

Bachelor en Informatique

Orientation : Développement logiciel et multimédia

Projet n°226 DataCube : Courbes implicites

Réalisé par
Mathias Marty

Encadrement : Prof.Dr. Gobron Stéphane
HE-Arc, HES-SO

Expert : Prof.Dr. Widmer Antoine
HEVS, HES-SO

Résumé

Avec la démocratisation de l'accès aux données de l'observation terrestre issues des satellites, il est nécessaire d'avoir des outils adaptés permettant de visualiser ces données. Elles sont principalement représentées par des matrices de plusieurs dimensions appelées *datacube*. Dans le cadre de ce projet, des rendus de ces données seront effectués à travers des *notebook* Jupyter tout en utilisant l'Application Programming Interface (API, Interface de programmation) WebGL. La combinaison de ces deux technologies permet d'avoir un rendu performant sur un outil utilisé par les scientifiques. En particulier, nous allons nous concentrer sur l'affichage et l'extraction des courbes de niveau des *datacube* à deux dimensions. Pour se faire, un algorithme de calcul des courbes de niveau a été développé ainsi que deux modèles de rendu. Le premier modèle de rendu permet d'afficher les courbes de niveau sur un plan que l'on peut bouger dans l'espace et le deuxième, de projeter les courbes sur une surface à trois dimensions où la hauteur de cette dernière représente l'intensité du *datacube* en tout point. En plus des modèles de rendu, des méthodes permettant de transformer les courbes en équations paramétriques ont aussi été développées. Ces équations permettent la reconstruction de courbe qui approxime les courbes de niveau à partir de seulement quelques points caractéristiques. Cela nous permet de réduire l'information nécessaire à l'enregistrement des courbes après leur extraction.

Abstract

With the democratization of access to Earth observation data from satellites, it is necessary to have adapted tools to visualize these data. They are mainly represented by multidimensional matrices called *datacube*. In this project, renderings of these data will be done through Jupyter *notebooks* while using the WebGL API. The combination of these two technologies allows to have a powerful rendering on a tool used by scientists. In particular, we will focus on the display and extraction of the two-dimensional *datacube* contours. To do this, a contouring algorithm has been developed as well as two rendering models. The first rendering model displays the contours on a plane that can be moved in space and the second one projects the contours on a three-dimensional surface where the height of the surface represents the intensity of the *datacube* at any point. In addition to the rendering models, methods to transform the curves into parametric equations have also been developed. These equations allow the reconstruction of curves that approximate the contour lines from only a few characteristic points. This allows us to reduce the information needed to record the curves after their extraction.

Mots-clés— Computer Graphics, Earth observation, Data visualization, Geometric modeling, Data cube, WorldWind, Web WorldWind, WebGL, GLSL, 3D Rendering, Real-time render, Image processing, Jupyter, Python, Aerospace, Surface, Forecasting, Climate change, Agricultural forecasting, Performance, Thresholding, Implicit curve, Marching-square, Marching-cube, Contour line, Field vectorization.

Remerciements

La réalisation de ce travail de Bachelor a été possible par la collaboration de plusieurs personnes que je souhaite remercier. J'aimerai tout d'abord remercier mon superviseur, Monsieur Stéphane Gobron, pour avoir suivi assidûment le développement du projet et pour les précieux conseils qu'il m'a fournis.

Je tiens aussi à remercier Monsieur Artan Sadiku qui a pu développer des outils qui m'ont fait gagner beaucoup de temps et qui m'a également conseillé sur la rédaction de ce rapport.

Je voudrais exprimer ma reconnaissance envers les membres de l'entreprise partenaire Solenix : Monsieur Yann Voumard et Monsieur Joep Neijt, pour l'accès aux données satellites et surtout pour leur collaboration dans le cadre de ce projet.

Je remercie l'expert, Monsieur Antoine Widmer, notamment pour la relecture de ce document.
Enfin, je souhaite à particulièrement remercier mon collègue Monsieur Antoine Lestrade, pour les nombreuses heures d'entraide et les relectures mutuelles de nos manuscrits respectifs.

Table des matières

Résumé	i
Abstract	iii
Remerciements	v
1 Introduction	7
1.1 Contexte	7
1.2 Twin bachelor	7
1.3 Etat de l'art	8
2 Données issues de l'EO	17
2.1 Regroupement	17
2.2 Chargement	17
3 Construction des courbes implicites	19
3.1 Modèle	19
3.2 Implémentation	21
3.3 Résultats	23
4 Paramétrisation des courbes implicites	25
4.1 Modèle	25
4.2 Implémentation	28
4.3 Résultats	29
5 Approximation polygonale d'une courbe discrète	31
5.1 Modèle	31
5.2 Implémentation	34
5.3 Résultats	35
6 Reconstruction des courbes discrètes	37
6.1 Modèle	37
6.2 Implémentation	37
6.3 Résultats	37
7 Modèles de rendu	39
7.1 Algorithme de <i>raycasting</i>	39
7.2 Rendu des courbes implicites	40
7.3 Rendu de la surface	40
7.4 Rendu des courbes sur une surface	43
7.5 Palette de couleur	44
8 Tests et résultats	47
8.1 Comparaison des modèles de rendu avec des données de l'EO	47
8.2 Comparaison des méthodes de paramétrisations	48

9 Discussion	53
9.1 Conclusion	53
9.2 Perspsective	53
A Démonstration du majorant de la dérivée de la série partielle de Fourier	V
B Palette de couleur	VII
C Erreurs sur les images spécifiques	IX
D Planning	XI

Acronymes

API Application programming interface (Interface de programmation). i, iii, 7, 13, 15–17, 39, *Glossaire* : API

CNES Centre national d'études spatiales . 9

CT Computerized tomography (Tomodensitométrie). 13, 14, 55

DFT Discrete fourier transform (Transformation de Fourier discrète). 25–27, 29, 38, 50

DLR Deutsches zentrum für luft- und raumfahrt (Le Centre allemand pour l'aéronautique et l'astronautique). 9

EO Earth observation (Observation de la Terre). 7–9, 11–15, 47, 48, 55, 56

ESA European space agency (Agence Spatiale Européenne). 7–10, 55

FAPAR Fraction of absorbed photosynthetically active radiation (Fraction de rayonnement solaire absorbée par les plantes). 47

GES Gaz à effet de serre . 12

GLSL Opengl shading language . 21, 39, 42, *Glossaire* : GLSL

GMT Generic mapping tools . 44

GPGPU General-purpose processing on graphics processing units (Calcul générique sur processeur graphique). 21, 22, *Glossaire* : GPGPU

GPU Graphics processing unit (Processeur graphique). 7, 21, 22, 37, 39–41, 46, 49, 53, *Glossaire* : GPU

HTML Hypertext markup language (Langage de balises pour l'hypertexte). 7

IRM Imagerie par résonance magnétique . 13

JS Javascript . 17, 37, 39, 45, 49, 53

JSON Javascript object notation . 44, 45

LAI Leaf area index (Indice de surface foliaire). 47

OpenGL Open graphics library . 13, 39, *Glossaire* : OpenGL

RGBA Red blue green alpha (Rouge Vert Bleu Alpha). 21

SDB Bathymétrie par satellite . 15

SDF Signed distance function (Fonction de distance signée). 41, 42

SDG Sustainable development goals (Objectifs de développement durable). 11, 12

SGBD Système de gestion de base de données . 9

Glossaire

API Ensemble de code qui offre à un logiciel la possibilité d'utiliser un service. i

Atoll Ensemble de rochers circulaire. 15

Bathymétrique Science de la mesures du relief et des profondeurs de l'océan. 14

Courbe de niveau Les courbes de niveau sont un cas spécifique des lignes de niveau, celui où la ligne forme une courbe. Elle est aussi appelée isocourbe. 7, 8, 19, 40, 43, 48

Datacube Un datacube est un tableau multidimensionnel de valeur. i, iii, 7–9, 13–17, 19, 23, 39–43, 47, 54, 55

Dysmorphose Anomalie de la forme d'une partie du corps. 26

Fragment shader Programme qui a pour but le calcul de la couleur d'un unique pixel. 39

GLSL Language de programmation de shaders dont la syntaxe est proche du C. 21

GPGPU Calcul générique sur processeur graphique. 21

GPU Unité de calcul qui est spécialisé dans le calcul d'image. 7

ModelView La matrice ModelView est la concaténation de la matrice de modèle et de matrice de vue. La matrice de vue définit la position de la caméra, tandis que la matrice de modèle définit la position des primitives à dessiner. 39

ModelViewProjection Cette matrice applique les mêmes transformations que la matrice ModelView et y rajoute les caractéristiques de la caméra comme le champ de vue, la projection et le plan de détourage. 39

OpenGL OpenGL est une API qui permet d'afficher des images 2D et 3D. 13

OpenGL ES Une spécification qui définit une API dérivée d'OpenGL qui permet de générer des images 3D. 39

Raycasting Le raycasting est aussi appelé « Lancé de rayon ». Représentation simplifiée où on ne s'occupe pas des rebonds et des effets de l'environnement sur le rayon lancé. vii, 8, 13, 14, 39, 40, 42, 55

Shader C'est un programme qui sera lancé par le GPU. Une multitude de shaders sont lancés en parallèle pour l'affichage d'une image à l'écran. 7, 18, 21, 37, 39, 40, 49

Spline Fonction définie par morceaux par des polynômes. 25, 27–29, 50, 55

Surface de niveau Les courbes de niveau sont un cas spécifique des lignes de niveau, celui où la ligne forme une surface. Elle est aussi appelée isosurface. 39, 54

Variable uniforme Variable qui permet de passer une même valeur à tous les shaders. 39

Vertex shader Programme qui va calculer la position de chaque sommet de chaque primitive de la scène 3D. 39

Voxel Le voxel est à la 3D ce que le pixel est à la 2D. 14, 40, 42, 55

Web WorldWind API écrite en JS permettant d'utiliser WorldWind depuis le web. 9

WebGL WebGL est une API qui permet de développer d'utiliser OpenGL sur une page web. i, iii, 7, 15, 16, 18, 21, 22, 39, 45, 46, 49

WorldWind Logiciel d'exploration de planètes, grâce à des photographies satellites et aériennes, développé par l'Ames Research Center de la NASA dans un but pédagogique. 9

Chapitre 1

Introduction

1.1 Contexte

La visualisation des données est autant importante que les données elles-mêmes. De ce fait, la création d'outils puissants, ergonomiques et portables est indispensable à toute personne qui veut tirer des informations des données. Ces informations, quand elles sont discrétisées sont représentées sous la forme de matrices de tailles variables appelées *datacube*. Ces *datacube* sont notamment utilisés par les scientifiques. Par exemple, l'European Space Agency (ESA, Agence Spatiale Européenne) traite des *datacube* tirés de satellites dans le cadre de l'Earth Observation (EO, Observation de la Terre). L'Earth observation (EO, Observation de la Terre) couvre des champs d'applications variés tels que la recherche atmosphérique, l'impact du changement climatique, la détection d'incendies en forêt, le rendement agricole, etc. La problématique est de pouvoir disposer d'un outil simple et performant de visualisation de ces *datacube* en deux dimensions ou trois dimensions en temps réel. Pour cela, il a été imaginé de concevoir une application Web pour pallier cette problématique.

Le projet consiste à visualiser ces *datacube* en temps réel dans un canevas Hypertext Markup Language (HTML, Langage de balises pour l'hypertexte). Il utilise des techniques de rendus implicites en utilisant des programmes qui tournent sur le Graphics Processing Unit (GPU, Processeur graphique) moyennant l'API WebGL. Le tout est exécuté dans un *notebook* Jupyter, qui est un reconnu dans la communauté scientifique. Ces derniers sont appelés shader. L'objectif principal est l'affichage des courbes de niveau d'un *datacube* à deux dimensions. Deux rendus sont proposés. Le premier consiste à afficher ces courbes de niveau sur un plan que l'on peut bouger dans l'espace. Le second permet d'afficher les courbes sur une surface à trois dimensions où l'intensité du *datacube* en chaque point représente la hauteur de la surface. La deuxième méthode offre un rendu volumique des données, elle permet donc, une valorisation des données. En plus de l'aspect visualisation, nous avons développé un moyen d'extraire des caractéristiques d'une courbe afin de réduire la quantité d'informations nécessaire à sa reconstruction.

Ce travail de Bachelor fait partie d'un projet en collaboration avec Solenix. Solenix est une société privée, indépendante et active au niveau international, qui fournit des services d'ingénierie et de conseil sur le marché spatial. Ce projet vise la communauté scientifique avec comme cadre l'ESA et le Swiss Space Center. La charge de travail à fournir est à moitié répartie entre l'entreprise partenaire et la Haute Ecole Arc. Solenix nous fournit les données satellites en provenance du programme européen Copernicus ainsi que les services pour y accéder.

1.2 Twin bachelor

Il existe deux projets de Bachelor en parallèle pour ce sujet. Les deux partagent une même finalité : permettre la visualisation de données extraites de l'EO. Cependant, ils diffèrent sur la méthode et les techniques utilisées pour atteindre cet objectif. Le contexte, la problématique et les contacts restent identiques dans les deux projets ; par contre, les objectifs informatiques et leur champ d'application sont

différents, mais toutefois complémentaires.

Les différences majeures entre les deux projets sont les dimensions des *datacube* utilisés. L'un des projets s'attarde sur les *datacube* à deux dimensions alors que l'autre s'occupe de *datacube* à trois dimensions. Pour visualiser les *datacube* à trois dimensions, un rendu ressemblant à une vue aux rayons X et un autre permettant de voir l'évolution au fil du temps et de comparer un intervalle temporel des données avec un temps t définit sont menés par Monsieur Antoine Lestrade. Concernant les *datacube* à deux dimensions, la visualisation consiste en l'affichage des courbes de niveau sur un plan à deux dimensions et sur une surface à trois dimensions où l'intensité du *datacube* en chaque point représente la hauteur de la surface. Le projet de Monsieur Mathias Marty s'attarde aussi sur le développement de moyen permettant d'extraire, d'enregistrer tout en réduisant la quantité de données, puis de reconstruire des courbes de niveau. De ce fait, ce projet propose une approche plus abstraite que celui de Monsieur Antoine Lestrade. Ce dernier, est plus concret et plus exploratoire. La réalisation sera rapprochée et collaborative au début des projets, des séances hebdomadaires permettront d'avoir un suivi sur ceux-ci, mais aussi d'acquérir les bases théoriques communes tout en gardant une certaine cohésion.

Voici la liste des modèles de rendu utilisés :

1. *Raycasting* (X Ray) ;
2. *Raycasting* (dérivé) ;
3. *Raycasting* avec surface de niveau ;
4. *Raycasting* avec courbes de niveau sur une surface ;
5. Courbe implicite sur le plan.

Les quatre premiers modèles utilisent l'algorithme de lancer de rayon (*raycasting*). Les trois premiers modèles fonctionnent sur des *datacube* à trois dimensions alors que les deux derniers utilisent des *datacube* à deux dimensions. L'explication plus détaillée de ces deux derniers modèles se trouve au chapitre 7.

Les différents modèles de rendus sont attribués comme suit : Comme les modèles numérotés 1, 2 et 3 nécessitent des *datacube* à trois dimensions, Monsieur Antoine Lestrade s'occupe de leur développement. Les modèles 4 et 5 utilisent des *datacube* à deux dimensions, ils sont donc attribués à Monsieur Mathias Marty. La figure 1.1 résume les modèles de rendus utilisés, leurs attributions et une brève description de ces derniers.

Modèle	Description	Attribué à
Ray-cast	<i>Ray-cast version x-ray</i>	Antoine
Ray-cast derived	<i>Ray cast avec la dérivée</i>	Antoine
Ray-cast with implicit curve (3D)/Implicit curve (3D)	<i>Courbe implicite sur surface 3D</i>	Mathias
Implicit curve (2D)	<i>Courbe implicite</i>	Mathias
Implicit surface	<i>Ray-cast pour afficher les surfaces implicites</i>	Antoine

FIGURE 1.1 – Attribution des modèles de rendu et brève description de ceux-ci.

1.3 Etat de l'art

1.3.1 Observation terrestre

Avec la croissance des volumes de données acquises par l'EO et le besoin de les analyser dans plusieurs domaines, ces dernières sont devenues plus accessibles [1]. En juin 2005, l'introduction de Google Earth [2] a créé un déclic dans la démocratisation de l'accès aux données issues de l'EO au grand public [3].

La politique d'accès libre et gratuite aux données récoltées lors du programme Copernicus [4] de l'ESA permet à n'importe quel utilisateur d'accéder à des volumes de données issue de l'EO. Aujourd'hui, l'ESA a envoyé huit satellites [5] dans le cadre du programme Copernicus. EarthServer [6] est un projet

qui vise à établir une technologie client-serveur pour l'accès aux données issues de l'observation terrestre. Le Système de gestion de base de données (SGBD) *Rasdaman [7] permet l'utilisation de tableau multidimensionnel appelé *datacube*. En utilisant Rasdaman, EarthServer offre à ses clients la possibilité d'interagir avec les données en direct.

Des outils tels que Web WorldWind ou WorldWind [8], développés par la NASA, permettent de visualiser ces données sur un globe terrestre. D'autres satellites ont pour mission d'acquérir des informations sur la terre, voici une liste non-exhaustive de ceux-ci :

- Aeolus, le premier satellite à obtenir des données sur le vent à une échelle globale [9] ;
- Depuis 1986, Airbus a lancé plus de cinquante satellites destinés à l'EO [10] ;
- En 2023 l'ESA a prévu le lancement du satellite Biomas, celui-ci est conçu pour évaluer le niveau global de biomasse tropicale [11] ;
- Le Centre national d'études spatiales (CNES) et Deutsches Zentrum für Luft- und Raumfahrt (DLR, Le Centre allemand pour l'aéronautique et l'astronautique) lanceront le satellite Merlin (Methane Remote Sensing Lidar Mission) ou mission de télédétection du méthane par lidar en 2024 [12] ;
- Un satellite de mesure des échanges de dioxyde de carbone présent dans l'atmosphère : MicroCarb est une future mission de l'ESA. Son lancement est prévu en 2021 [13] ;
- En 2021 l'ESA a prévu le lancement du satellite EarthCARE. Son but est l'étude de l'impact des nuages et des aérosols dans le bilan radiatif de la terre [14].

La figure 1.2 présente des exemples de prise de vues d'une partie des satellites cités en dessus. La photo (a) a été prise par le satellite optique Pléiades Neo 3 d'Airbus. La photo (b) représente la lumière infrarouge émise par une éruption de l'Etna. Cette photo a été prise dans le cadre du programme Copernicus par le satellite Sentinel-2. La photo (c), aussi prise dans le cadre du programme Copernicus, mais cette fois-ci par le satellite Sentinel-1, représente l'évolution des zones sinistrées par l'inondation et par les feux de forêt en Australie entre le 7 et le 19 mars 2019. L'image (d) montre des relevés de la vitesse du vent par rapport à l'altitude. Ces mesures ont été prises par le satellite Aeolus de l'ESA. Les images de la figure en question ont été récupérées sur les sites de l'ESA et d'Airbus.

1.3.2 Datacube

Un *datacube*, ou cube de données en français, est un tableau multidimensionnel de valeurs [15]. Dans le contexte de l'observation terrestre, ces *datacube* représentent des téraoctets ou même des pétaoctets¹ de données stockées sur des serveurs. Bien que le nom *datacube* suggère trois dimensions, celui-ci n'est pas limité à ce nombre de dimensions. Un *datacube* peut être de une à n dimensions temporelles, spatiales ou autre. L'avantage de ces *datacube* est qu'il est possible d'en extraire des partitions de données (tranches, cube) et ceci pour n'importe quels dimensions ou axes.

Les *datacube* en trois dimensions peuvent être utilisés pour représenter un volume. À l'instar du nuage de point, les données y sont discrétisées. Ceci réduit l'entropie des données à 0. D'autres méthodes permettent de visualiser un volume, voici une liste non exhaustive :

- Le nuage de point est un ensemble de points dans l'espace continu [16] ;
- La soupe de polygone est un ensemble de triangles, pas forcément adjacents, dans l'espace continu [17] ;
- Le maillage est un objet tridimensionnel constitué de sommet d'arrêtes et de faces [18].

La figure 1.3 montre quatre exemples de rendu avec les quatre méthodes citées plus haut. Le rendu (a) est un nuage de point [16]. Le rendu (b) correspond à une soupe de polygone [17]. Le rendu (c) utilise un maillage [18]. Enfin, le rendu (d) utilise un *datacube* à trois dimensions avec l'algorithme du marching cube [19].

1. 1 péta équivaut à 10^{15} .

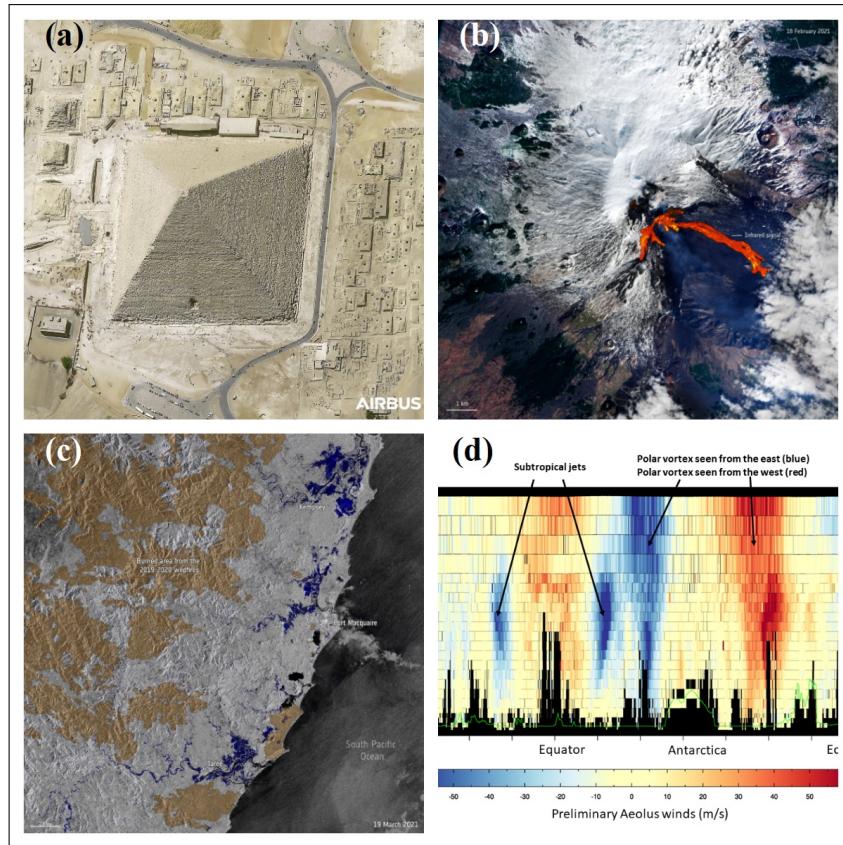


FIGURE 1.2 – Quatre prises de vues différentes venant de satellite de l’ESA et d’Airbus.

1.3.3 Limitations physiques des mesures des satellites

Pour couvrir la terre entièrement avec un échantillonnage offrant une résolution spatiale de 30 cm, la quantité de pixels nécessaires est d’environ 5.7 péta. En ignorant les données venant des surfaces immergées, on peut diviser la quantité de données par trois [3]. La résolution spatiale de l’échantillonnage a diminué au fil du temps. Cependant, elle ne suit pas la loi de Moore [20], il y a des limites physiques.

La première, les lois de Kepler [21], stipulent que le satellite ne peut pas être en orbite basse et être stationnaire. En effet, une telle orbite est appelée géostationnaire et elle se situe approximativement à 36 000 km d’altitude [22]. La figure 1.4 donne une idée de l’ordre de grandeur d’une telle orbite.

Le deuxième critère est celui du Rayleigh [23]. Il énonce que pour une longueur d’onde donnée, la résolution angulaire d’un télescope est inversement proportionnelle à l’ouverture² de ce dernier. Sur la figure 1.5, l’abscisse représente différentes résolutions angulaires et l’ordonnée des ouvertures différentes. On peut voir que pour une longueur d’onde donnée, la relation est linéaire. De ce fait, une augmentation de la résolution angulaire implique une augmentation du diamètre du miroir principal et par conséquent celle du poids du télescope. Ce dernier est évidemment limité par la puissance du lanceur.

