacents. Par exemple, halfedge()->facet() permet d'accéder dans cette figure à la face qui contient la demi-arête en rouge.

3.2.3 Chargement d'un obj

Le premier obstacle a été de gérer le chargement d'un fichier de format obj et de le transcrire comme un Polyhedron_3. En effet, CGAL ne gère pas nativement cette option, ayant préféré le format .off. Cependant, un développeur nommé James Greggson a publié sur son blog (http ://jamesgregson.blogspot.fr) en mai 2012 une version de loader obj et transcrit en Polyhedron_3. J'ai donc repris son code et l'ait amélioré pour gérer en plus la subdivision en plusieurs Polyhedrons s'il s'avère qu'un fichier .obj contenait plusieurs objets en eux-même (la version initiale du développeur chargeait tous les objets d'un fichier obj en un seul

Polyhedron_3).

De manière simple, CGAL peut incrémentalement créer un Polyhedron_3, en déclarant un nouveau polyèdre par la fonction begin_surface(nombre de sommets, nombre de faces). Ensuite, on ajoute chaque sommet avec la fonction add_vertex(objet de type Point(x, y, z)). Enfin, on créé les faces du Polyhedron_3. Pour chaque face à créer, on déclare la nouvelle face (avec begin_facet()), on ajoute un à un les numéros de sommets dans l'ordre du fichier obj (avec add_vertex_to_facet(num de la face)), puis on signale que la face à créer est terminée (avec end_facet()), ce qui va relier le dernier sommet au premier. Il faut juste faire attention au fait que le format obj commence à l'indice 1, mais c++, comme de nombreux langages de développement, commence à l'indice 0.

3.2.4 Exporter un obj

De la même manière, il m'a fallu récupérer le fichier obj une fois le traitement du Polyhedron_3 terminé. Étrangement, là où CGAL n'a pas prévu de manière native un chargeur obj, il a prévu un enregistreur, avec la fonction print_polyhedron_wavefront(). Il suffit juste d'exporter les polyèdres un par un dans un même fichier (via un ofstream), et on récupère ainsi un fichier obj prêt à l'emploi (par exemple pour le consulter via un logiciel adapté comme Blender).

3.3 Mailler un Polyhedron_3

3.3.1 Maillage initial

Le maillage d'un Polyhedron_3 est constitué de ses faces. Ainsi, si un objet de type cube est importé avec un maillage triangulaire, il sera constitué de 12 faces, avec 2 faces dans le même plan pour chaque véritable face du cube.

3.3.2 Chercher la ou les faces du point d'impact

Comme indiqué dans l'algorithme 1, selon où se trouve le point d'impact, il faut agir différemment. En effet, si le point est sur une arête, alors il faut modifier le maillage des faces adjacentes afin de réaliser un impact sans modifier drastiquement la structure de l'objet. CGAL fournit toutefois un itérateur de faces, nommé Facet_iterator, qui parcourt toutes les faces de l'objet 3D. Les faces de la scène 3D du pont Notre Dame étant toutes des triangles, il est possible de reconstruire un triangle avec pour coordonnées les points de la face. A partir de là, on vérifie si le point d'impact est sur le triangle. Si oui, on l'ajoute à un vecteur de faces. Sinon, on l'ignore. On reprend le processus jusqu'à la dernière face. Si le point d'impact est sur une arête, alors les deux faces qui composent l'arête seront détectées et ajoutées dans le vecteur de faces. La même chose sera appliquée pour un sommet, avec 3 faces ou plus d'ajoutées.

3.3.3 Cas d'un point sur une face

Subdiviser la face

De nombreuses recherches et tentatives ont été effectuées pour subdiviser une face.

Tout d'abord, j'ai commencé ma subdivision en effectuant une division barycentrique d'une face. Cette subdivision se fait en deux étapes. CGAL propose une fonction permettant de subdiviser une face du polyèdre, nommée create_center_vertex(pointeur d'halfedge de la face à subdiviser). Cette fonction relie tous les points de la face (Fig 2.1, halfedge de la face en rouge) vers un nouveau point créé de la face, qui sera placé au même endroit que le point pointé par l'halfedge initial, et retourne un Halfedge_handle (un pointeur d'halfedge) partant de l'origine du halfedge passé en paramètre et pointant vers le nouveau point (Fig 2.2, l'halfedge retourné en bleu). A partir de là, on peut modifier la valeur du point du halfedge retourné en accédant au point (HalfedgeRetourné->vertex()->point()), et on le calcule en prenant la moyenne en x, y, et z, des trois points du triangle (Fig 2.3). Le problème majeur provient du fait que les triangles ainsi créés sont allongés par rapport au triangle original, et qu'à force de subdivision, les triangles seront de plus en plus allongés. Les impacts seront donc beaucoup trop étalés sur un côté, ce qui ne donnera pas un résultat sympathique.

