

THÈSE

pour l'obtention du Grade de
DOCTEUR DE L'UNIVERSITÉ DE POITIERS
(Faculté des Sciences Fondamentales et Appliquées)
(Diplôme National - Arrêté du 7 Août 2006)

École Doctorale : **Sciences Pour l'Ingénieur et Aéronautique**
Secteur de Recherche : **Informatique et Applications**

Présentée par :
Pierre-François LÉON

Animation et contrôle de structures topologiques : application à la simulation d'évolution de couches géologiques 2D

Soutenue le 19 juin 2009 devant la Commission d'Examen composée de :

P. Lienhardt	Professeur au XLIM-SIC, Université de Poitiers	Président
D. Bechmann	Professeure au LSIIT, Université de Strasbourg	Rapporteure
L. Lucas	Professeur au CReSTIC, Université de Reims	Rapporteur
J.F. Rainaud	Ingénieur de recherche à l'Institut Français du Pétrole	Examinateur
O. Terraz	Maître de Conférences à XLIM, Université de Limoges	Examinateur
P. Meseure	Professeur à XLIM-SIC, Université de Poitiers	Directeur
X. Skapin	Maître de Conférences à XLIM-SIC , Université de Poitiers	Encadrant

TABLE DES MATIÈRES

1	Introduction	1
2	État de l'art	5
2.1	Modélisation à base topologique	5
2.1.1	Définitions	5
2.1.2	Modèles topologiques	10
2.1.3	Les n-G-Cartes	13
2.2	Modèles d'animation	16
2.2.1	Modèles descriptifs	16
2.2.2	Modèles générateurs	17
2.2.3	Choix de l'espace de représentation spatio-temporel	19
2.3	L'animation à base topologique	22
2.3.1	les L-Systèmes	22
2.3.2	Les systèmes vertex-vertex	23
2.3.3	MGS	24
2.3.4	Discussion	26
2.4	Conclusion	26
3	Modèle topologique spatio-temporel	29
3.1	Structure topologique temporelle	29
3.1.1	Approche $(n + 1)D$	30
3.1.2	Modèle par images clefs	33
3.1.3	Discussion	34
3.2	Étude du cas particulier 2D	35
3.2.1	Opérations topologiques 2D	35
3.2.2	Plongement géométrique	42

3.2.3	Contrôle de l'animation	43
3.3	Bilan	47
4	Un système d'animation topologique	49
4.1	Le modèle d'animation	49
4.1.1	La composante événementielle	50
4.1.2	La composante sémantique	52
4.1.3	Interaction entre les composantes	53
4.2	Étude du cas particulier 2D	53
4.2.1	Types et résolution d'événements de collisions	53
4.2.2	Calcul de trajectoires	59
4.2.3	Implantation d'un prototype 2D	60
4.3	Conclusion	61
5	Application à la géologie	63
5.1	Problématique	63
5.2	État de l'art en modélisation et simulation géologique	64
5.3	Modélisation 2D des phénomènes géologiques	69
5.3.1	Modèle de sédimentation	69
5.3.2	Modèle d'érosion	72
5.3.3	Modèle de création de failles	76
5.3.4	Modèle de glissement	79
5.4	Résultats	81
5.4.1	Modélisation topologique	81
5.4.2	Modélisation à base de scénario générateur	87
5.5	Conclusion	93
6	Conclusion et Perspectives	95
6.1	Bilan	95
6.2	Perspectives	96
6.2.1	Instanciation du modèle en 3D	97
6.2.2	Améliorations pour une utilisation plus facile du système	98
6.2.3	Amélioration de la robustesse et des performances	99
A	Sources	101
A.1	Fonctions utilitaires pour le script bas-niveau	101

A.2 Reconnaissance de motifs	106
Bibliographie	111

INTRODUCTION

Les sciences expérimentales trouvent souvent dans la nature des objets structurés de façon bien spécifique. Pour certains des structures observées, les scientifiques ont pu montrer qu'elles étaient particulièrement adaptées à une fonction donnée et même optimales pour un critère déterminé. Prenons l'exemple macroscopique du pavage d'un plan par un polygone régulier. Quel est le polygone dont le périmètre est minimal pour pavé ce plan ? La réponse a naturellement été trouvée par les abeilles : l'hexagone (cf. figure 1.1). Il est en effet prouvé par Thomas C. Hales [35] que toute partition du plan en régions de surfaces égales a un périmètre au moins égal à celui du pavage "en nid d'abeille" par des hexagones réguliers. Que la structure la plus adaptée soit celle que la nature utilise est tout à fait attendu : d'après Darwin, le processus de sélection naturel permet de conserver les espèces les mieux adaptées au milieu environnant, et ceci est également valable pour les structures. Cependant, il n'est pas toujours évident, pour les scientifiques, de trouver en quoi telle structure observée est « optimale », ni parfois à quoi la structure sert réellement.