1.3.4 Domaines d’utilisation

Les informations tirées de l’observation terrestre aident les scientifiques dans tous les domaines liés au changement climatique. Ces informations permettent d’approfondir et de vérifier nos connaissances sur différents domaines comme les courants océaniques, l’évolution des températures, l’effet de la fonte des neiges sur la circulation de l’eau et bien d’autres [24]. Dans le domaine de l’agriculture, ces données

2. L’ouverture correspond au diamètre du miroir principal.

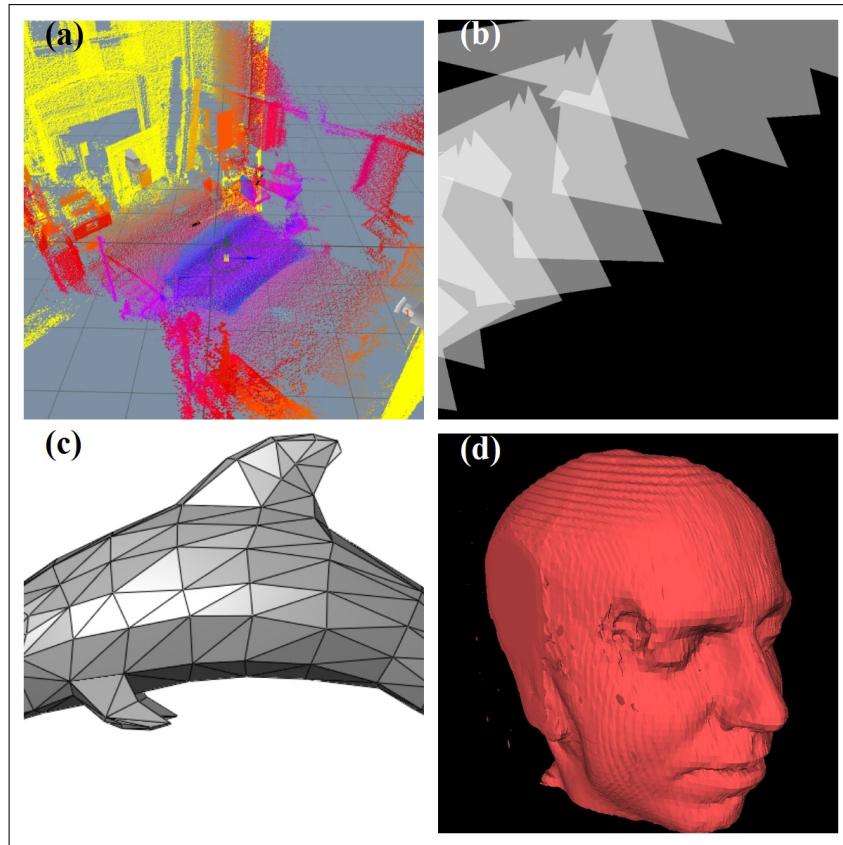


FIGURE 1.3 – Différentes méthodes de rendus.

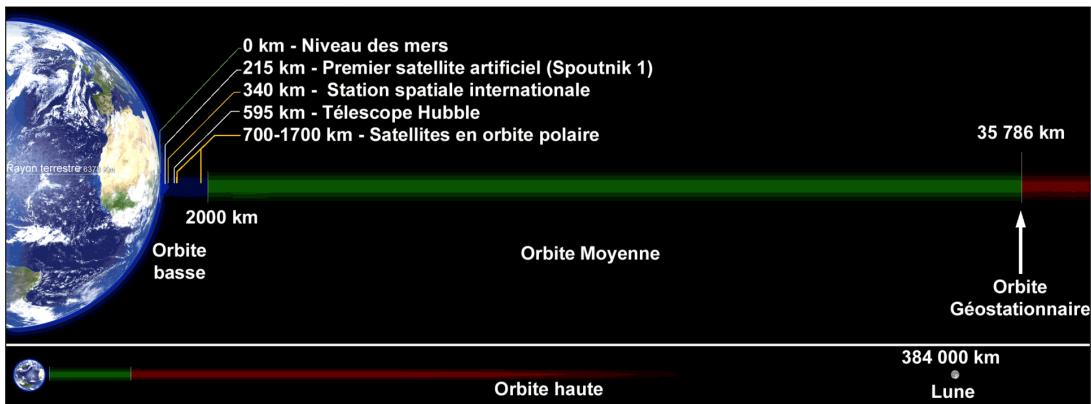


FIGURE 1.4 – Différentes orbites dont la géostationnaire.

peuvent être utiles pour mesurer l'humidité des sols et des cultures et ainsi prévoir des solutions appropriées pour l'irrigation en eau [25].

1.3.5 Etude du changement climatique

Le changement climatique est un sujet très important aujourd'hui. L'un des avantages de l'EO est qu'elle permet de surveiller une quantité de données suffisantes et pertinentes et d'ainsi, mieux comprendre les mécanismes du climat [1]. Le 25 septembre 2015, les Nations unies ont défini un ensemble de cent soixante-neuf cibles reparties dans dix-sept objectifs à atteindre : les Sustainable development goals (Objectifs de développement durable) (SDGs). Ces objectifs couvrent différents domaines liés au changement

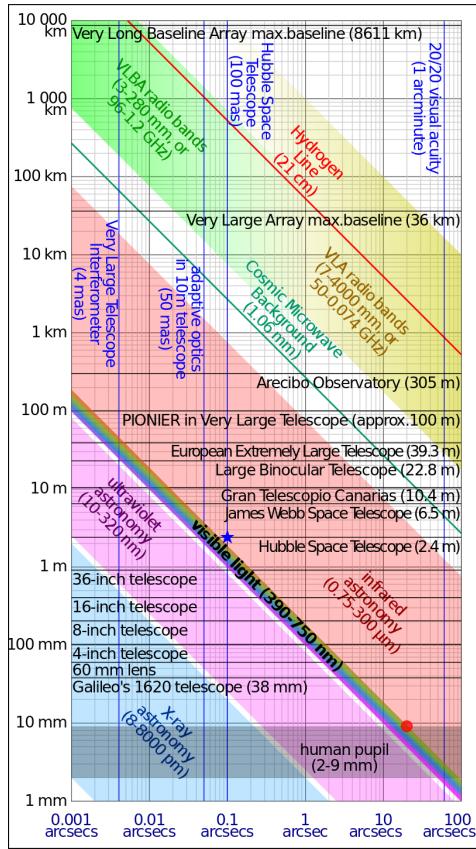


FIGURE 1.5 – Graphe des résolutions angulaires par rapport au diamètre du miroir pour différentes longueurs d'ondes.

climatique comme l'eau, l'énergie, le climat, les océans, l'urbanisation et les transports [26]. L'EO donne accès à vingt-quatre indicateurs qui permettent de mesurer vingt-neuf cibles dans onze des objectifs des SDGs [27].

1.3.6 Agriculture intelligente

L'agriculture est un domaine vulnérable aux aléas du changement climatique et notamment à cause des problèmes de disponibilité des ressources que ces changements peuvent induire. Ces changements ont actuellement, et auront d'autant plus ces prochaines décennies, non seulement un impact négatif sur l'agriculture, mais aussi indirectement un impact important sur notre mode de consommation. C'est donc dans ce domaine que les données de l'EO ont un potentiel. Cette agriculture intelligente a pour but notamment d'optimiser la production des récoltes avec un minimum de coûts et d'émission de Gaz à effet de serre (GES) [1]. Ces données issues de l'EO, mais aussi de capteurs terrestres, fourniront des informations prédictives concernant les conditions météorologiques, l'humidité de l'air et du sol, la croissance des plantations, la fertilité, etc... Par exemple, des sociétés comme "Earth Observing System" [28], "OneSoil" [29] ou "Planet" [30], pour ne citer qu'elles, proposent actuellement des solutions complètes de monitoring pour agriculteurs utilisant des données satellites.

Seulement, ces données sont uniquement consultables sur un plan fixe en 2 dimensions (façon carte Google Maps). L'originalité apportée dans ce projet résidera dans le fait que ces mêmes données seront visualisées dans un volume en trois dimensions et il sera alors possible d'afficher plus d'informations à l'aide de ce troisième axe représentant, selon le type de données, une composante temporelle ou spatiale.

1.3.7 Visualisation des *datacube*

Le projet 3DbvNCA [31] propose une application native de visualisation des *datacube* à trois dimensions. Elle a été développée en C++ avec l'API Open Graphics Library (OpenGL) [32] et elle permet de faire deux types de rendus différents :

- Un rendu X Ray ressemblant à celui d'une Imagerie par résonance magnétique (IRM) ;
- Un rendu permettant la visualisation des surfaces de niveau, pour un seuil donné.

Ce logiciel permet également de colorier les surfaces en fonction du sens et de la direction du vecteur normal à la surface. Cela permet d'avoir une meilleure représentation des surfaces dans l'espace.

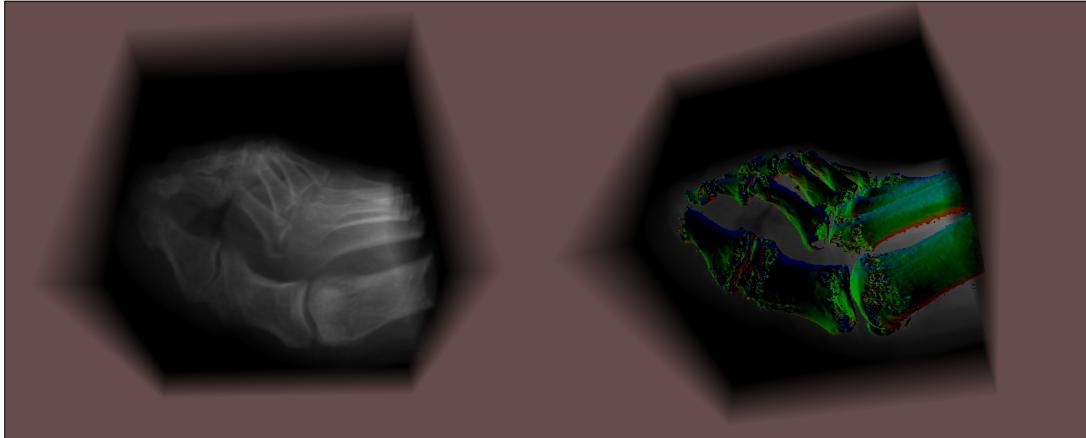


FIGURE 1.6 – Exemple de rendu de logiciel 3DbvNCA. A gauche le rendu X Ray et à droite le rendu de surface avec la colorisation en fonction des normales.

Le code source de 3DbvNCA sert entre autres de base de compréhension et de base de code pour les algorithmes à implémenter pour ce projet. Une explication en détails du fonctionnement de cet algorithme se trouve à la section 7.1.

1.3.8 Modèles de rendu

Le *raycasting* est utilisé notamment pour rendre des volumes dans le cadre de l'imagerie médicale. Par exemple, le rendu des balayages (*scan*) par rayons X ou rayons γ peut se faire par le biais de cet algorithme [33]. La figure 1.7 correspond au rendu provenant du computerized tomography (CT, Tomodensitométrie) *scan* d'un crâne humain.

Une autre utilisation autre fois populaire du *raycasting* était dans le domaine du jeu vidéo et ceci notamment dans les premiers jeux simulant une vue à la première personne comme Doom ou Wolfenstein 3D [35]. Les machines n'ayant pour la grande majorité pas de cartes accélératrices trois dimensions, cet algorithme était assez performant pour tourner sur les CPU de cette époque. Désormais, cet algorithme de rendu est d'autant plus performant avec la démocratisation des processeurs graphiques dédiés, même si il n'est plus autant utilisé dans ce domaine.

1.3.9 Application pour la visualisation des données terrestres

Les applications de visualisation des données issues de l'EO pour l'agriculture utilisent toutes le même principe : une carte sur laquelle on affiche les informations en utilisant une palette de couleurs. Par exemple, le projet AGRO-DE [36] propose une application Web permettant de visualiser les données en lien avec l'agriculture sur l'ensemble du territoire allemand. Cette application permet d'afficher plusieurs métriques et pour chacune, définir une transparence. Ceci permet de voir l'information tout en ayant encore la carte satellite devant nos yeux. La figure 1.8 représente des captures d'écran de cette application Web où l'on joue sur ce facteur de transparence. L'information affichée est l'intensité d'un indice de végétation durant l'année 2016 : la capture (a) utilise une transparence de 0%, c'est-à-dire que



FIGURE 1.7 – Rendu d'un CT scan d'un crâne en utilisant l'algorithme de *raycasting*. La dimension du datacube est 512x512x750 et chaque voxel est encodé sur 16 bits [34].

l'information n'est pas affichée, la (b) utilise une transparence de 25% et enfin, la (c) de 85%. Dans cette dernière capture, l'image satellite n'est presque plus visible, l'information perd alors de son contexte.

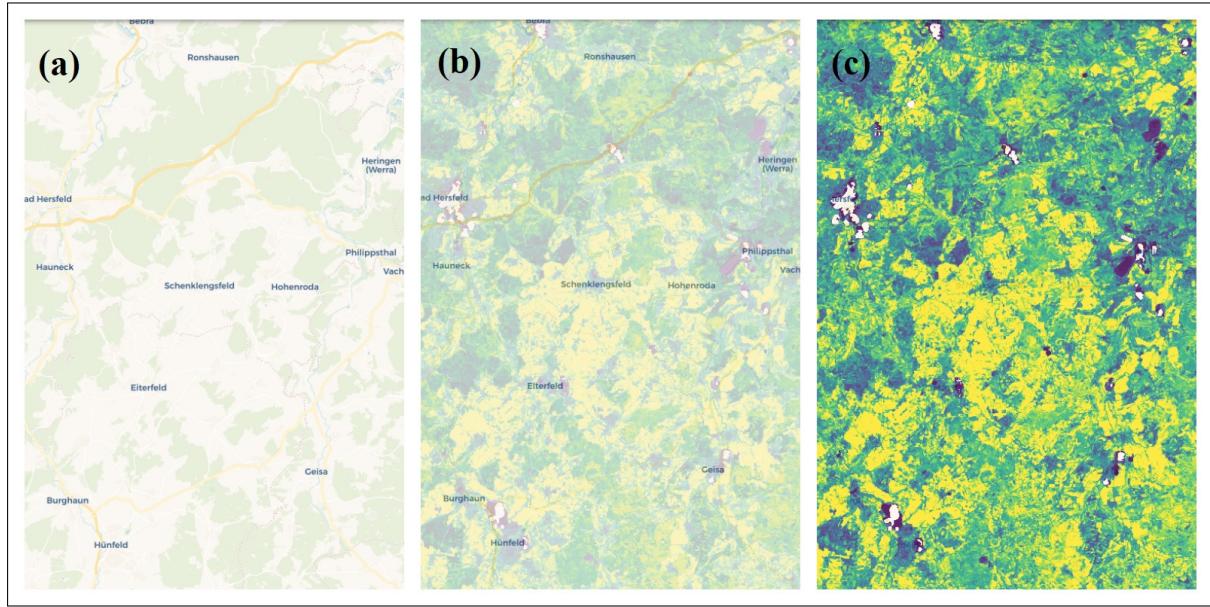


FIGURE 1.8 – Capture d'écran du site AGRO-DE en utilisant différentes transparences.

Dans le domaine maritime, certaines applications de visualisation de données bathymétriques offrent la possibilité d'afficher un rendu en trois dimensions de la profondeur. Un tel rendu correspond au modèle de rendu des courbes implicites sur une surface. Par exemple, l'entreprise Nautikaris propose une application de rendu en trois dimensions et en temps réel des données de topographie maritime. L'ensemble de données de ce produit ne vient pas de l'EO mais de différents types de sonars [37]. La figure 1.9 est une capture d'écran de cette application. On peut voir l'utilisation d'un modèle de rendu en trois dimensions des fonds marins.

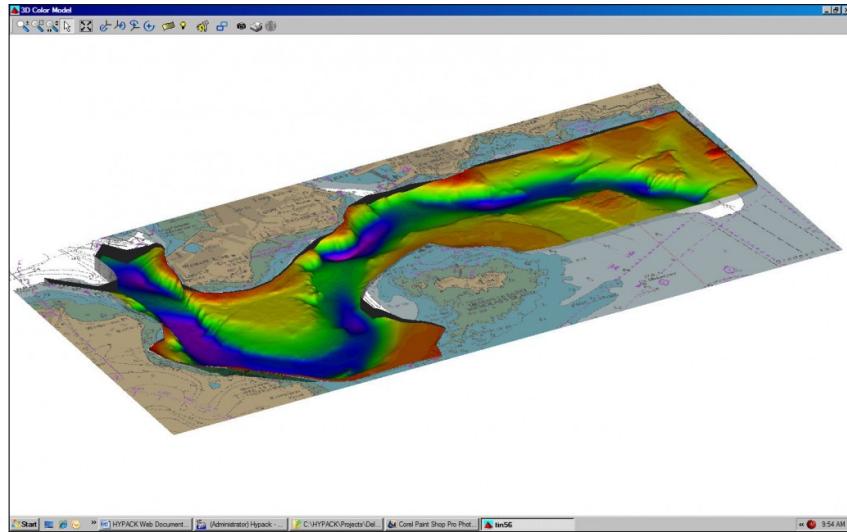


FIGURE 1.9 – Capture d'écran d'une application proposée par Nautikaris.

La collecte de données par sonars pose deux problèmes :

1. Le coût, car cette méthode demande du temps et de la main-d'œuvre ;
2. La dispersion des données.

L'utilisation de l'EO pour accomplir cette tâche permet d'atténuer ces problèmes [38]. En effet, la méthode de bathymétrie par satellite (SDB) permet d'avoir des cartes topographiques des fonds marins en utilisant des satellites. Des entreprises comme EOMAP [39] offrent des outils de visualisations en trois dimensions des fonds marins en utilisant les données issues de l'EO. La figure 1.10 est un exemple de rendu d'un des produits développés par EOMAP. Cette surface correspond à la topologie des fonds marins dans l'un des atolls des îles de Tuvalu en Océanie.

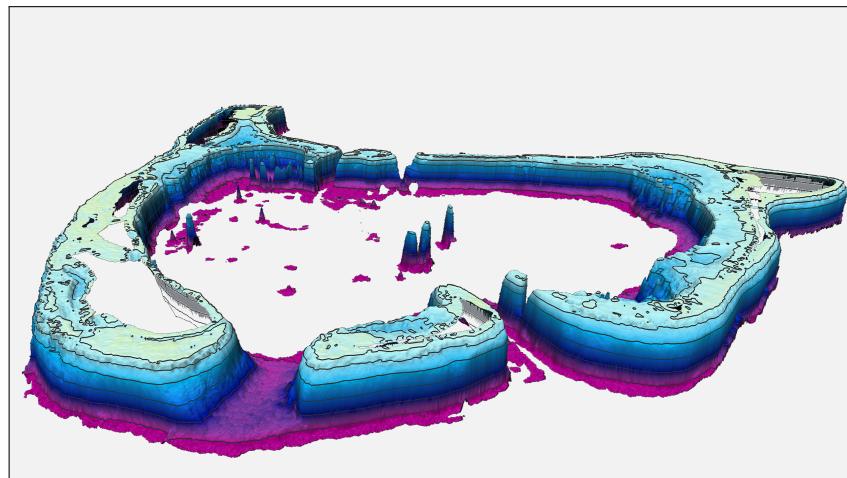


FIGURE 1.10 – Modélisation 3 dimensions des fonds marins d'un des atolls de Tuvalu dans un programme développé par EOMAP.

1.3.10 Contribution

L'application développée dans le cadre de ce projet permet la visualisation de *datacube* générique à deux dimensions. L'utilisation de l'API graphique WebGL [40] permet une grande portabilité, du fait qu'il suffit d'un navigateur Web pour le faire fonctionner, tout en gardant les performances d'un programme

développé avec un langage de bas niveau. La figure 1.11 présente la liste des navigateurs actuels supportant WebGL. L'utilisation de l'API plus moderne WebGL 2.0 n'est pas souhaitée pour le mandant. Bien que cette version offre des fonctionnalités supplémentaires vis-à-vis de sa version antérieure, les fonctionnalités de la première version suffisent pour le développement de ce projet. Les notebooks Jupyter sont utilisés parmi les équipes de recherche ; or, un rendu WebGL peut être effectué dans ces *notebook*. L'application ainsi développée propose d'afficher le rendu des *datacube* dans ces *notebook*.

IE	Edge	* Firefox	Chrome	Safari	Opera	Safari on * iOS	Opera Mini	* Android Browser	* Opera Mobile	* Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-3.6	4-7	3.1-5	10-11.5											
		12-18	4-23	8-32	5.1-7.1	12.1-18	3.2-7.1									
6-10	79-89	24-87	33-89	8-14	19-74	8-14.4		2.1-4.4.4	12-12.1				4-13.0			
11	90	88	90	14.1	75	14.5	all	90	62	90	87	12.12	14.0	10.4	7.12	2.5
		89-90	91-93	TP												

FIGURE 1.11 – Différents navigateurs supportant WebGL.

Bien que ces différents modèles de rendus soient déjà utilisés dans plusieurs domaines, l'application proposée centralise ces différents modèles et offre ainsi un outil complet de visualisation de *datacube* quelconque. Le tout dans un *notebook* Jupyter, qui est un outil utilisé par l'ensemble de la communauté scientifique, tout en gardant les performances d'une application native au moyen de l'utilisation de l'API WebGL.

Chapitre 2

Données issues de l'EO

2.1 Regroupement

Les données satellites sont récupérées de plusieurs sources. Une source correspond à un capteur d'un satellite d'observation terrestre. Ces sources de données proviennent du programme européen Copernicus. L'entreprise partenaire Solenix a comme objectif de centraliser les données sur une base de données Rasdaman installée sur leur serveur et de nous donner les outils nécessaires permettant l'accès à ces données. La figure 2.1 est un schéma qui représente comment les données sont regroupées et récupérées. Différents prétraitements pourraient être effectués sur le serveur afin de diminuer la charge de travail des clients. L'outil permettant l'accès à leur base de données est un client JavaScript (JS) qui possède les méthodes nécessaires pour communiquer avec une API développée sur leur serveur.

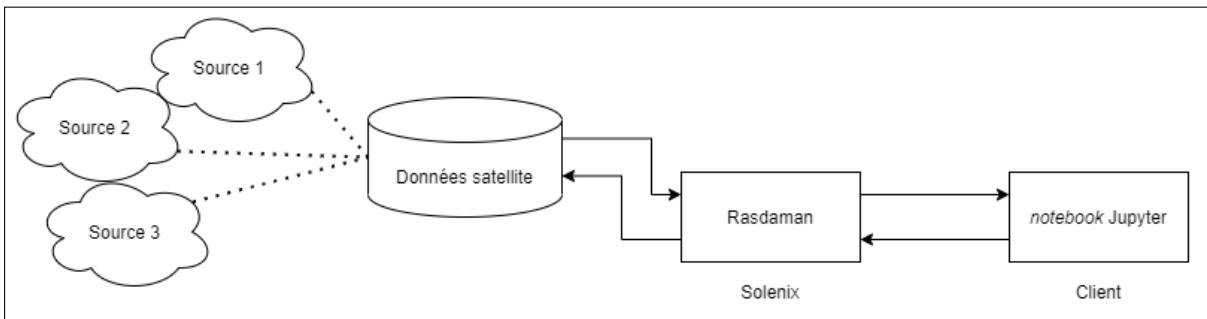


FIGURE 2.1 – Schéma représentant comment les données satellites sont enregistrées puis accédées.