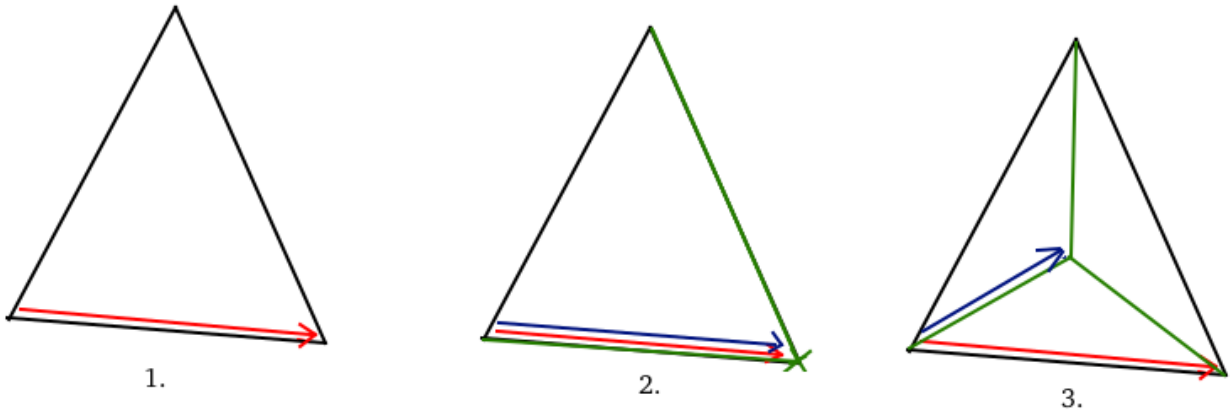

FIGURE 3.2 – Subdiviser une face

Ensuite je me suis tourné, avec l'avis de M. Rousselle, vers la triangulation de Delaunay, afin de pallier au problème des triangles allongés. Il est possible de réaliser cette triangulation via CGAL, mais pas avec un Polyhedron_3, mais avec un MeshDomain_3. Toutefois, par manque de temps, je n'ai pas pu approfondir les recherches à ce niveau. CGAL peut certainement proposer un moyen de convertir de l'un vers l'autre et réciproquement, auquel cas ce travail serait possible. De nouvelles recherches à ce niveau seront effectuées dans le cadre de mon stage afin d'améliorer la subdivision autour d'un point.

J'ai toutefois trouvé un moyen de contourner le problème des triangles al-

FIGURE 3.3 – Impact sur une face d'un cube en 3 points

longés. En effet, si chaque sous-triangle d'une face est elle-même divisée en 2, alors les triangles crées seront beaucoup moins allongés, car plus nombreux (6 au lieu de 3). CGAL prévoit aussi une fonction de division d'arêtes, nommée split_edge(pointeur d'halfedge à subdiviser), qui fonctionne de la même manière que la fonction précédente. On créé avec la fonction le point qui sert de subdivision, puis on replace le point là où on le souhaite, dans mon cas, au milieu de l'arête. Le souci majeur de cette fonction est que le point n'est pas relié aux points des faces qui le relient. Mais ce problème est vite réglé, car en faisant l'appel à split_edge() puis à create_center_vertex(), le point central créé va se relier aux sommets du triangle, mais aussi aux points créés sur les arêtes de ce triangle. Ainsi, au lieu de subdiviser une face en 3 triangles, on la subdivise en six, réduisant l'étirement des triangles (Figure 4).

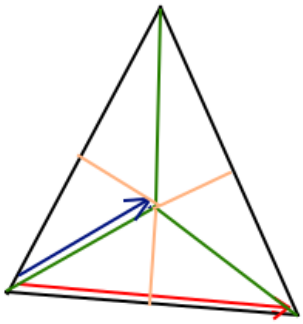

FIGURE 3.4 – Subdiviser une face en six points

Supprimer les T-vertices

La dernière méthode de subdivision présentée pose un très gros problème : la formation d'arêtes en T. Concrètement, dans le cas de la figure 3, on peut voir que les 3 points au centre des arêtes créés forment un T. De nombreux algo-

FIGURE 3.5 – Impact sur une face d'un cube en 6 points

rithmes de traitement de lumière, entre autres, n'arrivent pas à gérer correctement ces cas, générant des artefacts désagréables. Le but est donc de prolonger la subdivision aux points d'arêtes afin d'éviter ces T vertices.

Une méthode simple appliquée dans mon cas a simplement été de rechercher les faces adjacentes, et de réaliser à leur tour une subdivision barycentrique sans subdiviser les arêtes qui ne le sont pas déjà. Ainsi, les arêtes subdivisées génèrent une arête de prolongation, et les autres ne généreront pas de T-vertices à leur tour.