L'analyse de ces structures pose en outre des problèmes d'échelle. Il est courant qu'une structure définie à l'échelle nanoscopique se trouve à l'origine de phénomènes macroscopiques essentiels. Prenons l'exemple de structures composées d'un même et seul élément, le carbone, qui aboutissent à deux types de minéraux aux propriétés très différentes. Le premier est le graphite, une matière noirâtre et très friable. Le second est le diamant. Le diamant est le plus dur des matériaux naturels et il est de plus translucide. Ces différences de propriétés sont seulement dues à l'agencement des atomes de carbones qui les composent. Pour le graphite, les atomes de carbone ont une structuration en feuillets hexagonaux ; pour le diamant, ils ont une structuration cubique (cf. figure 1.2).

On constate ainsi que la structure des objets naturels est étroitement liée à leurs propriétés physiques et fonctionnelles. Étudier de telles structures, en comprendre le fonctionnement et l'évolution permet de comprendre leur rôle. L'objectif de ce mémoire est de fournir un système de représentation de ces structures, ainsi que leurs évolutions grâce à des outils apparentés à la simulation. Dans cette optique, nous présentons une nouvelle méthode d'animation de structures topologiques. Son objectif est de représenter l'évolution d'une structure à travers le temps, à partir d'une description pouvant être donnée par l'utilisateur de manière intuitive et en garantissant la cohérence topologique du système au cours du temps. Comme nous visons l'évolution



FIG. 1.1 – Rayon de cire d’abeilles domestiques portant des oeufs et des larves. Les parois des cellules ont été enlevées. Les larves (des faux-bourdons) sont âgées de 3 à 4 jours (Photo de Waugsberg extraite de wikipedia).

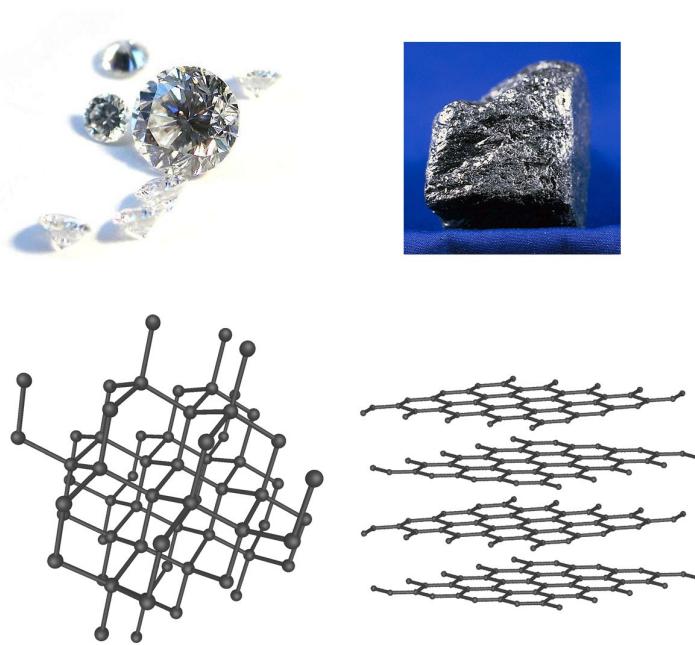


FIG. 1.2 – Photos d’un diamant et d’un morceau de graphite, et leurs structures (Image extraite de wikipedia).

d’un seul objet structuré, il est pratique de cantonner notre système à l’évolution de partitions de l’espace.

De par son étendue, le sujet peut être abordé sous plusieurs angles : celui de la modélisation géométrique à base topologique, celui de l’animation ou celui de la simulation.