2.2 Chargement

Chaque source de données représente une couverture satellite différente. Chaque couverture possède des données prises à des moments différents. Les données sont représentées par un *datacube*. Les *datacube* qui nous intéressent sont ceux à deux dimensions. Le client JS mis à disposition permet de récupérer ce *datacube* sous la forme d'une image en niveau de gris où l'intensité de celle-ci représente la valeur dans le *datacube*. Pour une couverture donnée, l'application développée va charger l'intégralité de la couverture. La couverture est représentée par une suite d'images en niveau de gris. Ceci permet de modifier le *datacube* à afficher en fonction du temps sans qu'il y ait de latence. Pour réduire le temps de chargement d'une couverture complète, les différents *datacube* d'une couverture sont chargés en parallèle en utilisant les promesses JS. En effet, il est possible de lancer une multitude de fonctions en parallèle et de placer une barrière de synchronisation afin d'attendre que tous les *datacube* soient chargés. La tableau suivant compare les temps de chargement sur différentes couvertures en utilisant ou pas le chargement asynchrone.

Source	Nombre d'image	Dimensions [pixel]	Temps synchrone [ms]	Temps asynchrone [ms]
FAPAR Suisse	74	1657 x 753	20'550	5'660
Humidité Suisse	699	552 x 251	64'806	9'347

Dès que les images sont transférées au client, elles sont transformées en texture WebGL. Ces textures sont placées dans un tableau. Pour changer l'image utilisée par le rendu, il suffit de changer la texture que l'on passe au shader dans la boucle de rendu.

Chapitre 3

Construction des courbes implicites

3.1 Modèle

Un des objectifs de ce projet est la visualisation des courbes de niveau d'un *datacube* à deux dimensions¹. Un *datacube* à deux dimensions est équivalent à une image, c'est-à-dire une matrice M_{ij} à deux dimensions où chaque pixel représente son intensité. L'image (a) de la figure 3.1 utilise différents niveaux de gris pour représenter l'intensité de ses pixels. Nous souhaitons réaliser une représentation graphique des courbes de niveau et permettre à l'utilisateur d'extraire ces courbes. Les courbes de niveau que l'on souhaite associer à cette matrice peuvent être représentées explicitement par l'équation suivante : $\{(i, j) \mid m_{ij} = iso\}$, avec iso comme constante fixée par l'utilisateur. Compte tenu de la nature implicite qui a permis de construire cette courbe, on parle alors de courbe implicite.

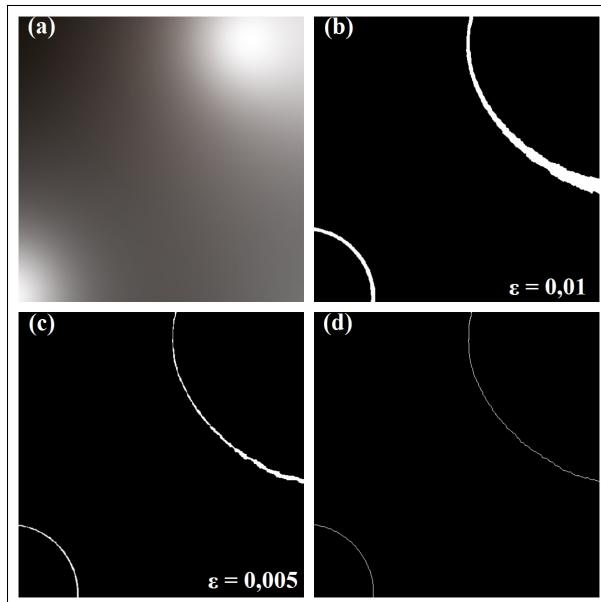


FIGURE 3.1 – Comparaison de l'algorithme naïf avec celui développé dans le cadre de ce projet.

La discréétisation de l'image mène à un problème : pour une valeur absolue iso donnée, il est probable qu'aucune des intensités des pixels de l'image ne correspondent à iso . Un algorithme naïf pour pallier ce problème serait d'utiliser une marge d'erreur ϵ et de considérer tous les pixels dont l'intensité se trouve dans l'intervalle $[iso - \epsilon, iso + \epsilon]$. Cependant, cet algorithme n'assure pas que la courbe de niveau ait une

1. En effet, le terme *datacube* peut s'adresser à des matrices à deux dimensions (xy), à trois dimensions (xyz) ou xyt ou à quatre dimensions ($xyzt$).

largeur constante. L'algorithme présenté dans cette section permet la construction des courbes implicites tout en s'assurant que leurs largeurs soient d'un pixel tout le long de la courbe.

Comme illustré dans la figure 3.1, l'algorithme naïf produit des courbes qui ne sont pas satisfaisantes alors que l'algorithme développé génère des courbes au pixel près. Les images (b) et (c) sont les résultats issus de l'algorithme naïf avec epsilon respectivement défini à 0.001 et 0.005. L'image (d) est le résultat dû à l'algorithme développé. Le seuil choisi pour générer ces courbes est 0.5. La figure 3.2 compare le fonctionnement des deux algorithmes au pixel près. Le seuil choisi est fixé à 0.5 et ϵ vaut 0.1. L'image (a) est l'image de base ; la (b) le résultat voulu ; la (g) le résultat obtenu en utilisant l'algorithme développé pour ce projet ; et la (c) le résultat en utilisant l'algorithme naïf. Les bords trouvés par l'algorithme naïf sont faux alors que ceux trouvés par l'algorithme utilisé correspondent à ceux voulus.

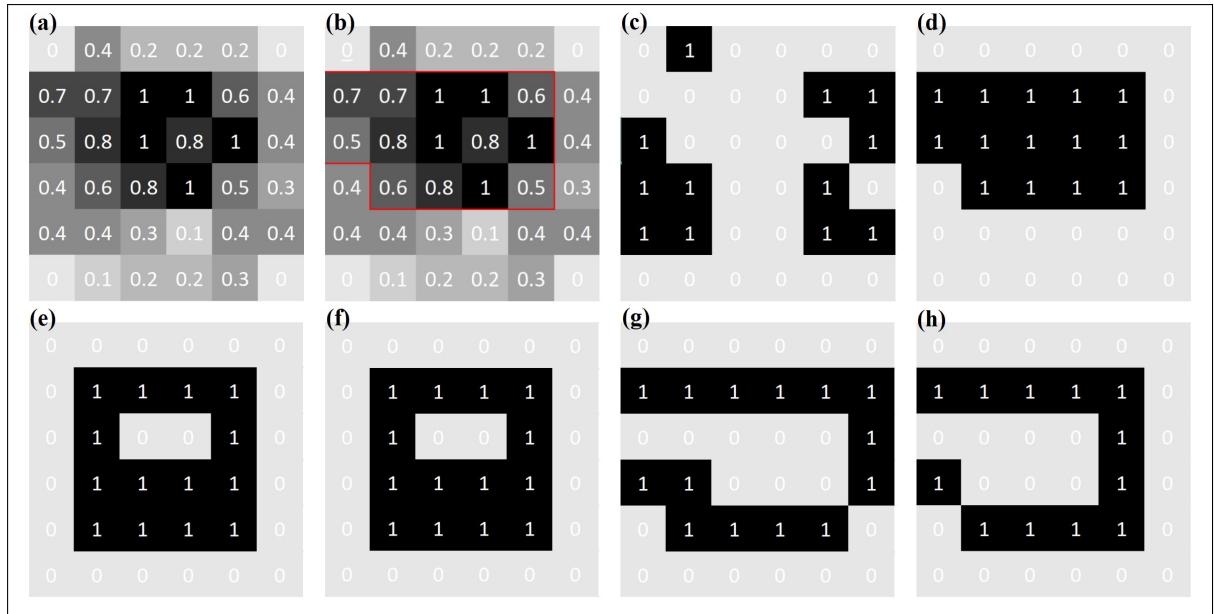


FIGURE 3.2 – Applications de filtres sur une image de base. La valeur iso est fixée à 0.5 et ϵ à 0.1. L'image (a) est l'image de base ; la (b) sont les bords voulus ; la (c) le résultat en utilisant l'algorithme naïf ; la (d) le résultat après la binarisation ; la (e) résultat en utilisant le filtre de Sobel ; la (f) le résultat en utilisant le filtre de Prewitt ; la (g) le résultat en utilisant l'algorithme développé pour ce projet et la (h), le résultat si l'on change le noyau G_x .

L'algorithme fonctionne en deux phases. À chacune des étapes, il prend une image en entrée et produit une image en sortie. La première phase consiste à binariser l'image par rapport au seuil donné par l'utilisateur, c'est-à-dire la variable iso . Pour tous les pixels de l'image, l'intensité du pixel correspondant dans l'image de sortie est 0 si la valeur du pixel de l'image d'entrée est plus petite que le seuil ou 1 dans le cas contraire. Un exemple d'application de cette phase est présenté dans l'image (d) de la figure 3.2.

Notre image est maintenant segmentée en une ou plusieurs régions. Les bords de ces régions correspondent aux courbes que l'on recherche. La seconde phase permet d'approximer ces bords. Elle consiste en deux étapes :

1. Approximer les dérivées partielles de l'intensité de l'image selon les deux axes de cette dernière, pour tous les pixels.
2. Calculer la norme du gradient en utilisant les valeurs des dérivées partielles calculées précédemment. Si la valeur du gradient est différente de 0, l'intensité du même pixel sur l'image finale vaut 1, sinon, elle vaut 0.

Le calcul des dérivées partielles se fait à l'aide d'un produit de convolution sur l'image avec les noyaux

centrés suivants :

$$G_x = \begin{bmatrix} -1 & 1 & 0 \end{bmatrix}, G_y = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} \quad (3.1)$$

Ces noyaux sont respectivement utilisés pour calculer les dérivées partielles par rapport à x et y . Ils diffèrent du filtre de Sobel [41] et de celui de Prewitt [42] car ces derniers ne permettent pas produire des bords d'un pixel de largeur. Les images (e), (f) et (g) de la figure 3.2 correspondent respectivement aux bords obtenus en utilisant les filtres de Sobel, Prewitt et le filtre utilisé dans ce projet. Les images (e) et (f) sont identiques. En effet, le choix du filtre parmi ceux de Sobel et Prewitt ne change pas le résultat de l'algorithme. Le filtre utilisé peut être remplacé par un autre. Par exemple, remplacer le filtre G_x par $\begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$ permet aussi d'obtenir des courbes de largeur constante d'un pixel. La différence entre ces deux filtres réside dans le fait que le premier filtre génère un bord qui appartient à la région considérée seulement si la dérivée partielle est positive alors que le deuxième filtre, seulement si la dérivée partielle est négative. Les images (g) et (h) de la figure 3.2 exposent la différence due au remplacement du noyau G_x . On remarque que la courbe est déplacée à droite. En effet, l'un des G_x assure que la courbe fait partie de la région lorsqu'elle décrit un bord à droite de la région binarisée et qu'elle ne fait pas partie de la région lorsqu'elle décrit un bord à gauche. L'autre G_x permet de faire l'inverse.

Le calcul de la norme de gradient se fait en utilisant la norme de la distance de Manhattan. Elle est définie comme la somme des valeurs absolues des composantes du vecteur. Dans notre cas, cela revient à calculer la somme des valeurs absolues des dérivées partielles. Cette norme permet d'éviter d'avoir à calculer les deux carrés présents dans le calcul de la norme euclidienne d'un espace à deux dimensions tout en ne perdant aucune précision. En effet, la seule information nécessaire à la création du bord d'une région est de pouvoir dire si le gradient est nul ou pas. Or, par définition d'une norme, si la norme d'un vecteur est nulle pour une norme donnée, elle l'est pour toutes les normes, dont celle de Manhattan [43]. Finalement, une binarisation similaire à la méthode utilisée durant la première phase est appliquée pour définir l'intensité de l'image finale.

En plus d'assurer que la largeur de la courbe soit d'un pixel, cet algorithme peut être exécuté en parallèle pour chaque pixel. Cette capacité permet de le faire tourner en temps réel à l'aide de la puissance de calcul des GPUs. Une telle utilisation des GPUs est appelée General-purpose processing on graphics processing units (GPGPU, Calcul générique sur processeur graphique), la section suivante décrit comment a été implémenté cet algorithme.

Une amélioration de cet algorithme est possible. En faisant deux passes dans lesquelles on alterne les filtre G_x et G_y avec leur deuxième version et en gardant le signe de la somme des dérivées, on peut construire la courbe de niveau tout en s'assurant qu'elle fasse entièrement partie, ou au contraire pas parti, de la région binarisée. Cette amélioration n'a pas été implémentée.

3.2 Implémentation

Le modèle expliqué à la section précédente est parallélisable. On peut pour chaque pixel de l'image, exécuter un programme qui calculera selon l'intensité du pixel d'entrée et de ses voisins, l'intensité du pixel de sortie. Le GPU offre une puissance de parallélisation suffisante. Chaque opération effectuée sur l'image est alors transcrise comme un programme pour le GPU. Ce type de programme est appelé : shader. Le langage de programmation utilisé est OpenGL Shading Language (GLSL), il est utilisé par WebGL.

Les shaders permettent de travailler avec des images par le biais des textures. Ces textures sont directement stockées sur le GPU et l'accès aux différents pixels de cette dernière est dès lors optimisé. Les images utilisées par WebGL possèdent trois canaux de couleurs et un d'opacité. Ces canaux correspondent au format de codage de couleur Red Blue Green Alpha (RGBA, Rouge Vert Bleu Alpha). Le canal rouge est utilisé pour transmettre l'intensité du champ scalaire. Une autre fonctionnalité de WebGL utilisée dans le cadre de ce projet est l'interpolation des intensités des pixels gérée par WebGL. Ceci permet de demander la couleur de l'image entre deux pixels et d'obtenir une couleur interpolée. La méthode

d'interpolation utilisée peut être définie. Par défaut, c'est l'interpolation linéaire qui est utilisée. Cette automatisation du processus d'interpolation est utilisée pour la création des palettes de couleur. Cette méthode est expliquée à la section 7.5.

Bien que WebGL soit avant tout utilisé pour faire des rendus sur un écran, il est tout à fait possible de faire le rendu sur une texture. Cette possibilité ouvre le champ au GPGPU. Cette méthode est appelée : rendu dans une texture. Elle est utilisée autant de fois qu'il est nécessaire d'opérer sur l'image. Voici un résumé des différentes opérations effectuées pour chaque pixel :

1. Binarisation ;
2. Calcul de la norme de Manhattan du gradient ;
3. Binarisation (seuil défini à 0).

WebGL normalise les différents canaux de couleur d'un rendu. Ceci implique, qu'une intensité négative sera redéfinie comme 0 et qu'une intensité plus grande que 1 sera redéfinie comme 1. Cette normalisation permet d'implémenter l'opération 3 sans utiliser d'instruction conditionnelle et donc, d'éviter la divergence de *thread*. Voici le pseudo-code de la binarisation :

Algorithme 1 : Binarisation

Data : I Image à traiter
i et *j* Coordonées du pixel à traiter dans l'image
c Seuil à utiliser
Résultat : *z* intensité final du pixel

$$z \leftarrow (I[i, j] - c) * \text{MAX_FLOAT};$$

Dans ce pseudo-codes et les suivants, la paire (*i*, *j*) représente les coordonnées du pixel, *I* l'image sous forme d'un tableau à deux dimensions, *c* le seuil et *MAX_FLOAT* la constante correspondant au nombre flottant maximum du langage de programmation utilisé.

Les étapes 2 et 3 peuvent être regroupées en 1 programme. Ce programme va calculer les deux dérivées partielles de l'image en (*i*, *j*), respectivement, *Dx* et *Dy*. Ces dérivées partielles sont calculées en appliquant un produit de convolution avec les noyaux décrits dans l'équation 3.1. Pour terminer, l'intensité finale est calculée comme la somme des valeurs absolues des dérivées partielles. La somme de deux nombres valant soit 0 soit 1 est forcément comprise dans l'ensemble de nombre suivant {0, 1, 2}. De ce fait et par la normalisation automatique effectuée par WebGL, il n'est pas nécessaire de binariser une nouvelle fois l'image. Voici l'algorithme :

Algorithme 2 : Norme du gradient

Data : I Image à traiter
i et *j* Coordonées du pixel à traiter dans l'image
Résultat : *N* Norme du gradient. Elle correspond donc à l'intensité du pixel final.

$$\begin{aligned} Dx &\leftarrow I[i, j] - I[i-1, j]; \\ Dy &\leftarrow I[i, j] - I[i, j-1]; \\ N &\leftarrow |Dx| + |Dy|; \end{aligned}$$

Le résultat final est une texture. Cette dernière pourrait être réutilisée dans un autre shader qui cette fois-ci, l'utilisera à une fin de visualisation sur l'écran. Dans le cadre de ce projet, l'extraction de caractéristiques des courbes de niveaux pour permettre une reconstruction paramétrique de ces dernières est demandée. Cette opération est en partie faite sur le GPU et elle est décrite dans le chapitre 4.

3.3 Résultats

La figure 3.3 est une grille représentant différents résultats de l'algorithme. Sur la colonne de gauche, les images de base représentant les *datacube* dont on veut afficher les courbes de niveau. Ensuite, sur les deux autres colonnes, les courbes de niveau de l'image de base sont affichées. La deuxième colonne est représentée par un *iso* de 0.3 et la troisième, par un *iso* de 0.6. Les images sont triées de haut en bas. L'image ressemblant le plus à un cas d'école se trouve en haut alors que celle correspondante plus à des données satellites se trouve en bas. La dernière image correspond à une carte de hauteurs d'une partie de l'Islande. Pour toutes ces données, l'algorithme fonctionne sans souci.

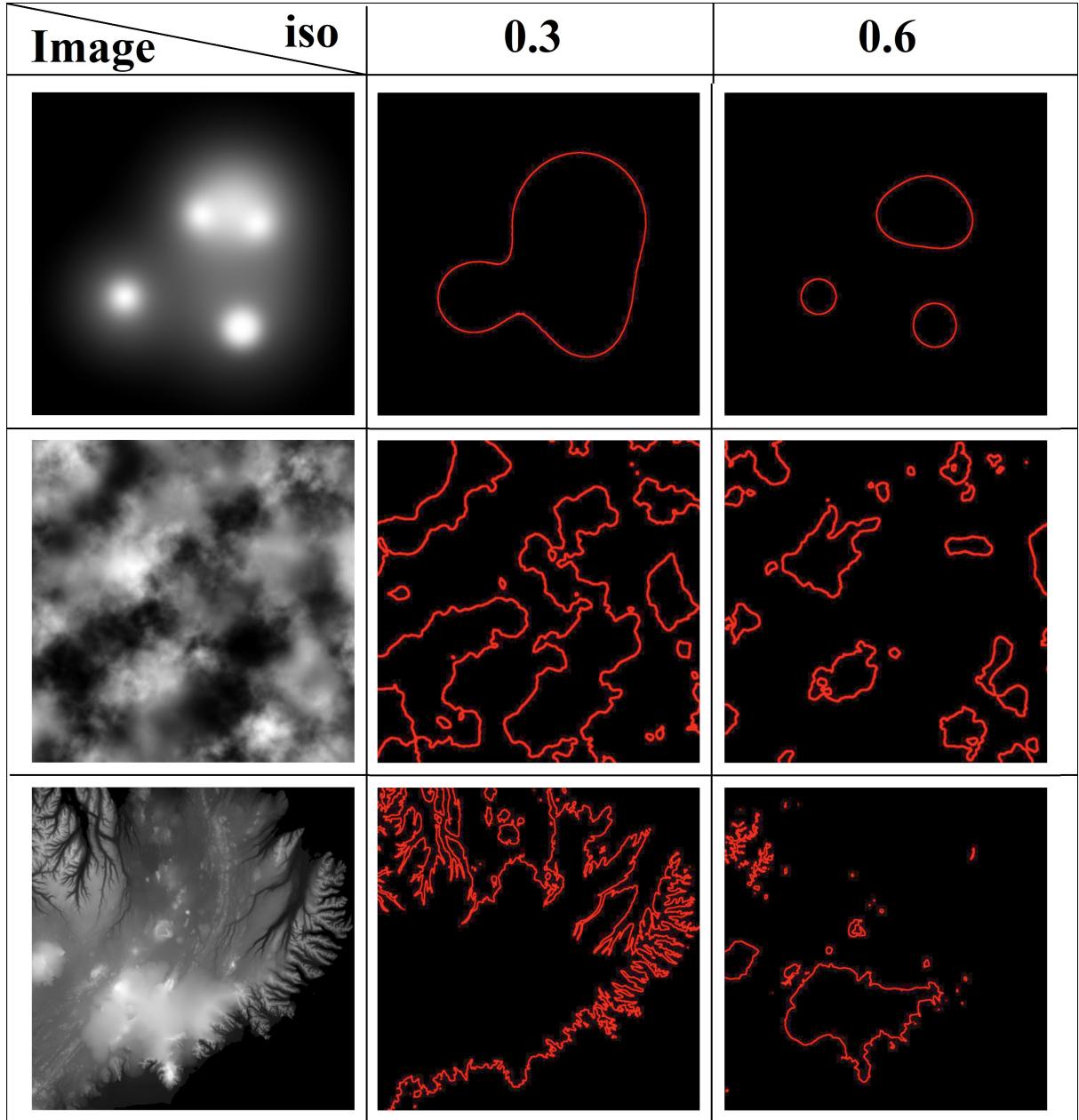


FIGURE 3.3 – Application de l'algorithme de construction des courbes de niveau à différentes images.

Chapitre 4

Paramétrisation des courbes implicites

4.1 Modèle

Pour paramétriser une courbe discrète, il faut obtenir une équation paramétrique d'une courbe réelle telle qu'en échantillonnant la courbe réelle sur une grille régulière à deux dimensions nous obtenons la courbe discrète de départ. En utilisant des équations paramétriques réelles, il faut autant d'équation qu'il y a de dimension. Dans le cas d'une image, il faut ainsi deux équations paramétriques. Formellement, on peut décrire une courbe $\mathcal{C}(t)$ comme :

$$\mathcal{C}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

Où t est le paramètre, $x(t)$ et $y(t)$ les équations définissant respectivement la position par rapport à l'abscisse et l'ordonnée. Deux objectifs se cachent derrière la paramétrisation des courbes discrètes. Le fait que la courbe paramétrique obtenue soit continue nous permet de l'échantillonner en tout t . Il n'est donc pas nécessaire de faire une interpolation pour reconstruire la courbe. Le deuxième avantage est que les méthodes utilisées pour transformer la courbe en équation paramétrique permettent de réduire la quantité d'information nécessaire pour la sauvegarder. En contreparti, une erreur sur la courbe se forme. Deux méthodes sont utilisées pour paramétriser les courbes. La première utilise les notions de Discrete Fourier Transform (DFT, Transformation de Fourier discrète) et la deuxième, les splines de Catmull-Rom.

4.1.1 Paramétrisation en utilisant la DFT

Les équations qui définissent notre courbe sont périodiques étant donné qu'en suivant cette dernière nous retombons sur le point de départ. D'après le théorème de convergence normal de Dirichlet, la série de Fourier d'une fonction $f(t)$ qui est T -périodique converge vers la fonction $f(t)$ [44]. On peut alors écrire :

$$f(t) = \sum_{k=-\infty}^{\infty} c_k(f) e^{2i\pi \frac{k}{T} t} = c_0 + \sum_{k=1}^{\infty} H_k(f)(t), \quad H_k(f)(t) = c_k(f) e^{2i\pi \frac{k}{T} t} + c_{-k}(f) e^{-2i\pi \frac{k}{T} t}$$

c_k sont les coefficients de Fourier de la fonction $f(t)$, t le paramètre et i l'unité imaginaire. H_k est appelé l'harmonique de rang k . Les coefficients de Fourier sont suffisants pour reconstruire la fonction $f(t)$. En fixant un nombre entier N on peut définir la série partielle de Fourier :

$$S_N(f)(t) = c_0(f) + \sum_{k=1}^N H_k(t)$$

La série partielle de Fourier permet d'approximer la fonction $f(t)$ tout en fixant le nombre de coefficients à calculer. De plus, le théorème de Riemann-Lebesgue [45] permet d'affirmer que les coefficients de Fourier $c_k(f)$ tendent vers zéro quand k tend vers l'infini.¹ Ceci implique que l'information contenue

1. Seulement si f est intégrable sur une période.

dans les premiers coefficients est supérieure à celle contenue dans les coefficients où k est grand. Plus formellement, si notre fonction f est de classe \mathcal{C}^k , c_k est négligeable devant $1/n^k$. Les coefficients de Fourier s'obtiennent, dans le cas d'une fonction discrète, en appliquant la DFT. La DFT prend N nombres réels² en entrée et retourne les N premiers coefficients de Fourier. Les N premiers coefficients de Fourier peuvent être utilisés pour classifier des formes par rapport à leur contour. On parle alors de descripteurs elliptiques de Fourier. Dans [46], ces descripteurs sont utilisés pour quantifier les dysmorphoses mandibulaires chez la souris.