Je rappelle que la structure du Polyhedron_3 proposé par CGAL identifie les arêtes non pas comme des edges, mais comme des halfedges. Ainsi, l'halfedge opposé à celui de la face en cours (halfedge->opposite()) concerne la même arête, mais cette fois, concerne la face de l'autre côté de l'arête, qui est donc adjacente à la face en cours de traitement. Rappeler la méthode de subdivision de faces, sur la face adjacente donc, permet de subdiviser la face adjacente, et donc de prolonger le point créé sur l'arête lors de la subdivision initiale.

Vérifier la taille de la face à impacter

Il faut maintenant vérifier si on doit raffiner le maillage ou non. Pour cela, on cherche la nouvelle face qui contient le point d'impact, on calcule la distance entre un point de la face trouvée et le point d'impact. Si la distance est supérieure à un certain seuil, on recommence les deux premières étapes. Sinon, on passe à la dernière étape. Cette vérification est en fait un appel récursif des deux premières étapes.

Impacter la face

Cette étape est en réalité la plus simple. En effet, nous avons déjà récupéré la face concernée par l'impact. Il suffit juste donc de créer un point au milieu de cette face, puis de déplacer le point créé de telle sorte qu'il créé une sorte de creux dans la face.

En l'état, l'impact est créé "en dur", à savoir que je rentre moi même les coordonnées pour modifier l'objet. Toutefois, il est prévu par la suite, et conformément au papier de recherche Surface Aging by Impacts, de calculer la normale de la face qui sera impactée, et de déplacer le point créé dans le sens opposé à cette normale. Néanmoins, l'impact reste relativement étiré même avec la division en six sous-triangles. La triangulation de Delaunay reste une solution optimale à ce problème, et devra donc être traité le plus rapidement possible.

3.3.4 Cas d'un point sur une arête ou sur un sommet

N'ayant pas pu réaliser cette tâche, ceci ne concernera que la partie théorique.

L'idée est qu'en cas d'impact sur une arête, on puisse casser l'arête et créer un creux dans cette arête. Il faut donc :

1. Casser l'arête ciblée avec la fonction split_edge()
2. Casser les deux arêtes ainsi créées à proximité du point d'impact
3. Utiliser la fonction create_center_vertex() sur les deux faces adjacentes et placer le point central à proximité du point d'impact (trouver un point à proximité est la raison pour laquelle cette section n'a pas été développée, car je n'ai pas encore trouvé de méthode optimale pour réaliser cette tâche)
4. déplacer le point d'impact afin de créer un petit creux qui ne déforme pas tout l'objet, d'où la présence des 4 autres points.

De la même manière, pour un sommet, il suffit de réaliser les même tâches, sans la première, car on déplacera directement le point "sommet" au lieu d'en créer un autre.

4 - Perspectives

Par la suite, il est prévu de travailler sur les tâches suivantes :

— Réaliser un impact sur une arête, puis sur un sommet, avec la méthode explicitée dans la section précédente.

— Permettre de réaliser un impact à partir d'un vecteur de frappe. Le point d'impact devra donc être calculé par l'intersection du vecteur généré et du polyèdre.

— Améliorer le sous-maillage, non plus via une triangulation barycentrique, mais via la triangulation de Delaunay, afin de réduire l'étirement des triangles générés.

5 - Conclusion

Dans le cadre de ma période de projet, il m'a été demandé de réaliser un module de vieillissement d'un objet 3D, en simulant des impacts. Après diverses recherches, j'ai travaillé sur ce module en utilisant la bibliothèque CGAL, qui proposait l'objet Polyhedron_3.

J'ai ainsi pu réaliser une première version d'impact sur une face d'un objet, en affinant le maillage de l'objet 3D puis en ajoutant un point à proximité du point d'impact. Cependant, l'impact généré s'avère être trop étiré par rapport à la réalité, et l'affinage du maillage nécessite donc d'être retravaillé.

Il reste encore du travail à faire après celà. En effet, les impacts peuvent avoir lieu sur les arêtes et sur les sommets d'un objet, auquel cas l'algorithme de travail devra être différent.

Ces travaux complémentaires seront réalisés à l'occasion d'un stage en fin de Master 1.

Bibliographie

[DGA05] Brett Desbenoit, Eric Galin, and Samir Akkouche. Modeling cracks and fractures. *The Visual Computer*, 21(8-10) :717–726, 2005.

[POC05] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 155–162, New York, NY, USA, 2005. ACM.

[PPD01] Eric Paquette, Pierre Poulin, and George Drettakis. Surface aging by impacts. In *Proceedings of Graphics Interface*, June 2001.