Dans le cas discret, pour obtenir la fonction de base à partir de ses coefficients on utilise l'équation suivante :

$$S_N(f)(t) = \sum_{k=-N/2}^{N/2} c_k(f) e^{2i\pi \frac{k}{T} t}$$

Cette transformation inverse permet d'obtenir la série $S_N(f)(t)$. Cette fonction est T -périodique. Le fait de pouvoir choisir le nombre de coefficients à extraire de la courbe permet donc de fixer la quantité d'information voulue pour enregistrer cette dernière en contrepartie d'une certaine perte d'information. Pour choisir les N points de la courbe discrète, deux méthodes sont proposées :

1. Faire un échantillonnage régulier de la courbe ;
2. Prendre les sommets d'un polygone d'approximation de la courbe.³

La figure 4.1 compare les deux méthodes d'échantillonnage. La partie gauche correspond à l'échantillonnage régulier alors que la partie droite, à celui par approximation polygonale. Les points échantillonnés sont en rouge, la courbe en bleu et en vert, le polygone reliant les points rouges.

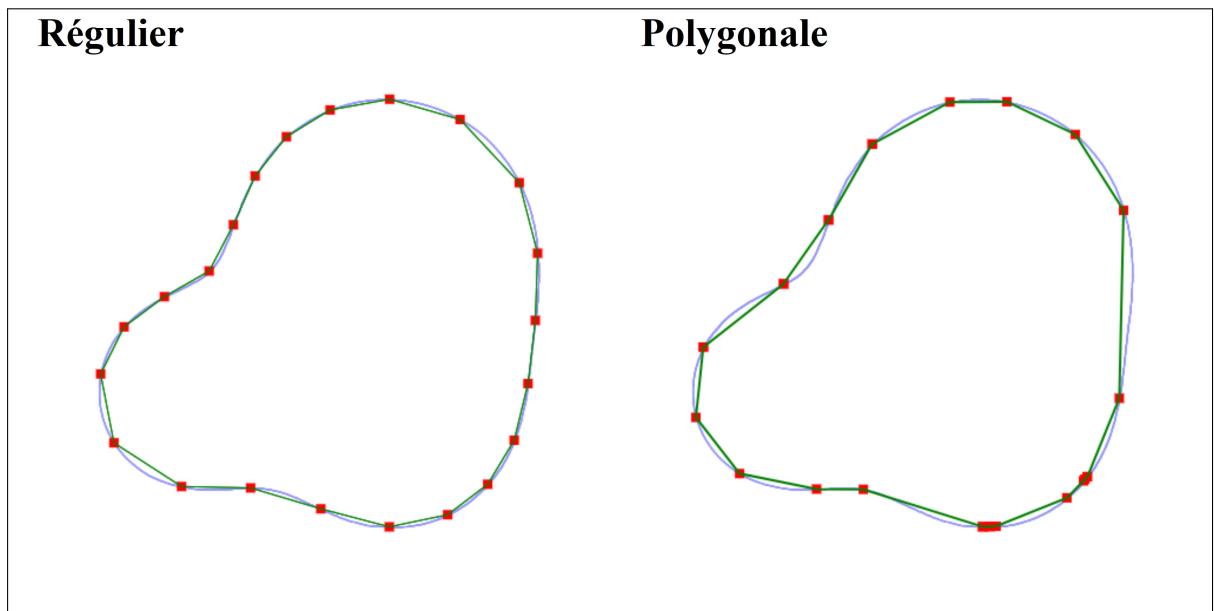


FIGURE 4.1 – Comparaison des deux méthodes d'échantillonnage.

En échantillonnant notre courbe on peut donc réduire la quantité d'information nécessaire pour la sauvegarder tout en permettant sa reconstruction. La reconstruction est faite en utilisant la transformée discrète de Fourier sur les échantillons pris sur $x(t)$ et $y(t)$ puis en appliquant la transformé inverse pour reconstruire les séries de Fourier S_N des fonctions respectives. Soit N le nombre de coefficients choisis, $x(t)$ et $y(t)$ les équations décrivant la courbe $\mathcal{C}(t)$, $c_k(f)$ le k -ième coefficient de la fonction $f(t)$ et $\tilde{\mathcal{C}}(t)$

2. De manière plus générale, la DFT fonctionne aussi avec des nombres complexes.
 3. La création du polygone d'approximation est expliqué dans la section 5.1.

la courbe d'approximation, alors :

$$\tilde{\mathbf{C}}(t) = \begin{pmatrix} S_N(x)(t) \\ S_N(y)(t) \end{pmatrix}$$

Ici, il faut calculer les coefficients des fonctions $x(t)$ et $y(t)$. Une manière plus élégante d'exprimer cette courbe serait d'utiliser une unique fonction complexe, où la partie réelle de celle-ci correspondrait à la fonction $x(t)$ et la partie imaginaire à la fonction $y(t)$: $z(t) = x(t) + iy(t)$. Notre courbe $\tilde{\mathbf{C}}(t)$ s'exprime maintenant par rapport d'une seule fonction :

$$\tilde{\mathbf{C}}(t) = \begin{pmatrix} \Re(S_N(z)(t)) \\ \Im(S_N(z)(t)) \end{pmatrix} \quad (4.1)$$

Cette fois-ci, une seule DFT a besoin d'être calculée, celle de la fonction complexe $z(t)$. Cependant, cela ne réduit pas le nombre d'opérations à effectuer, car la DFT d'une fonction réelle offre des propriétés qui permettent de simplifier les calculs.

4.1.2 Paramétrisation en utilisant les splines de Catmull-Rom

Les splines de Catmull-Rom appartiennent à la famille des splines d'interpolation cubiques. Cette spline se calcule en fonction d'une séquence de quatre points de contrôles $\{\mathbf{P}_i\}_{0 \leq i \leq 4}$. Dans le cas à deux dimensions, les points appartiennent à \mathbb{R}^2 . Une des particularités de ce spline est que la tangente en tout point \mathbf{P}_i est calculé en fonction du point précédent et suivant. Plus exactement, le vecteur tangent du point \mathbf{P}_i est égal à $\tau(\mathbf{P}_{i+1} - \mathbf{P}_{i-1})$. τ est un paramètre plus grand ou égale à zéro, il permet de modifier le comportement de la spline. Pour certaines paramétrisations, la spline prend des noms spécifiques. Le tableau suivant expose les différents noms en fonction des différentes paramétrisations [47].

τ	0	0.5	1
Nom	Uniforme	Centripète	Cordale

La spline s'exprime comme une fonction d'un paramètre t . Voici une représentation matricielle de la spline $\mathbf{p}(t)$:

$$\mathbf{p}(t) = [1 \ t \ t^2 \ t^3] \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} \mathbf{P}_{i-2} \\ \mathbf{P}_{i-1} \\ \mathbf{P}_i \\ \mathbf{P}_{i+1} \end{bmatrix}$$

Le facteur a été fixé à 0.5 car, cette paramétrisation de la courbe offre les courbes les plus élégantes [47]. Les splines utilisées sont donc des splines de Catmull-Rom centripète.

La partie de la spline qui nous intéresse est celle entre les points \mathbf{P}_i et \mathbf{P}_{i+1} . Si l'on définit t_i tels que $\mathbf{p}(t_i) = \mathbf{P}_i$, l'intervalle sur lequel nous voulons calculer $\mathbf{p}(t)$ est $[t_1, t_2]$. Les t_i se calcurent de manière récursive :

$$t_{i+1} = \begin{cases} 0 & \text{si } i = 0 \\ [d_2(\mathbf{P}_i, \mathbf{P}_{i+1})]^\tau + t_i & \text{sinon} \end{cases}$$

Où d_2 est la distance euclidienne. De la manière semblable à la méthode précédente, il est nécessaire d'échantillonner notre courbe discrète. Les deux méthodes proposées pour la DFT sont utilisées. Pour un nombre arbitraire de points, il faut définir la courbe $\tilde{\mathbf{C}}(t)$ par morceau. Soit N le nombre de points échantillonnés. Si T est la période voulue, on subdivise régulièrement l'intervalle $[0, T]$ par N . Ceci génère autant de subdivision qu'il y a de morceau de spline à calculer. Soit I_i le i -ième sous intervalle de la subdivision et $\{t_{i,j}\}_{0 \leq j \leq 4}$ la séquence des paramètres des points de contrôles de la i -ième spline $s_i(t)$. La figure 4.2 est un exemple du principe de subdivision de la courbe. On y voit les points de contrôles et les différents morceaux des différentes splines. Les splines s_0 et s_3 y sont indiquées. Elles sont restreintes pour être calculées respectivement sur l'intervalle $[t_{0,1}, t_{0,2}]$ et $[t_{3,1}, t_{3,2}]$.

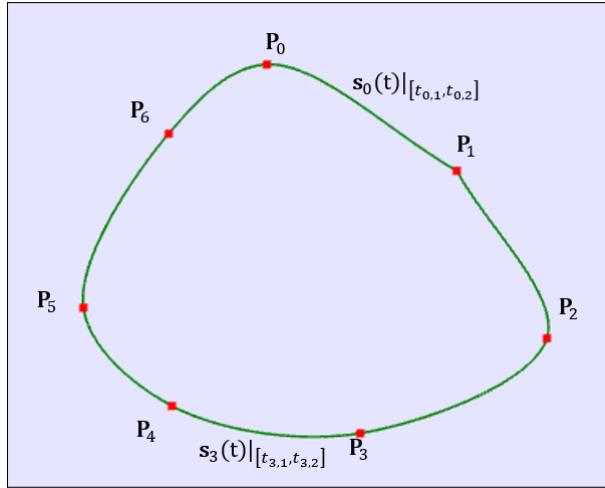


FIGURE 4.2 – Différentes splines composants une courbe.

Pour pouvoir passer du sous intervalle I_i à l'intervalle permettant de calculer la spline entre les points intéressants, il faut définir une application affine. Il existe une unique application affine $g_i : I_i \rightarrow [t_{i,1}, t_{i,2}]$. Cette application est définie comme :

$$g_i(t) = \left[t - (i-1) \frac{T}{N+1} \right] \frac{N+1}{T} (t_{i,2} - t_{i,1}) + t_{i,1}$$

On peut maintenant définir formellement notre courbe $\tilde{\mathcal{C}}(t)$ par morceaux :

$$\tilde{\mathcal{C}}(t) = \begin{cases} (s_0 \circ g_0)(t) & \text{si } t \in I_1 \\ \vdots \\ (s_i \circ g_i)(t) & \text{si } t \in I_i \\ \vdots \\ (s_{N-1} \circ g_{N-1})(t) & \text{si } t \in I_N \end{cases}$$

Pour construire les dernières et les premières splines il est nécessaire de boucler sur les points de contrôles. Par exemple, les points de contrôles que l'on va utiliser pour l'avant dernière spline $s_{N-2}(t)$ sont \mathbf{P}_{N-3} , \mathbf{P}_{N-2} , \mathbf{P}_{N-1} et \mathbf{P}_0 .

4.2 Implémentation

Pour que l'échantillonnage ait lieu, il faut que nos courbes soient représentées par une liste de points, or, elles sont pour l'instant représentées par une image binaire. L'algorithme développé permet de récupérer les listes de points des différentes courbes d'une image binaire. L'algorithme va chercher le premier pixel appartenant à la courbe dans l'image pour fixer le point initial. De ce point initial, on va chercher un pixel voisin 4-connexe qui n'a pas encore été ajouté. Si un tel pixel existe, il devient le pixel courant. Sinon, on cherche parmi les voisins 8-connexes (en excluant les 4-connexes déjà contrôlés). Le nouveau point étant trouvé, on contrôle qu'il soit différent du point initial. S'il l'est, l'algorithme continue. Cet algorithme ne marche pas dans tous les cas :

1. Si la courbe possède une bifurcation ;
2. Si une partie de la courbe se trouve en dehors de l'image. C'est-à-dire, que la courbe ne se boucle pas dans l'image.

Dans le premier cas, une partie de la courbe sera ignorée. Dans le deuxième, la courbe sera complètement ignorée. Un exemple de ces cas non fonctionnels est décrit dans la figure 4.3.

La courbe rouge et blanche représente le premier cas et la courbe verte le second. Parmi la courbe rouge et blanche, la courbe rouge est celle dont la liste de point est récupérée. Les courbes ont été grossies pour rendre la figure plus lisible. Un meilleur algorithme devrait être développé. Voici l'algorithme développé :

Algorithme 3 : Liste des courbes sous forme de liste de points à partir d'une image binaire

```

Data : I Une image binaire représentant les courbes de l'image
Réultat : L Une liste de courbe C représentée par une liste de points
L ← []
K ← [] ;           // Liste des points qui appartiennent à une courbe qui n'a pas
                  fonctionné
for Toutes les paires de coordonnées ( $x, y$ ) de I do
  if  $I[x, y] = 1$  then
    C ← []
    pointCourant ← (x, y)
    do
      if Un des pixels du voisinage V4 de pointCourant a comme intensité 1 et n'est pas
          dans C ni dans K then
        |   p ← Le pixel voisin
        |   L.push(p)
      else if Un des pixels du voisinage V8 de pointCourant a comme intensité 1 et n'est
          pas dans C ni dans K then
        |   p ← Le pixel voisin
        |   L.push(p)
      else
        |   K.concat(C)
    while pointCourant != (x, y);
    L.push(C)
  end
end
```

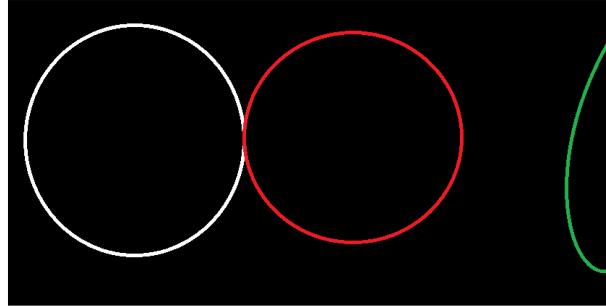


FIGURE 4.3 – Cas non fonctionnel de la transformation d'une courbe en liste de points.

4.3 Résultats

La figure 4.4 compare les deux méthodes de paramétrisation. L'échantillonnage choisi est l'échantillonnage régulier. La courbe verte est celle qu'on essaie d'approximer. Les courbes sont triées en fonction du nombre de points échantillonnes. La ligne du haut correspond à la paramétrisation par DFT et celle du bas par les splines de Catmull-Rom. On remarque que plus le nombre d'échantillons est élevé, plus la courbe d'approximation ressemble à celle de base. Dans cet exemple, aucune des deux méthodes ne se distingue au niveau de l'erreur par rapport à la courbe de base. Une comparaison plus en profondeur et une définition de l'erreur est faite à la section 8.2.

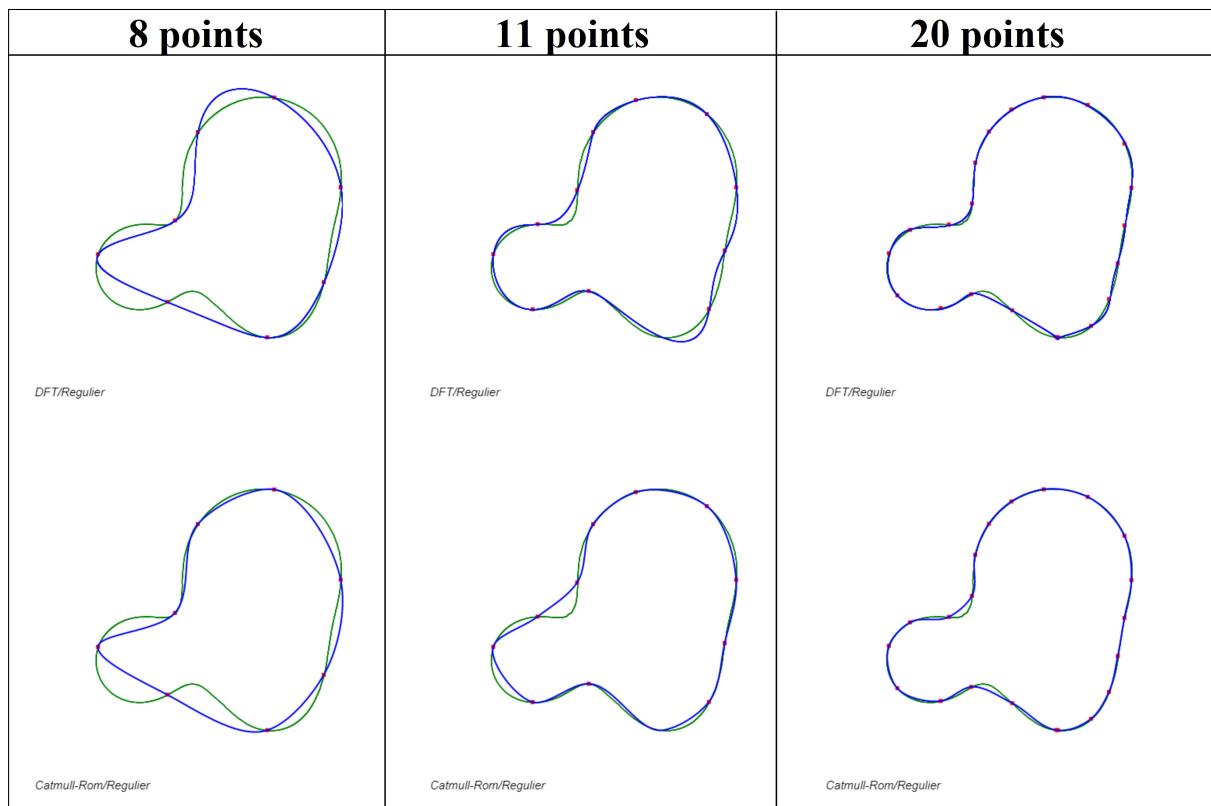


FIGURE 4.4 – Comparaison des méthodes de paramétrisation des courbes.

Chapitre 5

Approximation polygonale d'une courbe discrète

5.1 Modèle

L'approximation polygonale d'une courbe discrète permet d'échantillonner cette dernière. En effet, l'ensemble des sommets du polygone d'approximation sont des points de la courbe, ils représentent alors un échantillon de celle-ci. Pour les futures définitions, nous nous référons au premier octant du plan, c'est-à-dire :

$$0 \leq y \leq x \quad (5.1)$$

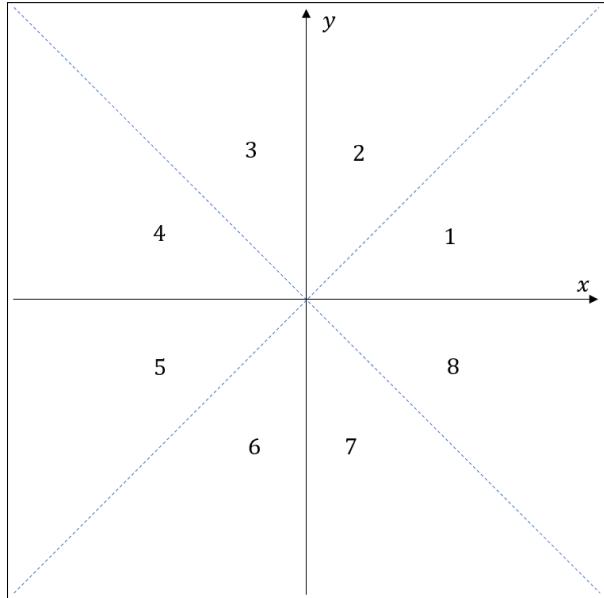


FIGURE 5.1 – Les huit octants du plan.

Les huit octants sont représentés dans la figure 5.1. L'algorithme utilisé est proposé dans [48], il utilise la notion de droite discrète décrite dans [49]. Une droite discrète est définie comme :

$$\mathcal{D}(a, b, \mu, \omega) = \{(x, y) \in \mathbb{Z}^2 \mid \mu \leq ax - by \leq \mu + \omega\}$$

Où a, b, μ, ω sont entiers, b n'est pas nul et a et b sont relativement premiers. $\frac{a}{b}$ est la pente de la droite, μ correspond à la borne inférieure et ω est la largeur de la courbe. Les droites $d_L : ax - by = \mu$ et $d_U : ax - by = \mu + \omega - 1$ sont respectivement appelées droite d'appuis inférieure et supérieure de \mathcal{D} .

La construction du polygone d'approximation s'appuie sur la notion de segment flou. Un ensemble de points $\mathcal{S}f$ est un segment flou d'ordre d si et seulement s'il existe une droite discrète $\mathcal{D}(a, b, \mu, \omega)$ telle que $\mathcal{S}f \subseteq \mathcal{D}(a, b, \mu, \omega)$ et $\frac{\omega}{\max|a|,|b|} \leq d$. La droite \mathcal{D} est appelée droite englobante de $\mathcal{S}f$ et l'ordre du segment flou caractérise la largeur de sa droite englobante. On nomme U_L et L_L les derniers points d'appui supérieur et respectivement inférieur de la droite \mathcal{D} présent dans $\mathcal{S}f$. La figure 5.2 est un exemple de segment flou avec sa droite englobante. Les points entiers en vert sont ceux appartenant à la droite discrète. Ceux contenant un triangle gris sont les points définissant le segment flou. La pente de cette courbe est égale à a/b . Les deux droites d'appui ainsi que leur dernier point d'appui y sont représentés.

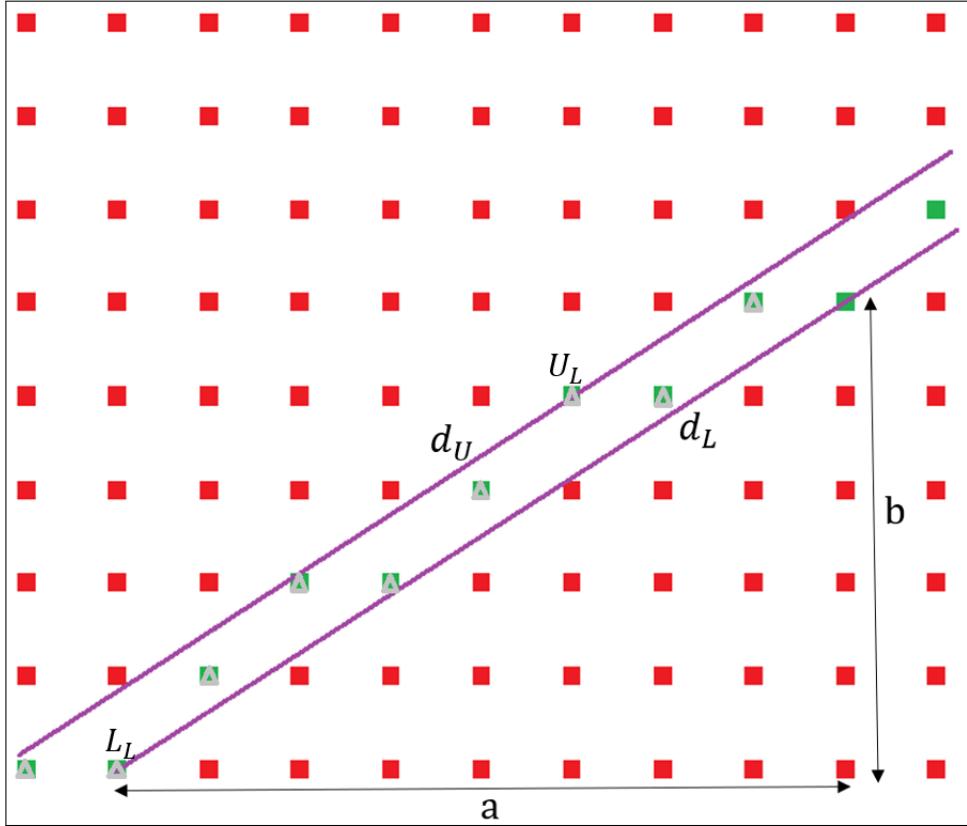


FIGURE 5.2 – Exemple de segment flou et de sa droite discrète englobante.

On appelle vecteur de décalage positif (respectivement négatif) de $\mathcal{D}(a, b, \mu, \omega)$ noté \mathbf{V}_{+1} (respectivement \mathbf{V}_{-1}), un vecteur entre deux points de \mathcal{D} tel que $a(\mathbf{V}_{+1})_x - b(\mathbf{V}_{+1})_y \neq 1$ (respectivement -1) et que $0 \leq (\mathbf{V}_{+1})_x \leq b$ (respectivement $0 \leq (\mathbf{V}_{-1})_x \leq b$). Ces vecteurs sont nécessaires au fonctionnement de l'algorithme. Soit $\mathcal{S}f$ un segment flou, \mathcal{D} sa droite englobante et (x', y') un point en dehors de $\mathcal{S}f$ mais connexe à ce dernier. La question est de savoir s'il existe une droite englobante $\mathcal{D}'(a', b', \mu', \omega')$ d'ordre d du nouveau segment flou $\mathcal{S}f' = \mathcal{S}f \cup (x', y')$. Si oui, quels sont les paramètres a' , b' , μ' et ω' de cette nouvelle droite ? L'algorithme suivant répond à cette problématique. Toutes les démonstrations nécessaires au développement de cet algorithme sont faites dans [49]. L'algorithme est expliqué en français dans [48].

Algorithme 4 : Ajout d'un point à la droite discrète

Data : $\mathcal{S}f$ Le segment flou courant d'ordre $\frac{\omega}{b}$
 $\mathcal{D}(a, b, \mu, \omega)$ La droite courante, ses deux vecteurs de décalage \mathbf{V}_{+1} et \mathbf{V}_{-1} et ses deux points d'appuis U_L et L_L
 $M = (x', y')$ Nouveau point à ajouter

Réultat : La nouvelle droite englobante $\mathcal{D}'(a', b', \mu', \omega')$, ses deux vecteurs de décalage \mathbf{V}'_{+1} et \mathbf{V}'_{-1} et ses deux points d'appuis U'_L et L'_L

```

if  $\mu \leq r(M) < \mu + \omega$  then
|    $(a', b', \mu', \omega') \leftarrow (a, b, \mu, \omega)$ 
else if  $r(M) \leq \mu - 1$  then
|    $(a', b') \leftarrow$  les coordonnées de  $\frac{x' - (\mathbf{V}_{-1})_x}{b} \begin{bmatrix} b \\ a \end{bmatrix} + \mathbf{V}_{-1}$ 
|    $\mu' \leftarrow a'x' - b'y'$ 
|    $\omega' \leftarrow a'L_{Lx} - b'L_{Ly} - \mu' + 1$ 
|    $\mathbf{V}'_{-1} \leftarrow \begin{bmatrix} b' - b \\ a' - a \end{bmatrix} + \frac{b}{b'} \begin{bmatrix} b' \\ a' \end{bmatrix}$ 
|    $\mathbf{V}'_{+1} \leftarrow \begin{bmatrix} b \\ a \end{bmatrix} - \frac{b}{b'} \begin{bmatrix} b' \\ a' \end{bmatrix}$ 
|    $U_L \leftarrow M$ 
|    $L_L \leftarrow$  Dernier point de  $\mathcal{S}f$  appartenant à  $d_L$ 
else if  $r(M) \geq \mu + \omega$  then
|    $(a', b') \leftarrow$  les coordonnées de  $\frac{x' - (\mathbf{V}_{+1})_x}{b} \begin{bmatrix} b \\ a \end{bmatrix} + \mathbf{V}_{+1}$ 
|    $\mu' \leftarrow a'U_{Lx} - b'U_{Ly}$ 
|    $\omega' \leftarrow a'x' - b'y' - \mu' + 1$ 
|    $\mathbf{V}'_{-1} \leftarrow \begin{bmatrix} b \\ a \end{bmatrix} - \frac{b}{b'} \begin{bmatrix} b' \\ a' \end{bmatrix}$ 
|    $\mathbf{V}'_{+1} \leftarrow \begin{bmatrix} b' - b \\ a' - a \end{bmatrix} + \frac{b}{b'} \begin{bmatrix} b' \\ a' \end{bmatrix}$ 
|    $U_L \leftarrow$  Dernier point de  $\mathcal{S}f$  appartenant à  $d_U$ 
|    $L_L \leftarrow M$ 

```

En parcourant notre courbe discrète et en fixant un ordre d'erreur d , on peut générer un ensemble de segments flous tels que tous les segments flous couvrent tous les points de notre courbe. Ensuite, il suffit de récupérer les premiers points de chaque segment flou pour obtenir les sommets de notre polygone d'approximation. L'algorithme suivant permet de faire cette tâche.

Algorithme 5 : Calculer le polygone d'approximation

Data : \mathcal{C} Une suite de points connexes représentant notre courbe discrète
 d Ordre autorisé des segments flous

Réultat : L Liste segments flous composant la courbe \mathcal{C}

```

L ← []
a ← 0, b ← 1, μ ← 0
ω ← b,  $M_c \leftarrow (0, 0)$ ,  $L_L \leftarrow (0, 0)$ ,  $U_L \leftarrow (0, 0)$ 
 $V_{-1} \leftarrow (0, 1)$ ,  $V_{+1} \leftarrow (0, -1)$ , estSegment ← true
estMêmeOctant ← true, fin ← false
M ← Premier point de  $\mathcal{C}$ 
while !fin do
    segmentCourant ← []
    while estSegment et estMêmeOctant et !fin do
         $M_{last} \leftarrow M$ 
        M ← Prochain point de  $\mathcal{C}$ 
         $M_c \leftarrow$  Image de  $M$  dans le première octant a
        estMêmeOctant ← Est-ce que  $M_c$  est dans le premier octant
         $a_{last} \leftarrow a$ ,  $b_{last} \leftarrow b$ ,  $\mu_{last} \leftarrow \mu$ ,  $\omega_{last} \leftarrow \omega$ 
        if estMêmeOctant then
            ajouterPointSf( $M_c$ ) ; // Voir l'algorithme décrit en dessus
            estSegment ←  $\frac{\omega}{b} \leq d$ 
            if estSegment then
                | segmentCourant.push(M)
            end
        end
        fin ←  $\mathcal{C}$  est complètement parcourue
    end
    L.push(segmentCourant)
    a ← 0, b ← 1, μ ← 0
    ω ← b,  $M_c \leftarrow (0, 0)$ ,  $L_L \leftarrow (0, 0)$ ,  $U_L \leftarrow (0, 0)$ 
     $V_{-1} \leftarrow (0, 1)$ ,  $V_{+1} \leftarrow (0, -1)$ , estSegment ← true
    estMêmeOctant ← true,
end

```

a. Voir section 5.2.

5.2 Implémentation

Pour chaque nouveau segment flou créé durant l'algorithme, il faut savoir dans quel octant le segment se trouve. Pour se faire, la position moyenne des N points suivants est calculée. Le futur octant est celui du point moyen. Étant donné que tous les calculs se font dans le premier octant, il faut pouvoir transformer les points d'un octant dans le premier octant. La transformation de n'importe quel octant au premier se fait au moyen de huit symétries. Ces symétries sont les mêmes que celles du groupe de symétrie du carré. Ces transformations peuvent être représentées sous la forme matricielle. Soit $o_i(\mathbf{P})$ la transformation du point \mathbf{P} du i -ième octant au premier. On peut écrire : $o_i(\mathbf{P}) = M_i \mathbf{P}$ où $M_i \in \mathcal{M}_{2,2}$. Voici les huit matrices M_i :

$$M_1 = I_2, M_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, M_3 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, M_4 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$M_5 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, M_6 = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}, M_7 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, M_8 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

En ayant connaissance de l'octant du segment flou effectif, il est possible d'appliquer la transformation à tous les futurs points. Ensuite, en utilisant l'équation 5.1, on définit si le point transformé est bien dans le premier octant. Cette valeur booléenne est ensuite placée dans la variable `estMêmeOctant`.

5.3 Résultats

Dans la figure 5.3, l'algorithme est exécuté plusieurs fois en faisant varier la valeur d . Les points rouges représentent les sommets du polygone, les segments verts, les segments du polygone et la courbe bleue, la courbe à approximer. Les différentes parties de la figure sont triées par rapport à la valeur de d utilisée. L'image (a) utilise un d fixé à dix, la (b) à trois cent, la (c) à cinq cent et la (d) à mille. Dans l'image (a), le polygone se confond avec la courbe. Dans les images (b) et (c), le polygone est presque le même. Dans l'image (d), la valeur d est si petite que l'approximation polygonale ignore complètement la boucle à gauche.

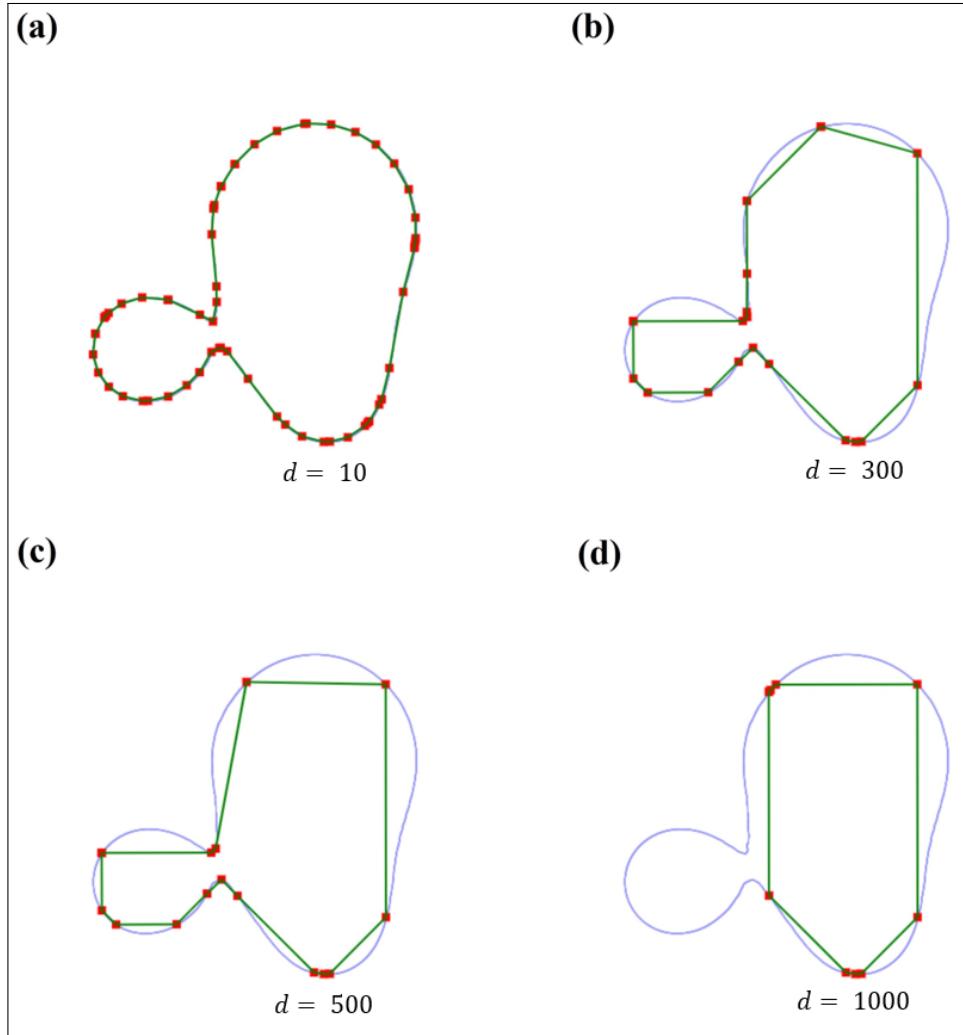


FIGURE 5.3 – Comparaison de l'approximation polygonale avec différentes valeurs de d .

Chapitre 6

Reconstruction des courbes discrètes

6.1 Modèle

La reconstruction des courbes permet de passer de l'équation paramétrique de ces dernières, obtenue en utilisant les caractéristiques de la courbe, à sa représentation sur une image binaire. Cette image binaire sera ensuite utilisée pour le rendu sur le plan et sur la surface. Deux méthodes ont été développées pour reconstruire les courbes :

1. Reconstruction au pixel près ;
2. Reconstruction en traçant des lignes.

La première méthode permet, comme son nom l'indique, de reconstruire la courbe au pixel prêt. Il n'y a donc pas de perte d'information lors de la reconstruction. Cette méthode est plus gourmande que la seconde. Pour se faire, on va fixer un pas constant Δt tel que : $\forall t \geq 0, \|\tilde{\mathbf{C}}(t + \Delta t) - \tilde{\mathbf{C}}(t)\|_2 \leq \text{un pixel}$. En itérant par pas de Δt sur l'équation paramétrique, on s'assure donc de ne pas sauter de pixel. Pour fixer un tel pas, il faut définir un majorant m de la norme de la dérivée de l'équation paramétrique de la courbe. Un tel majorant permet de définir le pas comme : $\Delta t = 1/m$. Cette méthode n'est pas utilisée pour Catmull-Rom.

La deuxième méthode fonctionne de la même manière que la première méthode, en fixant un pas d'itération constant. Cependant, elle n'assure pas que la courbe soit reconstruite au pixel prêt. Pour relier les points reconstruits entre eux, un segment est dessiné. On définit le pas par le nombre de points que l'on veut échantillonner. Soit N le nombre de points voulus, alors le pas d'itération est : $\Delta t = T/N$.

6.2 Implémentation

L'algorithme de la première méthode est implémenté côté JS. Une boucle `for` permet de dessiner, pixel par pixel dans un canevas JS. Ce canevas est ensuite utilisé comme texture d'entrée dans les shaders qui permettent d'utiliser les modèles de rendus. Le majorant est défini comme $m = \frac{2\pi}{T} \sum_{k=-N/2}^{N/2} |k| \|c_k\|$ où T est la période de la courbe. La démonstration est fait dans l'annexe A.

L'algorithme de la deuxième méthode est implémenté côté JS. Les segments sont construits en utilisant des canevas JS. Dans les exemples de code venant de [50], Jeanmonod Quentin propose une implémentation côté GPU de cet algorithme.

6.3 Résultats

La figure 6.1 compare le résultat issu de ces deux algorithmes. Le deuxième algorithme utilise trois N différents. L'image (a) de la figure correspond au résultat obtenu en utilisant la première méthode. Elle est donc identique à la courbe de base. Les autres images, (b), (c) et (d) correspondent aux résultats

obtenus en utilisant la deuxième méthode et en fixant N à respectivement : dix, cent et mille. On peut voir que déjà après cent itérations, on ne voit presque plus les segments qui composent la courbe. Le tableau suivant compare les temps de calcul permettant la reconstruction de ces courbes. La paramétrisation utilisée est celle de la DFT.

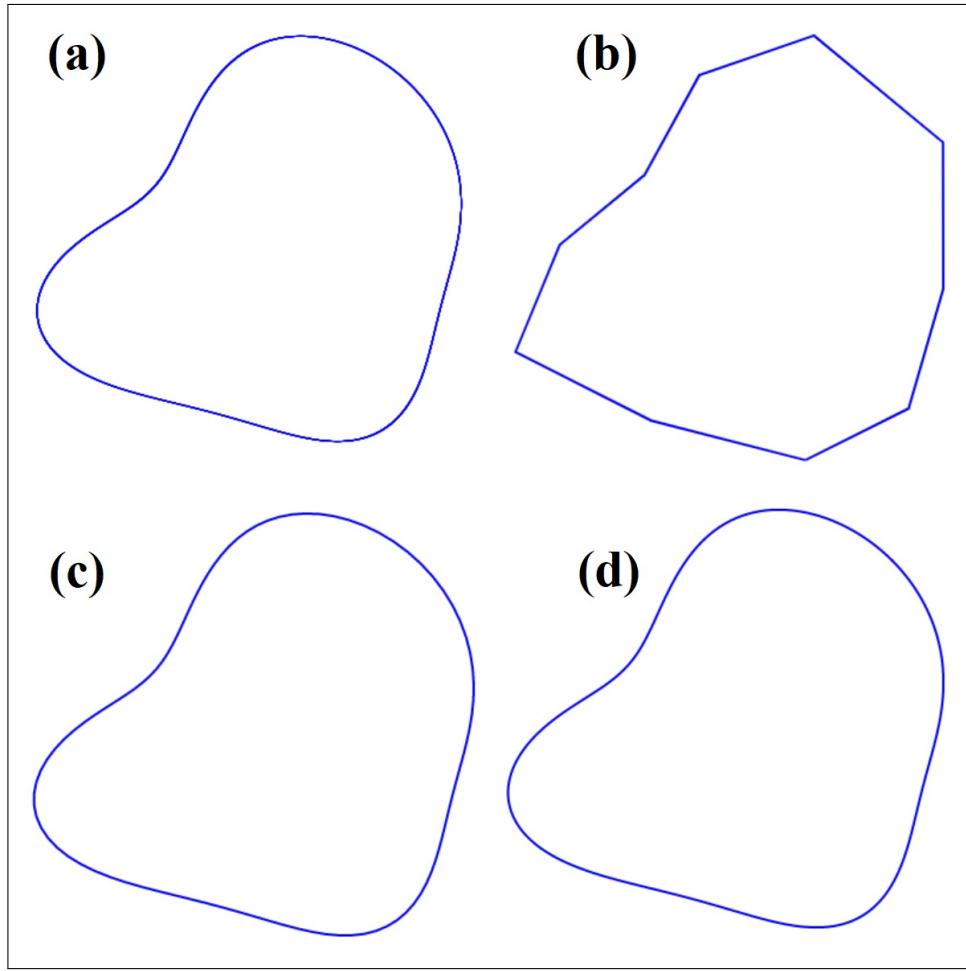


FIGURE 6.1 – Comparaison des méthodes de reconstruction des courbes.

Méthode	Pixel près	Régulier (10)	Régulier (100)	Régulier (1000)
Temps de reconstruction [ms]	9.5	0.1	0.2	2.3

Comme prévu, la méthode au pixel près est la plus lente. En revanche, la deuxième méthode, en utilisant cent points, offre un bon compromis entre erreur de reconstruction et temps de calcul.

Chapitre 7

Modèles de rendu

7.1 Algorithme de *raycasting*

Le *raycasting* est un algorithme permettant le rendu d’images à trois dimensions. Dans le cadre de ce projet, le rendu se fait dans un cube placé au milieu de la scène. La position de l’œil sera reculée de façon à avoir l’intégralité du cube dans le champ de vision. Ensuite, pour chaque pixel, on va imaginer un rayon qui part de l’œil et passe par le pixel. Si ce rayon intersecte le cube, on va calculer le point d’entrée du rayon dans le cube et le point de sortie. Le segment formé entre le point d’entrée et point de sortie est celui qui sera parcouru par l’algorithme de *raycasting*. On va subdiviser régulièrement ce segment afin de pouvoir le parcourir itérativement. Pour implémenter les différents modèles de rendu, il faut implémenter le comportement voulu à l’intérieur de la boucle.

Dans [31], cet algorithme de *raycasting* implémenté sur le GPU a été développé. Cet algorithme permet de traverser des *datacube* à trois dimensions afin d’en afficher les surfaces de niveau. Cet algorithme a été développé en utilisant l’API OpenGL et, par ce fait, en utilisant le langage de programmation GLSL. Le programme développé durant ce travail de Bachelor nécessite l’utilisation du *raycasting* pour le rendu trois dimensions des courbes de niveau décrites dans la section 1.3.8. Le code du *raycasting* a été transformé pour pouvoir être utilisé dans le cadre de ce projet. L’intégralité du code source de l’algorithme est contenu dans deux fichiers : un fragment shader et un vertex shader.

Cependant, ces shaders ont été écrits pour l’API OpenGL. Cette dernière diffère de l’API utilisée dans le cadre de ce projet sur certains points et un simple copier-coller de ces shaders ne suffit pas à les faire fonctionner. L’API que nous utilisons est WebGL. Elle est une spécification d’OpenGL ES et non d’OpenGL. OpenGL ES a été conçue pour les plateformes mobiles ou embarquées [51] et de ce fait, elle est plus limitée. Une transformation du code est donc nécessaire. En plus de modifier la logique du code, il est refactorisé pour respecter les conventions utilisées dans ce projet. Voici les deux points qui dans notre cas, nécessitent une transformation du code :

1. Certaines variables uniformes intégrées n-existent pas ;
2. Les expressions non constantes dans les boucles **for** sont interdites.

Pour le premier cas, les variables uniformes intégrées utilisées qui posent problème sont les variables représentant les matrices inverses des matrices *ModelView* et *ModelViewProjection*. Ce sont des matrices de taille quatre par quatre. Dans le développement d’application graphique, les matrices *ModelView* et *ModelViewProjection* sont indispensables. OpenGL calcule automatiquement leur inverse pour que le développeur n’ait pas besoin de le faire lui-même si il a besoin de les utiliser. L’impossibilité d’utiliser ces variables oblige à calculer l’inverse de ces deux matrices dans le programme JS et de les passer aux deux shader comme des variables uniformes.

Le second cas est dû au besoin d’optimisation de WebGL. Pour optimiser l’exécution du programme sur le GPU, la valeur qui définit la condition d’arrêt d’une boucle doit être une constante. Cette restriction permet à WebGL d’avoir connaissance, lors de la compilation des shaders, du nombre d’itérations

maximum et d'ainsi optimiser le programme. Pour pallier ce problème, il faut définir un nombre d'itération maximum constant tel que l'on itère sur tous les cas possibles. On pourra alors ensuite quitter la boucle grâce à l'instruction `break` lorsque la condition, pouvant cette fois contenir des expressions non constantes car placée dans la boucle, est vérifiée. Évidemment, cette valeur doit être le plus petit possible. Le choix de cette valeur dépend alors du contexte, dans notre cas, la seule boucle `for` utilisée permet de traverser le cube en utilisant l'algorithme de *raycasting*. Le nombre d'itérations représente le nombre de pas qu'il est nécessaire de faire pour qu'un rayon donné traverse le cube de part et d'autre. La longueur d'un pas est égale à celle d'un voxel du *datacube* multiplié par un facteur de précision p compris dans l'intervalle $[0, 1]$. Le nombre d'itérations maximum survient dans le cas où l'on traverse le cube dans sa diagonale. Soit c la longueur des côtés du cube comptée en voxel. La précision étant variable¹, il faut prendre le cas où elle est maximale, c'est à dire $p = 1$. Dans ce cas, la valeur d'itération maximale est égale à $\lceil \sqrt{3}c \rceil$.

7.2 Rendu des courbes implicites

Ce rendu ne va pas afficher les courbes de niveau d'un seul niveau, mais les courbes de niveau d'un ensemble de différents niveaux. Ceci permet de mieux visualiser l'information. Pour se faire, il faut définir une variable supplémentaire : δ . Cette variable représente la différence qu'il y a entre deux niveaux. Pour un iso et un δ donnés, l'ensemble des niveaux est $\{iso + k\delta \mid 0 \leq iso + k\delta \leq 1, k \in \mathbb{Z}\}$.²

Pour chacun de ces niveaux, il faut appliquer l'algorithme de construction des courbes discrètes proposé au chapitre 3. Le résultat de cet algorithme est une image binaire où les pixels dont l'intensité n'est pas nulle sont les pixels appartenant aux courbes. Les images résultantes de l'application de l'algorithme sur tous les niveaux doivent être additionnées.³ L'image finale contient toutes les courbes de niveau. Pour mettre en évidence la courbe représentant le niveau iso , elle est dessinée dans une couleur différente. Les images (d) et (e) de la figure 7.1 montrent les différences obtenues en faisant varier δ . L'image utilisée comme entrée de l'algorithme est l'image (a) de la même figure.

L'image finale est passée par le biais d'une texture au GPU puis projetée sur un carré. Ce carré se trouve dans l'espace à trois dimensions, l'utilisateur a la possibilité de le faire tourner. Le programme permet aussi de zoomer puis se déplacer sur la texture. Un utilisateur peut donc se concentrer sur une région de l'image. Les images (b), (c), (d) et (f) de la figure 7.1 montrent le résultat de ce rendu en appliquant différentes rotations au carré et différents zooms. Pour améliorer la qualité du rendu, une dilatation morphologique puis un filtre gaussien sont appliqués à l'image contenant les courbes. De cette façon, les courbes sont plus épaisses et un effet d'anti-aliasing est appliqué.

7.3 Rendu de la surface

Comme expliqué à la section 7.1, l'algorithme de *raycasting* permet, pour chaque pixel de l'écran, de lancer un rayon afin de traverser un espace de voxel et d'ainsi calculer quelle doit être la couleur du pixel. Cet algorithme est utilisé dans le rendu de la surface. Pour chaque point (x, y, z) de la surface, la hauteur de cette dernière est proportionnelle à l'intensité du *datacube* utilisé en entrée au point (x, y) . La figure 7.2 compare l'affichage d'un *datacube* sur une image et le rendu du *datacube* en utilisant une surface. Les images (b), (c) et (d) jouent sur un facteur d'échelle appelé λ pour changer la hauteur de la surface. Les valeurs de λ sont respectivement 0.3, 0.6 et 1. Pour mettre en évidence la hauteur, des palettes de couleurs sont utilisées. L'implémentation des palettes de couleur est expliquée à la section 7.5.

À chaque itération de la boucle de *raycasting*, les deux premières coordonnées (x, y) des coordonnées (x, y, z) du point courant dans l'espace de voxel sont utilisées pour récupérer la valeur du *datacube* en (x, y) . Ceci est possible car le *datacube* est passé comme une texture au shader. Ensuite, la différence

1. L'utilisateur a le contrôle sur la précision de l'algorithme durant l'exécution du programme.
 2. Par soucis de simplicité, nous supposons que l'intensité de l'image varie entre zéro et un. En réalité, elle est encodée sur huit bits, son intensité est donc comprise dans l'intervalle entier $[0, 255]$.
 3. Dans le sens d'un ou logique au niveau de bits.

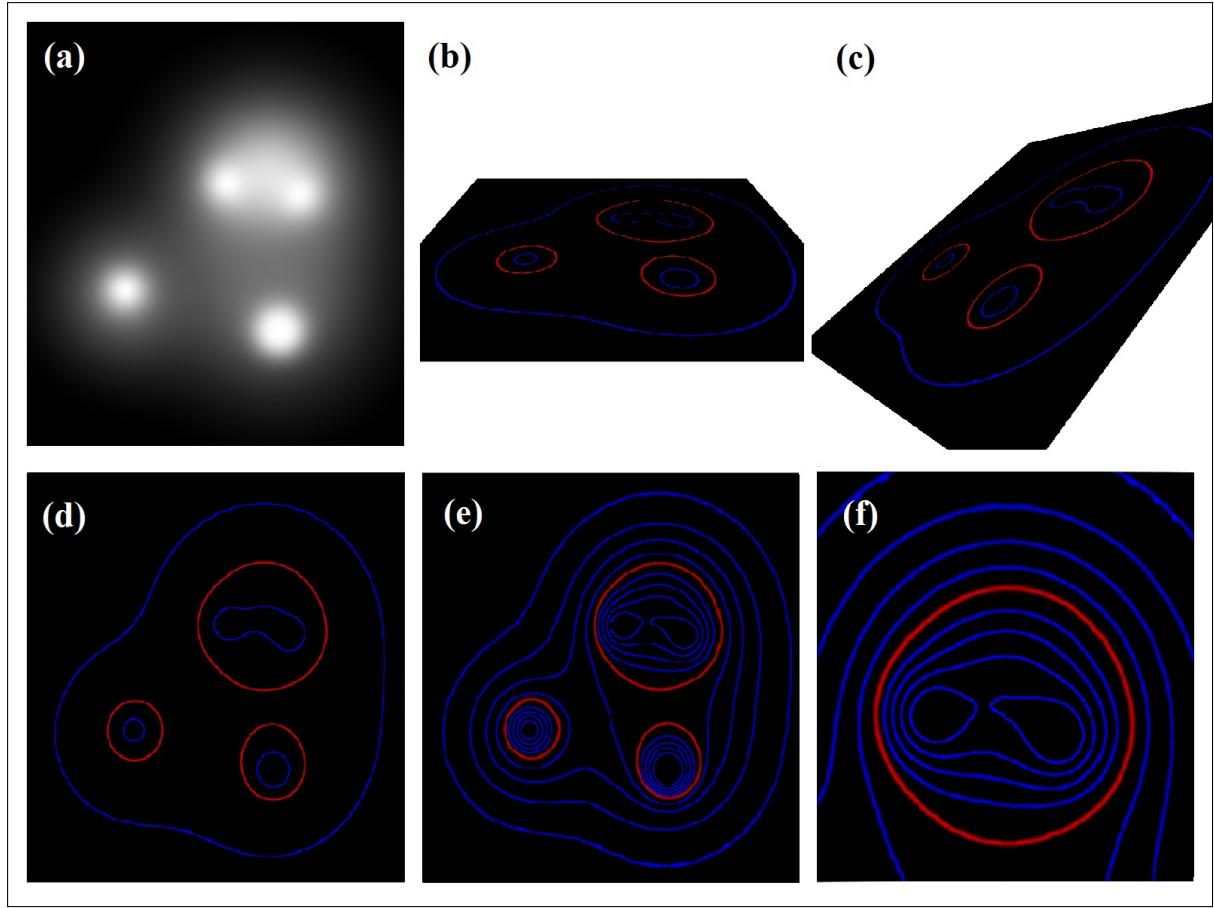


FIGURE 7.1 – Différentes prises de vues du rendu des courbes avec l'image de base.

signée entre le niveau du point courant z et la valeur z_{data} du *datacube* en (x, y) multiplié par le facteur d'échelle λ est calculée. La fonction qui permet de calculer la différence signée est appelée Signed distance function (SDF, Fonction de distance signée). Si la SDF est égale à zéro, le pixel qui a lancé le rayon doit dessiner la surface, sinon, il ne dessine rien. Évidemment, dans le monde discret, il est impossible d'assurer que cette SDF sera nulle. Ce problème est similaire à celui de la construction des courbes de niveau. Pour s'assurer qu'un pixel qui devrait dessiner la surface la dessine bien, il faut regarder le signe la SDF. Si à la i -ième itération le signe est différent qu'à la $(i - 1)$ -ième itération, alors, la SDF doit être nulle entre ces deux itérations et on peut considérer que le pixel dessine la courbe.⁴ Dans le cas présent, la SDF est définie comme : $(z, z_{data}) \mapsto z - \lambda z_{data}$. L'algorithme est exécuté sur le GPU. Voici l'algorithme en question :

4. Vrai que si l'on considère la SDF continue. La SDF est continue si et seulement si le champ scalaire représenté par le *datacube* est continu.

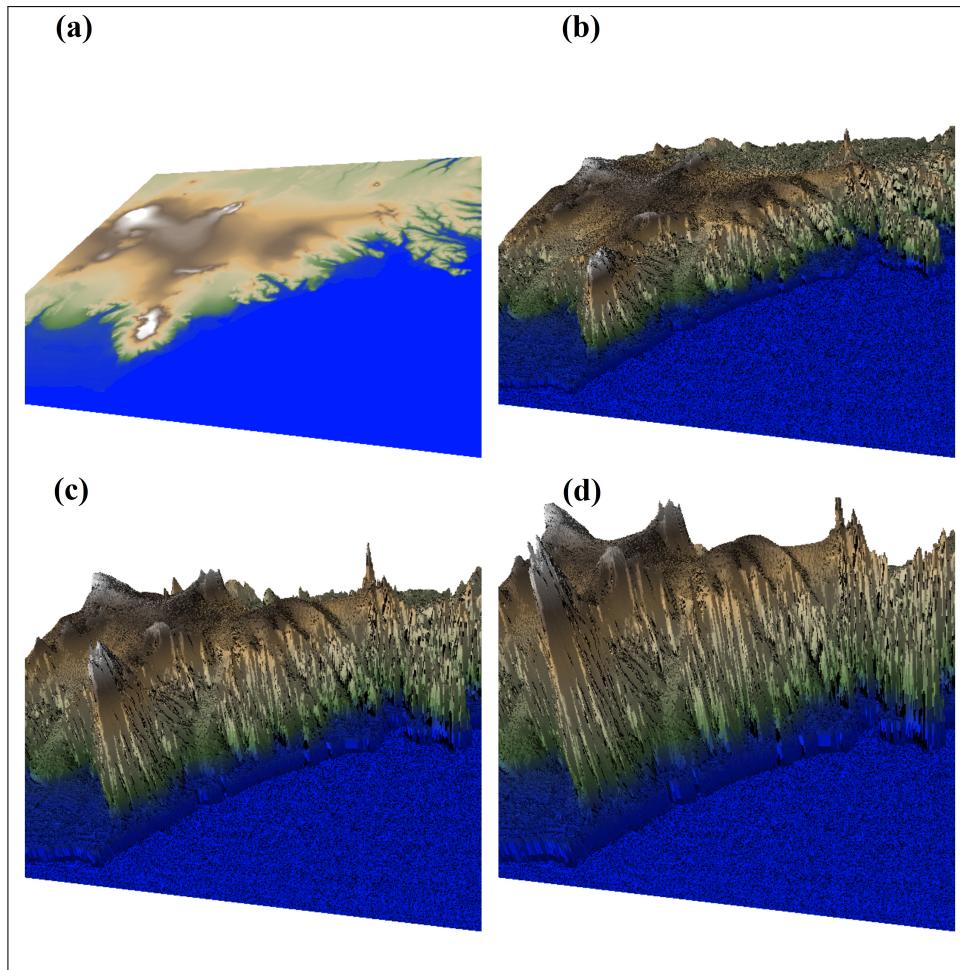


FIGURE 7.2 – Comparaison du rendu de la surface en utilisant des facteurs de hauteurs différentes.

Algorithme 6 : Test si le pixel doit dessiner la surface

Data : `data` Matrice à deux dimensions représentant le *datacube*
 λ Échelle de hauteur (comprise entre zéro et un)

Réultat : Un booléen `dessineSurface` exprimant si le pixel doit dessiner la surface. Si c'est le cas, les coordonnées (x_s, y_s, z_s) du point de la surface sont aussi renvoyées.

```

 $s_{old} \leftarrow 0$ 
 $s \leftarrow 0$ 
dessineSurface  $\leftarrow$  false
while Appliquer l'algorithme du raycasting vu à la section 7.1 ou arrêter si dessineSurface est vrai. do
     $(x, y, z) \leftarrow$  Coordonnées du voxel actuel
     $s_{old} \leftarrow s$ 
     $s \leftarrow \text{sgn}(SDF(z, data[x, y]))$ 
    if  $ss_{old} \leq 0^a$  then
        dessineSurface  $\leftarrow$  true
         $(x_s, y_s, z_s) \leftarrow (x, y, z)$ 
    end
end
```

a. Le fait que l'inégalité ne soit pas stricte vient du fait qu'en GLSL, le signe de zéro est défini comme zéro. Si la SDF vaut zéro, le pixel dessine la courbe. Il faut donc considérer ce cas.

En plus des palettes de couleurs, un éclairage diffus est calculé afin de mettre le relief en évidence. Le calcul de l'éclairage diffus se fait à partir de la normale de la surface. Deux modèles ont été développés pour calculer la normale. Les deux modèles font une approximation de la normale en fonction des quatres voisins (4-connexes) du point courant sur la surface. Soit (i, j) les coordonnées du point entier courant et M une matrice de taille $n \times n$ représentant le *datacube*. L'intensité en chaque point du *datacube* étant normalisé entre zéro et un, il faut faire de même avec les coordonnées (i, j) . Il faut donc les diviser par la taille de la matrice. On peut maintenant définir les points voisins $\mathbf{P}_{1,2,3,4}$ et le point \mathbf{P}_0 courant dans l'espace :

$$\begin{aligned}\mathbf{P}_0 &= \left(\frac{i}{n}, \frac{j}{n}, m_{i,j}\right), \mathbf{P}_1 = \left(\frac{i+1}{n}, \frac{j}{n}, m_{(i+1),j}\right), \\ \mathbf{P}_2 &= \left(\frac{i}{n}, \frac{j+1}{n}, m_{i,(j+1)}\right), \mathbf{P}_3 = \left(\frac{i-1}{n}, \frac{j}{n}, m_{(i-1),j}\right), \\ \mathbf{P}_4 &= \left(\frac{i}{n}, \frac{j-1}{n}, m_{i,(j-1)}\right)\end{aligned}$$

Voici les équations respectives des deux modèles permettant de trouver le vecteur normal \mathbf{N} . Pour simplifier les formules, on écrira \mathbf{P}^0 le vecteur \mathbf{P} normalisé.

1. $\mathbf{N} = (\mathbf{P}_1 - \mathbf{P}_3) \times (\mathbf{P}_2 - \mathbf{P}_4)$;
2. $\mathbf{N} = (\mathbf{P}_1 - \mathbf{P}_0)^0 + (\mathbf{P}_2 - \mathbf{P}_0)^0 + (\mathbf{P}_3 - \mathbf{P}_0)^0 + (\mathbf{P}_4 - \mathbf{P}_0)^0$.

La normale étant définie, on peut calculer en fonction de cette dernière et de la direction de la lumière un facteur γ , compris entre zéro et un, représentant l'intensité diffuse de la lumière en un point. Ce facteur multipliera chaque canal de la couleur finale du pixel, de cette façon, plus le facteur est petit, plus le couleur sera plus sombre. Pour éviter de trop assombrir l'image, on utilise une transformation pour que γ varie entre un demi et un. De la même façon que dans le calcul de la composante diffuse de l'illumination de Young, il faut calculer le produit scalaire entre le vecteur lumière \mathbf{L} et le vecteur normal \mathbf{N} . Ces deux vecteurs doivent être normalisés.

$$\gamma = \frac{\langle \mathbf{N}^0 | \mathbf{L}^0 \rangle + 1}{2}$$

La figure 7.3 compare les deux modèles. L'image (a) représente le rendu utilisant le premier modèle et la (b) celui qui utilise le deuxième modèle. On remarque que dans les deux cas, les bords droits des collines sont assombris, alors que les bords gauches sont éclairés. Ceci est dû à la direction de la lumière. Sur les régions planes, on remarque des cavités. Elles sont dues à l'orientation de la surface à ces endroits-là. Bien que l'intensité en tout point du datacube est normalisée, elle n'en reste pas moins discrétisée. Dans notre cas, l'intensité est codée sur un canal de huit bits. Au lieu d'appartenir à l'ensemble $\{0, \dots, 2^8 - 1\}$ représenté par les huits bits, l'intensité normalisée du datacube appartient à l'ensemble $\frac{\{0, \dots, 2^8 - 1\}}{2^8 - 1}$ qui est aussi un ensemble discret. De ce fait, la hauteur de la surface est aussi discrète. Au point où l'on passe d'une hauteur à une autre, la surface devient verticale et son vecteur normal ne possède donc plus de composante verticale. Ceci provoque un changement brusque du facteur γ . C'est ce phénomène qui provoque ces cavités. Le rendu en utilisant le premier modèle est plus propre, mais il a l'inconvénient de demander plus de calcul.

7.4 Rendu des courbes sur une surface

Le modèle expliqué à la section 7.3 n'est pas directement utilisé dans le cadre de ce projet, car le but reste l'affichage des courbes de niveau. Dans cette section nous allons voir comment afficher les courbes de niveau sur cette surface. La combinaison des courbes et des surfaces permet de mettre le relief en évidence tout en affichant les courbes. La méthode développée pour parvenir à ce résultat est naïve, elle se base uniquement sur l'image binaire résultante du rendu des courbes de niveau sur un plan présenté à la section 7.2 et non pas l'équation paramétrique de ces dernières. La méthode consiste à projeter l'image contenant les courbes sur la surface. Pour cela, il est nécessaire que dans l'algorithme de rendu de la surface, les pixels devant dessiner la surface, vérifie qu'au même point sur l'image binaire, l'intensité est bien nulle avant de calculer leur couleur par rapport à la palette. Si cette intensité n'est pas nulle, le

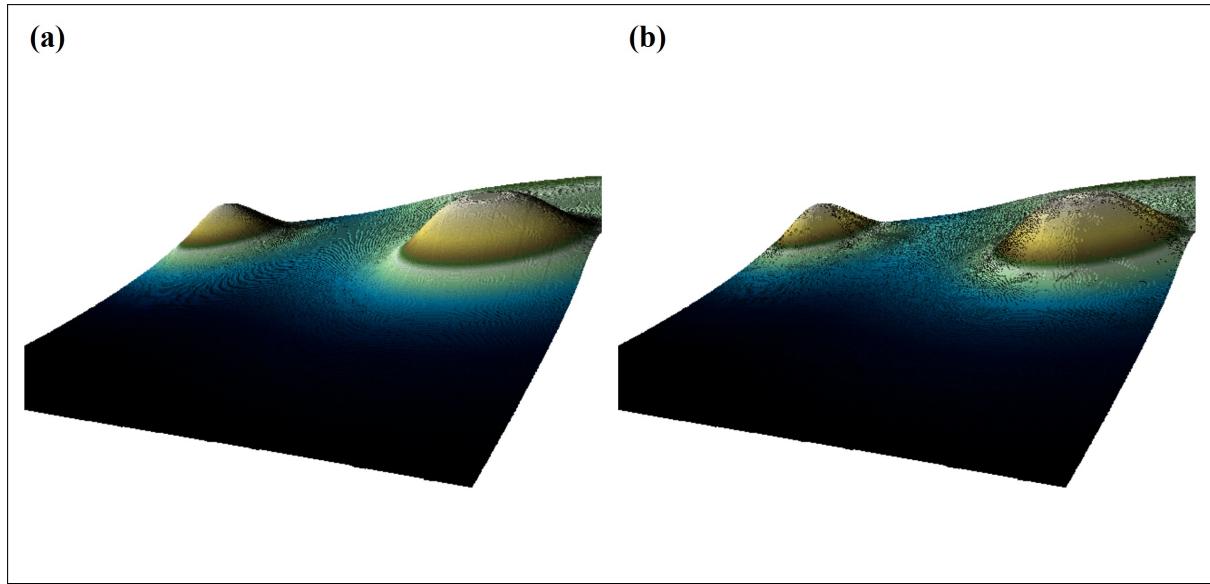


FIGURE 7.3 – Comparaison des deux modèles de calcul de normale.

pixel doit dessiner une courbe et une couleur spéciale lui est affectée.

La méthode proposée possède un problème, bien que l’algorithme de construction des courbes assure que la largeur de ces dernières soit d’un pixel, la projection de l’image binaire sur la surface ne permet pas de contrôler la largeur vis-à-vis du nouvel axe représentant la hauteur ajoutée lors d’un rendu à trois dimensions. De ce fait, il est possible qu’une partie verticale d’un mur soit entièrement dessiné comme faisant partie d’une courbe de niveau. La figure 7.4 illustre ce problème. Une tranche de la surface est représentée en bleu et l’image binaire correspondant à cette tranche se trouve en dessous de celle-ci. Les cases noires représentent des pixels non nuls. Les zones traitillées en rouge sont les parties de la surface considérées comme dans la courbe. La surface intersecte deux fois la hauteur représentée par l’intensité *iso*. Lors de la première intersection, la surface n’est pas très raide. Par conséquent, la largeur de la courbe sur la surface est presque d’un pixel. En revanche, lors de la deuxième intersection, la surface est très raide. On peut voir que toute la partie raide est considérée comme dessinant la courbe. En regardant cette surface sous cet angle, la courbe est bien plus large que prévu. La figure 7.5 compare ce rendu sur deux images, la première utilise des données telles que la surface ne soit pas très raide alors que la deuxième utilise la carte de hauteurs de l’Islande rendue dans la figure 7.2. Dans ce deuxième exemple, certaines parties des courbes semblent plus larges. Cela est dû à la variation de la pente de la surface en ces points.

7.5 Palette de couleur

L’utilisation d’une palette de couleur au lieu d’une échelle de gris pour représenter une valeur permet de mettre cette dernière davantage en évidence. Le programme développé offre la possibilité à l’utilisateur de choisir une palette pour afficher les valeurs du champ scalaire décrit par la texture. Un ensemble de vingt palettes est mis à disposition. Ces palettes sont décrites par la version 4.1.3 du standard Generic Mapping Tools (GMT) [52]. La figure 7.6 montre une mise en application des vingt palettes différentes dans un rendu généré par l’application. Ce rendu est celui des courbes de niveaux sur une surface et les courbes de niveau ont été enlevées pour visualiser complètement les palettes. En dessous de chaque exemple, le nom de la palette en question est précisé.

Les couleurs sont stockées dans un fichier JavaScript Object Notation (JSON). Ce fichier décrit un dictionnaire où les clés sont les noms des palettes, encodés comme chaîne de caractères, et les valeurs, un tableau contenant toutes les composantes des couleurs mises côte à côte, encodées comme un entier

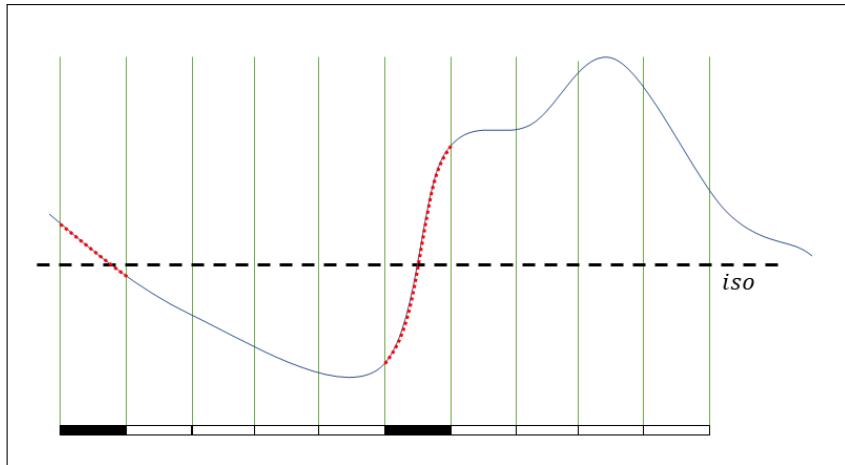


FIGURE 7.4 – Problème dû à la méthode d'intégration des courbes de niveau sur la surface à trois dimensions.

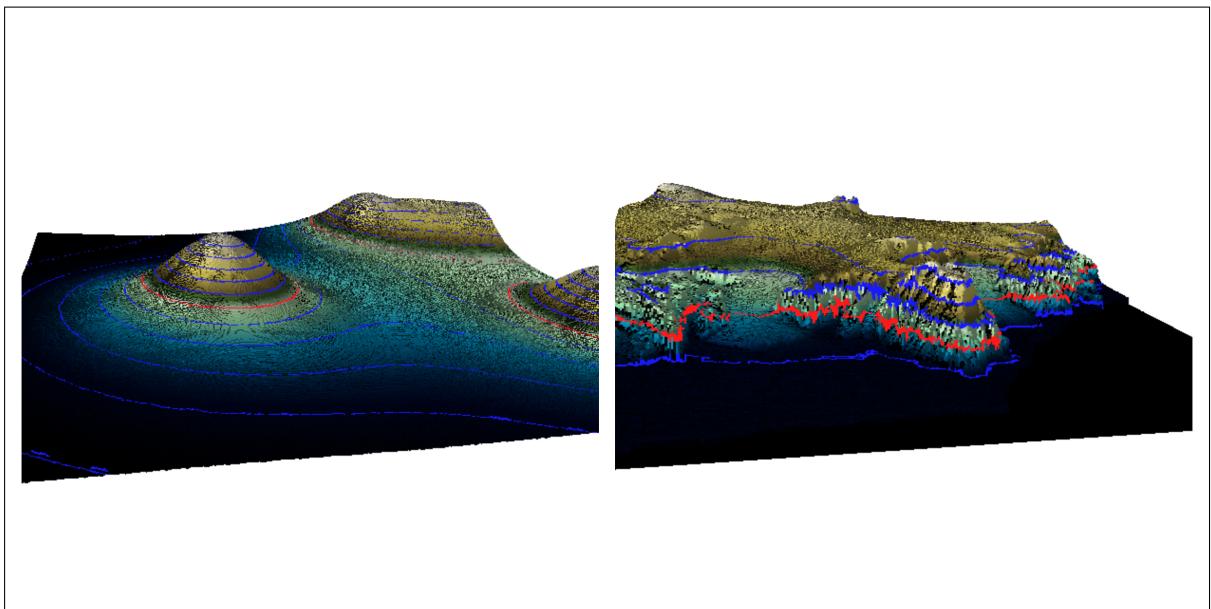


FIGURE 7.5 – Rendu des courbes sur les surfaces à trois dimensions appliquées à deux images différentes.

compris entre zéro et deux cents cinquante-cinq. L'annexe B représente une partie de ce fichier. Pour les deux premières palettes, les couleurs correspondantes sont affichées.

Pour obtenir l'effet dégradé présent dans la figure 7.6, une interpolation linéaire est effectuée entre les différentes couleurs définies par la palette. Voici les différentes étapes qui permettent d'interpoler et d'utiliser une palette :

1. Les couleurs sont récupérées du JSON ;
2. Un tableau à une dimension contenant les différentes composantes des couleurs de la palette mises côte à côte est créé côté JS ;
3. Ce tableau est passé à WebGL pour générer une texture à une dimension de taille égale au nombre de couleur de la palette ;
4. Définir l'interpolation linéaire comme filtre de réduction de la texture ;
5. Ensuite, pour faire coïncider une valeur normalisée x à une couleur de la palette, il suffit d'accéder à la couleur de la texture en utilisant comme coordonnée $uv : (0.5, x)$.

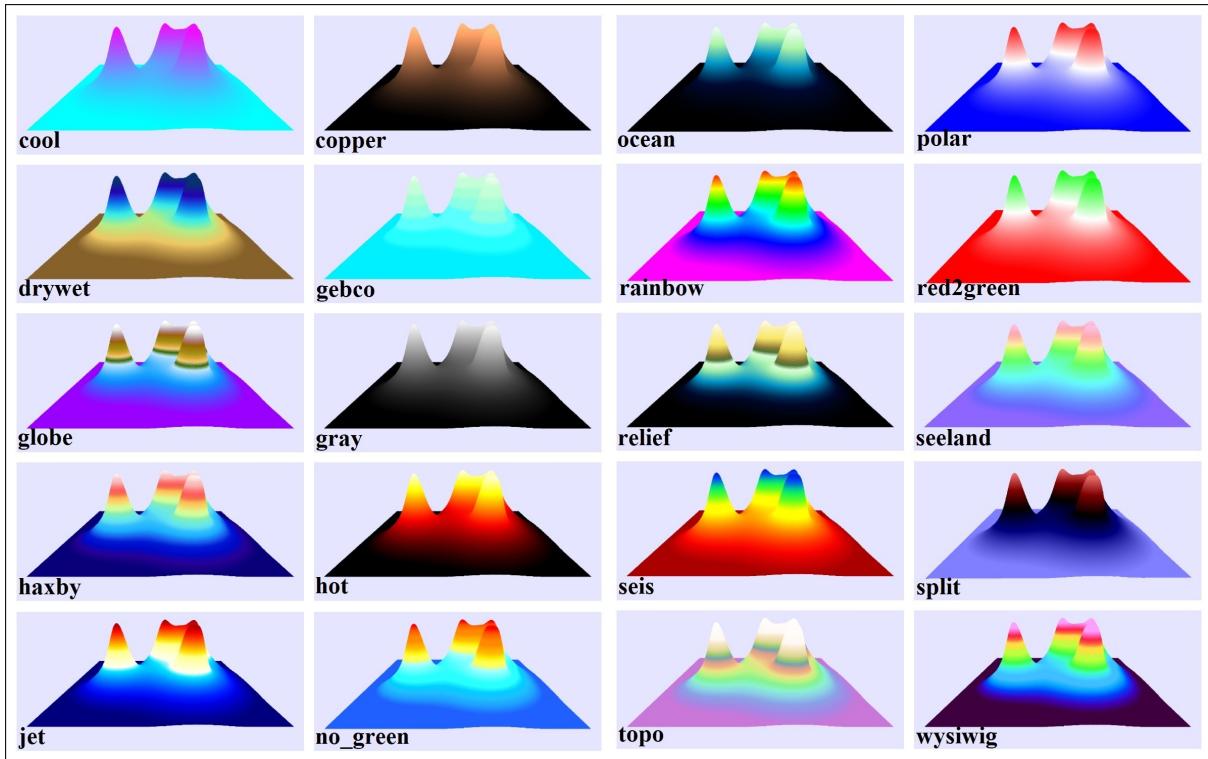


FIGURE 7.6 – Différentes palettes à disposition.

Cette méthode permet de faire abstraction du nombre de couleur défini par la palette et de laisser WebGL calculer l'interpolation entre les couleurs. Le premier avantage est primordial, car WebGL ne permet pas de transférer des tableaux de longueur fixée lors de l'exécution au GPU [53].

Chapitre 8

Tests et résultats

8.1 Comparaison des modèles de rendu avec des données de l'EO

Les différents modèles de rendus ont été développés pour afficher les données issues de l'EO. Dans cette section, nous allons combiner le chargement des données présenté au chapitre 2 avec les modèles de rendu expliqué à la section 7.2 et 7.4. Les données satellites à disposition sont des données brutes, elles sont donc bruitées. Par conséquent, les rendus ne sont pas très lisibles. Il faut en plus de cela considérer le problème concernant l'affichage des courbes de niveau qui est inhérent au rendu des courbes sur la surface à trois dimensions. La figure 8.1 montre le rendu de deux *datacube* venant de l'EO en utilisant le modèle de rendu des courbes sur la surface. Pour simplifier les rendus, les courbes de niveaux d'un seul niveau ont été dessinées. Les données du rendu (a) et (b) sont des mesures de la végétation en Suisse. Le premier utilise l'indice appelé Fraction of Absorbed Photosynthetically Active Radiation (FAPAR, Fraction de rayonnement solaire absorbée par les plantes) alors que le second : Leaf Area Index (LAI, Indice de surface foliaire). De manière générale, il y a du bruit. En plus de cela, il semble que l'image (b) soit encodée sur deux intensités différentes, ce qui provoque ces piques.

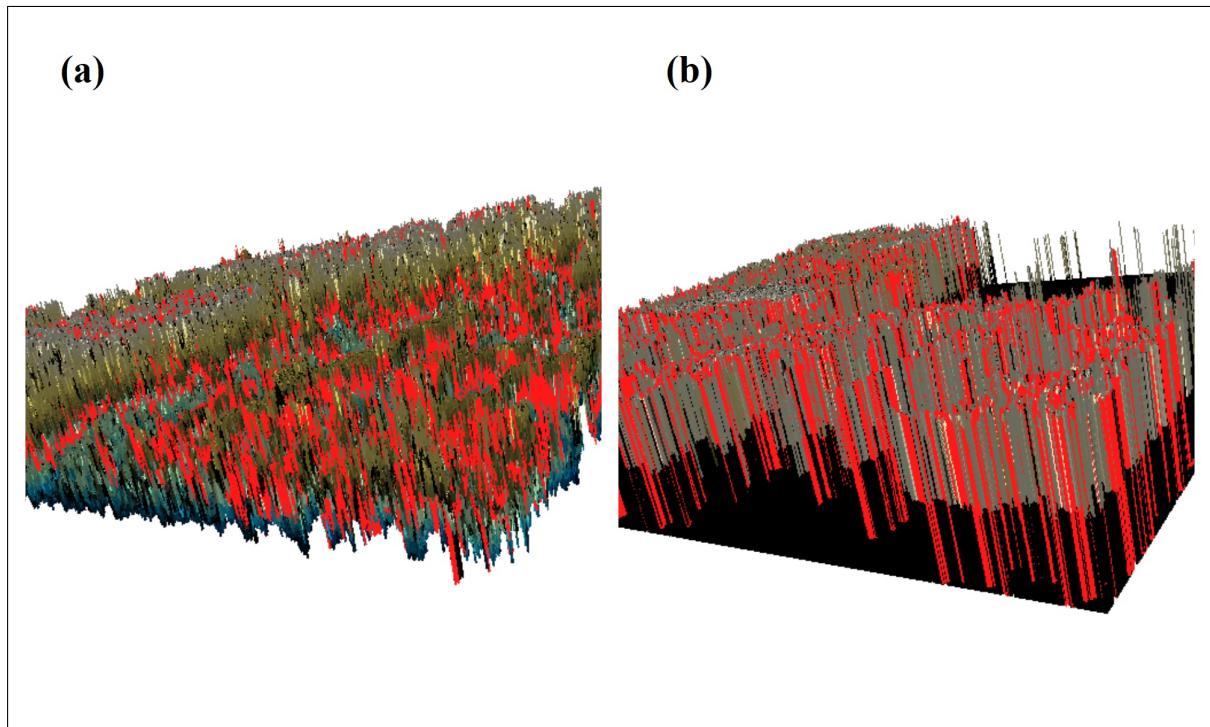


FIGURE 8.1 – Utilisation de données issues de l'EO pour les rendus des courbes sur la surface.

La figure 8.2 illustre l'utilisation des données issues de l'EO avec le modèle de rendu des courbes de niveau le plan. Cette fois-ci, des courbes de plusieurs niveaux différents sont affichées. La courbe de base en rouge correspond à un *iso* de 0.5. Il y a deux courbes voisines en bleu, elles correspondent à des *iso* de 0.1 et 0.9. Les quatre images sont prises à quatre moments différents. On peut voir l'évolution des courbes avec le temps.

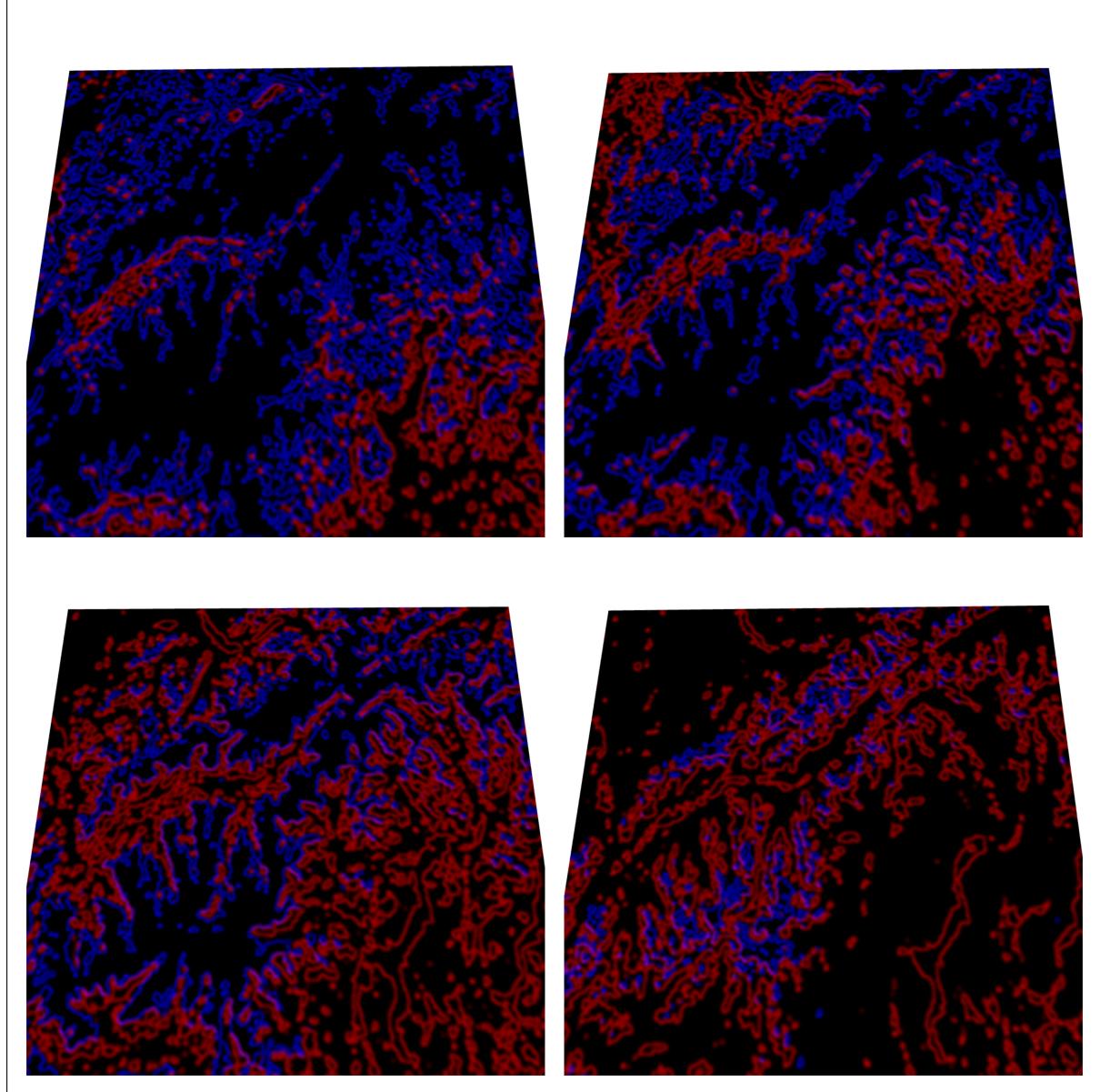


FIGURE 8.2 – Utilisation de données issues de l'EO pour les rendus des courbes sur le plan.

8.2 Comparaison des méthodes de paramétrisations

8.2.1 Calcul de l'erreur

L'erreur $E(\mathcal{C}, \tilde{\mathcal{C}})$ entre la courbe de base \mathcal{C} et la courbe reconstruite $\tilde{\mathcal{C}}$ qui essaie de s'en approcher peut être calculée comme la moyenne des distances entre chaque point de la courbe reconstruite et la courbe de base. Dans le domaine continu :

$$E(\mathcal{C}, \tilde{\mathcal{C}}) = \frac{1}{L} \int_0^T \min_{u \in [0, T]} d_2(\tilde{\mathcal{C}}(t), \mathcal{C}(u)) dx$$

Où T est la période des équations paramétriques des deux courbes et L la longueur de la courbe reconstruite. Dans le domaine discret, l'intégral se transforme en somme :

$$E(\mathcal{C}, \tilde{\mathcal{C}}) = \frac{1}{N} \sum_{\mathbf{p} \in \tilde{\mathcal{C}}} \min_{\mathbf{q} \in \mathcal{C}} d_2(\mathbf{p}, \mathbf{q})$$

Où N est le nombre de points composant la courbe. L'algorithme implémenté pour calculer cette erreur est de complexité $O(n^2)$ car il implique une double boucle. Il est implémenté côté JS, ce qu'il le rend particulièrement lent. Voici l'algorithme en question :

Algorithme 7 : Calcul de l'erreur entre deux courbes

```

Data :  $\mathcal{C}$  Liste de point représentant la courbe de base
 $\tilde{\mathcal{C}}$  Liste de point représentant la courbe reconstruite
Résultat : L'erreur  $E(\mathcal{C}, \tilde{\mathcal{C}})$ 
E  $\leftarrow 0$ 
for  $p \in \tilde{\mathcal{C}}$  do
    min  $\leftarrow \infty$ 
    for  $q \in \mathcal{C}$  do
        tmp  $\leftarrow d_2(p, q)$ 
        if tmp  $\leq$  min then
            | min  $\leftarrow$  tmp
        end
    end
    E  $\leftarrow E +$  min
end
E( $\mathcal{C}, \tilde{\mathcal{C}}$ )  $\leftarrow E / (\text{longueur de } \tilde{\mathcal{C}})$ 
```

Une amélioration possible serait d'implémenter cet algorithme côté GPU. Ainsi, pour chaque point de la courbe reconstruite, un shader calculerait la distance par rapport à la courbe de base et affichera cette valeur sur un des canaux de couleur. Il ne resterait ensuite plus qu'à sommer tous les pixels de l'image. Une problématique subsiste, il faudrait passer la liste des points de la courbe de base à tous les shaders puis itérer sur celle-ci. Malheureusement, WebGL ne permet pas de faire cela, il faudrait utiliser WebGL 2. Cependant, une solution a déjà été à moitié résolue dans la section 7.5. Ce chapitre propose un moyen de passer un tableau par le biais d'une texture. Une telle implémentation reviendrait à exécuter un algorithme de complexité $O(n)$ dans chaque shader.

8.2.2 Méthodologie

Le but ici est de comparer l'erreur due aux différentes méthodes de paramétrisation développées dans ce projet. Pour parvenir à un tel résultat, il faut un grand nombre de courbes d'exemple. Plusieurs images proches d'un cas d'école sont utilisées. Sur chacune de ces images, toutes les courbes de niveau pour tout les *iso* possibles sont calculées. L'intensité étant codé sur huit bits, les *iso* possibles sont les niveaux de zéro à deux cent cinquante-cinq (non normalisé). Ensuite, pour chaque *iso*, les différentes courbes seront récupérées avec l'algorithme présenté à la section 3.1. Les courbes seront ensuite transformées en liste de points en utilisant l'algorithme présenté à la section 4.2. Les cas que l'algorithme ne prend pas en compte seront ignorés. Pour ne pas considérer les hameaux de pixels, il a été décidé que seules les courbes de longueur strictement supérieure à quatre seront utilisées. Pour chacune des courbes d'un *iso* donné, un échantillonnage de celle-ci sera effectué. Les échantillonnages possibles sont l'échantillonnage régulier et celui par approximation polygonale. Pour comparer équitablement ces méthodes sur des courbes de longueurs différentes, on fixera une constante qui définira le nombre de points qu'il faut échantillonner. Il est trivial de fixer le nombre d'échantillon à récupérer lors d'un échantillonnage régulier. Pour faire cela dans l'échantillonnage par approximation polygonale, il faut définir un facteur d'erreur d tel que

le nombre d'échantillons soit celui voulu. Plus d est petit, plus il y aura d'échantillons. De ce fait, une recherche dichotomique sur d permet d'obtenir un nombre d'échantillons donné.¹

De par cette série de points que représente l'échantillonnage d'une courbe, on va calculer les équations paramétriques de la courbe. Les méthodes permettant de passer d'une liste de points à une équation paramétrique sont les deux méthodes présentées à la section 4.1. Les courbes seront alors reconstruites en utilisant l'algorithme permettant de les reconstruire aux pixels prêts.² Pour gagner en performance, la reconstruction ne se fait pas dans un canevas : la liste de points résultant de l'algorithme de reconstruction est directement utilisée. Ceci pose un problème : la liste de points ne représente pas correctement la courbe. En effet, les points sont reconstruits en fonction de l'équation paramétrique en utilisant un pas constant ; or, plus la dérivée de l'équation en un point est faible, plus la concentration de points reconstruits sera grande. Pour éviter ce problème, il faudrait reconstruire la courbe sur une texture puis utiliser l'algorithme permettant de récupérer cette courbe comme liste de points. Pour des raisons de performance, cette solution n'a pas pu être implémentée. Pour terminer, l'algorithme de calcul d'erreur expliqué ci-dessus est utilisé.

8.2.3 Résultats

Le tableau suivant résume les erreurs obtenues en appliquant la méthodologie expliquée plus haut. Les différentes images utilisées sont représentées dans la figure 8.3. Les lignes du tableau représentent les différentes méthodes de paramétrisation possibles. Il y a deux méthodes d'échantillonnage et deux méthodes de paramétrisation, ce qui nous laisse avec quatre possibilités. Les colonnes représentent le nombre de points à échantillonner par courbe.

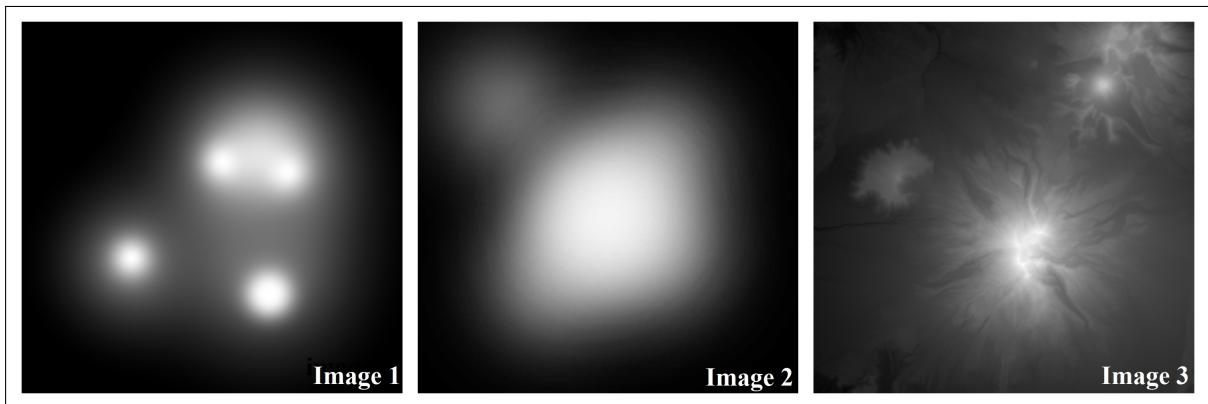


FIGURE 8.3 – Images utilisées pour calculer les statistiques sur les erreurs obtenues après reconstruction.

Méthode	Nombre d'échantillons		
	10	50	80
DFT/Régulier	1.30	0.43	0.40
DFT/Polygonale	2.63	1.20	0.84
Catmull-Rom/Régulier	1.34	0.52	0.45
Catmull-Rom/Polygonale	1.42	0.73	0.56

Comme prévu, l'augmentation du nombre d'échantillons diminue l'erreur obtenue. L'échantillonnage par approximation polygonale semble réduire la précision. Cet effet est plus amplifié en utilisant la DFT. D'une manière générale, la combinaison DFT avec échantillonnage régulier offre les meilleurs résultats.

1. Cependant, il est possible qu'il n'existe pas de d tel que le nombre de sommets du polygone soit celui voulu. Dans tel cas, d sera fixé pour que le nombre de sommets soit le plus proche possible tout en restant plus petit que celui demandé.

2. Cette méthode n'étant pas implémenté pour les splines de Catmull-Rom, une valeur arbitrairement petite a été choisie comme pas de reconstruction.

Des tableaux d'erreur du même genre spécifique à chaque image sont présentés dans l'annexe C. Il n'y a pas de grande différence entre le tableau représentant l'intégralité des images présenté ici et ceux représentant chaque image. Cependant, on peut noter que dans l'image trois, la combinaison splines de Catmull-Rom et échantillonnage polygonal semble être la meilleure.

Chapitre 9

Discussion

9.1 Conclusion

Le cahier des charges a été rempli bien que les données satellites fournies, ne permettent pas d'utiliser pleinement les capacités des rendus développés. L'utilisation d'une carte de hauteur comme entrée au rendu trois dimensions montrées dans la figure 7.5 à la section 7.4, démontre qu'il est possible d'avoir des rendus de bonne qualité tout en utilisant des données satellites. En plus des objectifs principaux, un objectif secondaire a été réalisé. Il s'agit de l'utilisation de palette de couleurs. Ce travail de Bachelor fait partie d'un projet qui va continuer à être développé, c'est pourquoi il est intéressant de proposer des améliorations bien que le temps n'ait pas suffi à les implémenter. Comme l'écoute des clients était au cœur de la définition des objectifs, la gestion de projet à dû être flexible et itérative. Les objectifs ont été redéfinis plusieurs fois, et nous avons pu proposer des idées à la volée. C'est principalement par suite de cette liberté que les projets de Bachelor de Monsieur Mathias Marty et Monsieur Antoine Lestrade ont divergé.

D'une manière générale, nous avions de l'avance sur le planning. Ce confort nous a permis d'explorer le domaine de l'extraction des courbes de niveau et la reconstruction de celles-ci par une équation paramétrique. Cet objectif prend une grande place dans le projet. L'annexe D compare ce qui a été planifié et le temps effectif passé dessus. Il est arrangé par semaine et les deux séances les plus importantes y sont notées comme des milestones.

9.2 Perspsective

La suite du projet consiste à offrir ces modèles de rendus par le biais d'une boîte à outils développée en Python. De cette façon, les scientifiques pourraient facilement incorporer des rendus dans un *notebook* Jupyter. Le projet utilisant des technologie Web, il serait envisageable d'imaginer un site Web qui offrirait la possibilité d'accéder au modèle de rendu appliquée à des données satellites. De la manière d'un Google map, un tel service pourrait être utilisé par des utilisateurs qui ne sont pas forcément des scientifiques afin qu'ils aient un accès rapide et facile aux données satellites. Comme le code des rendus et du chargement des données satellites est en JS, le portage de l'application sur un site Web ne demandera pas de travail supplémentaire vis-à-vis du code déjà développé.

Concernant la partie extraction des courbes, elle aurait deux potentielles utilités :

1. Utiliser l'équation paramétrique de la courbe extraite pour sa reconstruction du côté du GPU et développer un moyen de l'afficher proprement sur le rendu de la surface à trois dimensions ;
2. Mettre en place un serveur qui précalculerait les courbes de niveau d'une couverture satellite donnée puis les sauvegarderait tout en gardant un contrôle sur le facteur qualité/quantité des données. Ensuite, un client pourrait faire une requête sur ce serveur afin d'obtenir les caractéristiques des courbes et d'ainsi, de pouvoir les reconstruire lui-même.

Un autre rendu proposé par Monsieur Stéphane Gobron est le rendu *wireframe* (fil de fer), il nécessite l'utilisation du rendu rayons X développé par Monsieur Antoine Lestrade, l'extraction des courbes de niveau et leur reconstruction. En récupérant les courbes de niveau d'un seuil donné de plusieurs rendu rayons X d'un *datacube* à trois dimensions, il serait possible d'obtenir l'équation paramétrique du *wireframe*. On aurait ainsi, un moyen d'extraire et de reconstruire la topologie des surfaces de niveau.

Table des figures

1.1	Attribution des modèles de rendu et brève description de ceux-ci.	8
1.2	Quatre prises de vues différentes venant de satellite de l'ESA et d'Airbus.	10
1.3	Différentes méthodes de rendus.	11
1.4	Différentes orbites dont la géostationnaire.	11
1.5	Graphe des résolutions angulaires par rapport au diamètre du miroir pour différentes longueurs d'ondes.	12
1.6	Exemple de rendu de logiciel 3DbvNCA. A gauche le rendu X Ray et à droite le rendu de surface avec la colorisation en fonction des normales.	13
1.7	Rendu d'un CT <i>scan</i> d'un crâne en utilisant l'algorithme de <i>raycasting</i> . La dimension du datacube est 512x512x750 et chaque voxel est encodé sur 16 bits [34].	14
1.8	Capture d'écran du site AGRO-DÉ en utilisant différentes transparences.	14
1.9	Capture d'écran d'une application proposée par Nautikaris.	15
1.10	Modélisation 3 dimensions des fonds marins d'un des atolls de Tuvalu dans un programme développé par EOMAP.	15
1.11	Différents navigateurs supportant WebGL.	16
2.1	Schéma représentant comment les données satellites sont enregistrées puis accédées.	17
3.1	Comparaison de l'algorithme naïf avec celui développé dans le cadre de ce projet.	19
3.2	Applications de filtres sur une image de base. La valeur <i>iso</i> est fixée à 0.5 et ϵ à 0.1. L'image (a) est l'image de base ; la (b) sont les bords voulus ; la (c) le résultat en utilisant l'algorithme naïf ; la (d) le résultat après la binarisation ; la (e) résultat en utilisant le filtre de Sobel ; la (f) le résultat en utilisant le filtre de Prewitt ; la (g) le résultat en utilisant l'algorithme développé pour ce projet et la (h), le résultat si l'on change le noyau G_x .	20
3.3	Application de l'algorithme de construction des courbes de niveau à différentes images.	23
4.1	Comparaison des deux méthodes d'échantillonnage.	26
4.2	Différentes splines composants une courbe.	28
4.3	Cas non fonctionnel de la transformation d'une courbe en liste de points.	29
4.4	Comparaison des méthodes de paramétrisation des courbes.	30
5.1	Les huit octants du plan.	31
5.2	Exemple de segment flou et de sa droite discrète englobante.	32
5.3	Comparaison de l'approximation polygonale avec différentes valeurs de d .	35
6.1	Comparaison des méthodes de reconstruction des courbes.	38
7.1	Différentes prises de vues du rendu des courbes avec l'image de base.	41
7.2	Comparaison du rendu de la surface en utilisant des facteurs de hauteurs différentes.	42
7.3	Comparaison des deux modèles de calcul de normale.	44
7.4	Problème dû à la méthode d'intégration des courbes de niveau sur la surface à trois dimensions.	45
7.5	Rendu des courbes sur les surfaces à trois dimensions appliquées à deux images différentes.	45
7.6	Différentes palettes à disposition.	46
8.1	Utilisation de données issues de l'EO pour les rendus des courbes sur la surface.	47

8.2 Utilisation de données issues de l'EO pour les rendus des courbes sur le plan.	48
8.3 Images utilisées pour calculer les statistiques sur les erreurs obtenues après reconstruction.	50

Bibliographie

- [1] H. Hassani, X. Huang, and E. Silva, “Big Data and Climate Change,” *Big Data and Cognitive Computing*, vol. 3(1), Feb. 2019.
- [2] “Présentation – Google Earth,” May 2021. [Online]. Available : <https://www.google.com/intl/fr/earth/>
- [3] G. Denis, A. Claverie, X. Pasco, J.-P. Darnis, B. de Maupeou, M. Lafaye, and E. Morel, “Towards disruptions in Earth observation ? New Earth Observation systems and markets evolution : Possible scenarios and impacts,” *Acta Astronautica*, vol. 137, pp. 415–433, Aug. 2017. [Online]. Available : <https://www.sciencedirect.com/science/article/pii/S0094576516313492>
- [4] “Copernicus (programme),” May 2021, page Version ID : 183235960. [Online]. Available : [https://fr.wikipedia.org/w/index.php?title=Copernicus_\(programme\)&oldid=183235960](https://fr.wikipedia.org/w/index.php?title=Copernicus_(programme)&oldid=183235960)
- [5] “Sentinel (satellite),” Apr. 2021, page Version ID : 181803844. [Online]. Available : [https://fr.wikipedia.org/w/index.php?title=Sentinel_\(satellite\)&oldid=181803844](https://fr.wikipedia.org/w/index.php?title=Sentinel_(satellite)&oldid=181803844)
- [6] P. Baumann, A. P. Rossi, B. Bell, O. Clements, B. Evans, H. Hoenig, P. Hogan, G. Kakaletris, P. Koltsida, S. Mantovani, R. Marco Figuera, V. Merticariu, D. Misev, H. B. Pham, S. Siemen, and J. Wagemann, “Fostering Cross-Disciplinary Earth Science Through Datacube Analytics,” in *Earth Observation Open Science and Innovation*, ser. ISSI Scientific Report Series, P.-P. Mathieu and C. Aubrecht, Eds. Cham : Springer International Publishing, 2018, pp. 91–119. [Online]. Available : https://doi.org/10.1007/978-3-319-65633-5_5
- [7] *the rasdaman raster array database – rasdaman*, May 2021. [Online]. Available : <http://www.rasdaman.org/>
- [8] *Web WorldWind/NASA WorldWind*, May 2021. [Online]. Available : <https://worldwind.arc.nasa.gov/web/>
- [9] “ESA - Aeolus,” May 2021. [Online]. Available : https://www.esa.int/Applications/Observing_the_Earth/Aeolus
- [10] “Monitoring and understanding the Earth,” May 2021. [Online]. Available : <https://www.airbus.com/company/history/space-history/monitoring-the-earth.html>
- [11] “Biomass (satellite),” May 2021, page Version ID : 182980719. [Online]. Available : [https://fr.wikipedia.org/w/index.php?title=Biomass_\(satellite\)&oldid=182980719](https://fr.wikipedia.org/w/index.php?title=Biomass_(satellite)&oldid=182980719)
- [12] “Methane Remote Sensing Lidar Mission,” May 2020, page Version ID : 171482828. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Methane_Remote_Sensing_Lidar_Mission&oldid=171482828
- [13] “MicroCarb,” Apr. 2021, page Version ID : 182045557. [Online]. Available : <https://fr.wikipedia.org/w/index.php?title=MicroCarb&oldid=182045557>
- [14] “EarthCARE,” Nov. 2020, page Version ID : 176332619. [Online]. Available : <https://fr.wikipedia.org/w/index.php?title=EarthCARE&oldid=176332619>
- [15] “Data cube,” Apr. 2021, page Version ID : 1019849415. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Data_cube&oldid=1019849415
- [16] “Nuage de points (géométrie),” Dec. 2018, page Version ID : 155223201. [Online]. Available : [https://fr.wikipedia.org/w/index.php?title=Nuage_de_points_\(g%C3%A9om%C3%A9trie\)&oldid=155223201](https://fr.wikipedia.org/w/index.php?title=Nuage_de_points_(g%C3%A9om%C3%A9trie)&oldid=155223201)

- [17] “Polygon soup,” Aug. 2018, page Version ID : 853571813. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Polygon_soup&oldid=853571813
- [18] “Mesh (objet),” May 2021, page Version ID : 183308968. [Online]. Available : [https://fr.wikipedia.org/w/index.php?title=Mesh_\(objet\)&oldid=183308968](https://fr.wikipedia.org/w/index.php?title=Mesh_(objet)&oldid=183308968)
- [19] “Marching cubes,” Jun. 2020, page Version ID : 172247010. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Marching_cubes&oldid=172247010
- [20] “Loi de Moore,” Apr. 2021, page Version ID : 182007650. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Loi_de_Moore&oldid=182007650
- [21] “Lois de Kepler,” May 2021, page Version ID : 182746716. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Lois_de_Kepler&oldid=182746716
- [22] “Orbite géostationnaire,” Apr. 2021, page Version ID : 182238550. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Orbite_g%C3%A9ostationnaire&oldid=182238550
- [23] “Pouvoir de résolution,” Nov. 2020, page Version ID : 176378282. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Pouvoir_de_r%C3%A9solution&oldid=176378282
- [24] H.-D. Guo, L. Zhang, and L.-W. Zhu, “Earth observation big data for climate change research,” *Advances in Climate Change Research*, vol. 6, no. 2, pp. 108–117, Jun. 2015, number : 2. [Online]. Available : <https://www.sciencedirect.com/science/article/pii/S1674927815000519>
- [25] P. Srivastava, M. Piles, and S. Pearson, “Earth Observation-Based Operational Estimation of Soil Moisture and Evapotranspiration for Agricultural Crops in Support of Sustainable Water Management,” *Sustainability*, vol. 10, p. 181, Jan. 2018.
- [26] “Home | Sustainable Development,” May 2021. [Online]. Available : <https://sdgs.un.org/>
- [27] “Compendium of EO contributions to the SDGs just released,” Jan. 2021, section : Sustainable Development. [Online]. Available : <https://eo4society.esa.int/2021/01/15/compendium-of-eo-contributions-to-the-sdgs-just-released/>
- [28] “EOS Crop Monitoring - A New Farm Software For Agricultural Sector,” Dec. 2020. [Online]. Available : <https://eos.com/products/crop-monitoring/>
- [29] “OneSoil | Monitor fields and crops with satellite imagery for free,” May 2021. [Online]. Available : <https://onesoil.ai/en/applications>
- [30] “Precision Agriculture Imaging with Planet Satellite Solutions,” May 2021. [Online]. Available : <https://www.planet.com/markets/monitoring-for-precision-agriculture/>
- [31] S. Gobron, A. Çöltekin, H. Bonafos, and D. Thalmann, “GPGPU computation and visualization of three-dimensional cellular automata,” *Vis Comput*, vol. 27, no. 1, pp. 67–81, Jan. 2011, number : 1. [Online]. Available : <http://link.springer.com/10.1007/s00371-010-0515-1>
- [32] “OpenGL - The Industry Standard for High Performance Graphics,” May 2021. [Online]. Available : <https://www.opengl.org/>
- [33] N. Freud, P. Duvauchelle, J. M. Létang, and D. Babot, “Fast and robust ray casting algorithms for virtual X-ray imaging,” *Nuclear Instruments and Methods in Physics Research Section B : Beam Interactions with Materials and Atoms*, vol. 248, no. 1, pp. 175–180, Jul. 2006. [Online]. Available : <https://www.sciencedirect.com/science/article/pii/S0168583X0600245X>
- [34] “Volume ray casting,” Dec. 2020, page Version ID : 995198788. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Volume_ray_casting&oldid=995198788
- [35] “Raycasting,” Apr. 2020, page Version ID : 169082064. [Online]. Available : <https://fr.wikipedia.org/w/index.php?title=Raycasting&oldid=169082064>
- [36] “Homepage.” [Online]. Available : <https://agro-de.info/>
- [37] “Nautikaris is your best address for your needs for Hydrographic and Ocean.” [Online]. Available : <https://www.nautikaris.com/>
- [38] N. S. Hashim, O. M. Yusof, W. Windupranata, and S. A. H. Sulaiman, “Integration of Satellite-Derived Bathymetry and Sounding Data in Providing Continuous and Detailed Bathymetric Information,” *IOP Conf. Ser. : Earth Environ. Sci.*, vol. 618, p. 012018, Dec. 2020. [Online]. Available : <https://iopscience.iop.org/article/10.1088/1755-1315/618/1/012018>

- [39] eomap, “Home.” [Online]. Available : <https://www.eomap.com/>
- [40] “WebGL,” Feb. 2021, page Version ID : 180305753. [Online]. Available : <https://fr.wikipedia.org/w/index.php?title=WebGL&oldid=180305753>
- [41] “Sobel operator,” Sep. 2006, page Version ID : 74907814. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Sobel_operator&oldid=74907814
- [42] “Prewitt operator,” Mar. 2013, page Version ID : 542544197. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Prewitt_operator&oldid=542544197
- [43] “Norme (mathématiques),” Mar. 2021, page Version ID : 181361027. [Online]. Available : [https://fr.wikipedia.org/w/index.php?title=Norme_\(math%C3%A9matiques\)&oldid=181361027](https://fr.wikipedia.org/w/index.php?title=Norme_(math%C3%A9matiques)&oldid=181361027)
- [44] “Série de Fourier — Wikipédia.” [Online]. Available : https://fr.wikipedia.org/wiki/S%C3%A9rie_de_Fourier
- [45] “Théorème de Riemann-Lebesgue,” Oct. 2020, page Version ID : 175714683. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Th%C3%A9or%C3%A8me_de_Riemann-Lebesgue&oldid=175714683
- [46] F. Bornert, “Traitements d’images et analyse de Fourier elliptique : Application à la quantification des dysmorphoses mandibulaires chez la souris Tabby (Modèle murin de la dysplasie ectodermique),” p. 54.
- [47] C. Yuksel, S. Schaefer, and J. Keyser, “Parameterization and applications of Catmull–Rom curves,” *Computer-Aided Design*, vol. 43, no. 7, pp. 747–755, Jul. 2011. [Online]. Available : <https://www.sciencedirect.com/science/article/pii/S0010448510001533>
- [48] I. Debled-Rennesson, R. Jean-Luc, and J. Rouyer-Degli, “Segmentation of Discrete Curves into Fuzzy Segments,” *Electronic Notes in Discrete Mathematics*, vol. 12, pp. 372–383, Mar. 2003. [Online]. Available : <https://linkinghub.elsevier.com/retrieve/pii/S1571065304005001>
- [49] J.-P. Reveillès, “Géométrie discrète, calcul en nombres entiers et algorithmique,” Habilitation à diriger des recherches, Université Louis Pasteur, Dec. 1991. [Online]. Available : <https://hal.archives-ouvertes.fr/tel-01279525>
- [50] S. Gobron and M. Gutierrez, *WebGL par la pratique*, 1st ed. Lausanne : PPUR, 2015.
- [51] “OpenGL ES,” Mar. 2021, page Version ID : 180988152. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=OpenGL_ES&oldid=180988152
- [52] “The Generic Mapping Tools Documentation — GMT 6.3.0 documentation.” [Online]. Available : <https://docs.generic-mapping-tools.org/dev/index.html>
- [53] R. Simpson, “The OpenGL ES Shading Language,” p. 119, 2009.
- [54] S. Gobron, C. Marx, J. Ahn, and D. Thalmann, “Real-time textured volume reconstruction using virtual and real video cameras,” p. 4.
- [55] M. Fröhlich, C. Bolinhas, A. Depersinge, A. Widmer, N. Chevrey, P. Hagmann, C. Simon, V. B. C. Kokje, and S. Gobron, “Holographic Visualisation and Interaction of Fused CT, PET and MRI Volumetric Medical Imaging Data Using Dedicated Remote GPGPU Ray Casting,” in *Simulation, Image Processing, and Ultrasound Systems for Assisted Diagnosis and Navigation*, D. Stoyanov, Z. Taylor, S. Aylward, J. M. R. Tavares, Y. Xiao, A. Simpson, A. Martel, L. Maier-Hein, S. Li, H. Rivaz, I. Reinertsen, M. Chabanais, and K. Farahani, Eds. Cham : Springer International Publishing, 2018, vol. 11042, pp. 102–110, series Title : Lecture Notes in Computer Science. [Online]. Available : http://link.springer.com/10.1007/978-3-030-01045-4_12
- [56] “WebGL 2 Volume Rendering using Ray Casting,” Aug. 2019. [Online]. Available : <https://observablehq.com/@ukabuer/webgl-2-volume-rendering-using-ray-casting>
- [57] “Lebara,” Mar. 2021. [Online]. Available : <http://www.lebarba.com/>
- [58] L. Barbagallo, “lebara/WebGLVolumeRendering,” Mar. 2021, programmers : _ :n324 original-date : 2014-10-12T20:40:46Z. [Online]. Available : <https://github.com/lebara/WebGLVolumeRendering>
- [59] Franz, “Scarysize/webgl-raycasting,” Jan. 2021, programmers : _ :n327 original-date : 2017-04-20T06:07:36Z. [Online]. Available : <https://github.com/Scarysize/webgl-raycasting>
- [60] S. Macke, “s-macke/VoxelSpace,” Mar. 2021, programmers : _ :n335 original-date : 2017-09-23T18:26:07Z. [Online]. Available : <https://github.com/s-macke/VoxelSpace>

- [61] N. Jiarathanakul, “nopjia/WebGL-Volumetric,” Feb. 2021, programmers : __ :n346 original-date : 2012-08-15T22 :46 :51Z. [Online]. Available : <https://github.com/nopjia/WebGL-Volumetric>
- [62] K. Hayward, “Graphics Runner : Volume Rendering 101,” Jan. 2009. [Online]. Available : <http://graphicsrunner.blogspot.com/2009/01/volume-rendering-101.html>
- [63] “Gutenko | Web Volume Rendering,” Mar. 2021. [Online]. Available : <https://www3.cs.stonybrook.edu/~igutenko/webglvolrender.html>
- [64] “inviwo/inviwo,” Mar. 2021, original-date : 2017-10-25T11 :47 :56Z. [Online]. Available : <https://github.com/inviwo/inviwo>
- [65] A. K. Whitcraft, I. Becker-Reshef, B. D. Killough, and C. O. Justice, “Meeting Earth Observation Requirements for Global Agricultural Monitoring : An Evaluation of the Revisit Capabilities of Current and Planned Moderate Resolution Optical Earth Observing Missions,” *Remote Sensing*, vol. 7, no. 2, pp. 1482–1503, Feb. 2015, number : 2 Publisher : Multidisciplinary Digital Publishing Institute. [Online]. Available : <https://www.mdpi.com/2072-4292/7/2/1482>
- [66] G. Manogaran and D. Lopez, “Spatial cumulative sum algorithm with big data analytics for climate change detection,” *Computers & Electrical Engineering*, vol. 65, pp. 207–221, Jan. 2018. [Online]. Available : <https://linkinghub.elsevier.com/retrieve/pii/S004579061730811X>
- [67] H. Ray, H. Pfister, D. Silver, and T. Cook, “Ray casting architectures for volume visualization,” *IEEE Trans. Visual. Comput. Graphics*, vol. 5, no. 3, pp. 210–223, Sep. 1999, number : 3. [Online]. Available : <http://ieeexplore.ieee.org/document/795213/>
- [68] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, and O. Ruiz, “Interactive visualization of volumetric data with WebGL in real-time,” in *Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11*. Paris, France : ACM Press, 2011, p. 137. [Online]. Available : <http://dl.acm.org/citation.cfm?doid=2010425.2010449>
- [69] “Mesh (objet),” May 2021, page Version ID : 183308968. [Online]. Available : [https://fr.wikipedia.org/w/index.php?title=Mesh_\(objet\)&oldid=183308968](https://fr.wikipedia.org/w/index.php?title=Mesh_(objet)&oldid=183308968)

Annexe A

Démonstration du majorant de la dérivée de la série partielle de Fourier

Soit la série de Fourier partiel suivante :

$$S_N(f)(t) = \sum_{k=-N/2}^{N/2} c_k(f) e^{2i\pi \frac{k}{T} t}$$

Étant un polynôme trigonométrique, on peut facilement calculer sa dérivée :

$$S_N(f)'(t) = \frac{2i\pi}{T} \sum_{k=-N/2}^{N/2} k c_k(f) e^{2i\pi \frac{k}{T} t}$$

Il faut maintenant trouver un majorant m de la norme de la dérivée la courbe décrite à l'équation 4.1, c'est-à-dire :

$$m \geq \| \begin{pmatrix} \Re(S_N(z)'(t)) \\ \Im(S_N(z)'(t)) \end{pmatrix} \|_2 \Rightarrow m \geq |S_N(z)'(t)|$$

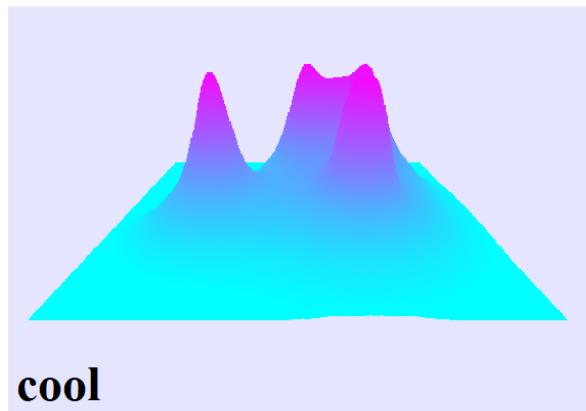
On définit m en utilisant l'inégalité triangulaire.

$$\begin{aligned} |S_N(z)'(t)| &= \left| \frac{2i\pi}{T} \sum_{k=-N/2}^{N/2} k c_k(f) e^{2i\pi \frac{k}{T} t} \right| \\ &\leq \frac{2\pi}{T} \sum_{k=-N/2}^{N/2} |k| \|c_k(f) e^{2i\pi \frac{k}{T} t}\| \\ &= \frac{2\pi}{T} \sum_{k=-N/2}^{N/2} |k| \|c_k(f)\| \\ &= m \end{aligned}$$

Annexe B

Palette de couleur

```
{  
    "cool": [  
        [ 0, 255, 255,  
         13, 242, 255,  
         ...  
         102, 153, 255,  
         ...  
         242, 13, 255  
        ],  
    ],  
    "copper": [  
        [ ...  
        ],  
    ],  
    "drywet": [  
        [ ...  
        ],  
        ...  
    ]  
}
```



Annexe C

Erreurs sur les images spécifiques

Erreurs provenant de l'image 1 :

Méthode	Nombre d'échantillons	10	50	80
DFT/Régulier		1.57	0.40	0.38
DFT/Polygonale		2.64	1.22	1.00
Catmull-Rom/Régulier		1.98	0.43	0.39
Catmull-Rom/Polygonale		2.29	0.89	0.72

Erreurs provenant de l'image 2 :

Méthode	Nombre d'échantillons	10	50	80
DFT/Régulier		1.23	0.38	0.37
DFT/Polygonale		4.06	1.69	1.03
Catmull-Rom/Régulier		1.26	0.47	0.43
Catmull-Rom/Polygonale		1.72	0.85	0.59

Erreurs provenant de l'image 3 :

Méthode	Nombre d'échantillons	10	50	80
DFT/Régulier		1.24	0.50	0.43
DFT/Polygonale		1.30	0.72	0.60
Catmull-Rom/Régulier		1.12	0.63	0.50
Catmull-Rom/Polygonale		0.72	0.53	0.45

Annexe D

Planning