La modélisation géométrique à base topologique cherche à représenter la structure indépendamment de la forme (géométrie) de l'objet. Cette structure autorise le calcul de propriétés topologiques telles que la caractéristique d'Euler, le nombre de « trous » et/ou de cavités, de composantes connexes, etc., ce qui permet par ailleurs de classifier les objets. La forme de l'objet ainsi que ses autres propriétés physiques, sémantiques, etc., sont alors associées à cette structure par un mécanisme nommé plongement. Une telle représentation permet de structurer les données et d'en faciliter l'accès, comme de favoriser le calcul de nouvelles données (genre de l'objet, nombre de trous, nombre de cavités, etc.). Cette structure peut alors chercher à englober l'intégralité de l'espace considéré. On parle alors de *partition de l'espace*. Cette structure intègre non seulement les objets ou parties d'objets étudiés, mais également les volumes les liant. Le but de ce travail n'est cependant pas de représenter et d'étudier la structure statique des objets (ce sujet a déjà largement été traité (cf. section 2.1)), mais plutôt de modéliser l'évolution de celle-ci au cours du temps. De ce fait, notre approche doit se baser non seulement sur la modélisation géométrique, mais également sur les techniques d'animation/simulation.

L'animation vise à fournir des modèles spatio-temporels décrivant des évolutions au cours du temps et à définir les informations nécessaires à l'évolution d'un environnement virtuel, en utilisant des mécanismes de description et de contrôle de mouvement. La simulation, quant à elle, a pour but de fournir des solutions de génération des mouvements en se basant sur des lois d'évolution. Nous prenons ici le terme « simulation » dans un sens large, qui inclut la simulation physique, c'est-à-dire l'utilisation d'équations issues du réel pour décrire une évolution, mais également des simulations basées sur des lois de comportement générales (par exemple un système expert) ou des règles de métamorphoses d'objets (telles que les L-Systèmes pour la simulation de croissance de plantes).

Pour étudier l'évolution d'une partition de l'espace, il faut ainsi s'atteler à deux problèmes : la représentation de la partition et son évolution. Dans un premier temps, nous avons traité ce problème sous un angle de modélisation géométrique, en proposant des opérations de transformation fournissant les étapes de l'animation d'une subdivision. Nous nous sommes orientés ensuite vers une méthode plus automatique, que nous qualifions de *génératrice*, propre à simplifier la description des animations. En effet, décrire le mouvement de chaque élément de la structure est fastidieux de par le volume de données à fournir ainsi qu'en raison des erreurs potentiellement introduites. Les techniques d'animation ont ainsi classiquement recours à des modèles génératrices. Dans notre cas, donner un état initial du système ainsi que des opérations haut-niveau (c'est-à-dire proches du langage de l'animateur) permet de réduire le travail de description de la scène. La modélisation de l'animation de structures topologiques que nous proposons s'articule autour d'un langage qui manipule la topologie et la géométrie de la scène. Pour automatiser ce système, nous avons choisi – à partir d'une scène de départ – de fournir des comportements à adopter en fonction d'événements apparaissant durant l'animation. La détection et le traitement de ces événements, en particulier les modifications topologiques, nous permettent de garantir la cohérence du système. En outre, nous souhaitons fournir comme résultat non seulement une animation correcte, mais également un historique des évolutions topologiques du système, c'est-à-dire les changements de voisinage entre les entités du modèle étudié. En d'autres termes, l'animation doit également porter des informations sémantiques permettant de comprendre l'origine de chaque entité et les transformations que cette dernière a subies.

Ce modèle général d'animation est étudié dans le cadre d'animation de structures géométriques 2D. Ce choix se justifie par le besoin de valider notre modèle d'animation tout en conservant un nombre raisonnable de cas géométriques à traiter. Ces cas nous ont servi à élaborer une méthodologie générale d'utilisation du modèle. Nous montrons que cette contrainte d'une ani-

mation purement $2D$ n'est pas limitative dans l'application choisie. Une extension à la $3D$ est directe au niveau du modèle mais nécessite une étude des cas de collisions ainsi que de leur traitement dans le cas général, comme discuté dans les perspectives.

Afin de mettre en application notre modèle, nous avons cherché un champ d'application orienté vers l'utilisation d'une partition de l'espace. Le laboratoire travaille depuis de nombreuses années en collaboration avec l'Institut Français du Pétrole (IFP) et l'École des Mines de Paris (EMP) sur des sujets connexes à la géologie, et plus particulièrement sur l'analyse et la représentation de la structure du sous-sol. Après quelques échanges, il est apparu que l'un des problèmes que tentent de résoudre les géologues est de reconstituer l'histoire de la création de la croûte terrestre qui a débuté il y a quelque 4,6 milliards d'années. L'étude de la structure du sous-sol permet, sur la base de connaissances physico-chimiques, d'en déduire la composition et, partant, de déterminer l'existence de matières spécifiques sans qu'il soit besoin de réaliser pour cela des forages aussi hasardeux que coûteux. Pour étudier une période de cette histoire, les géologues cherchent donc à définir les phénomènes qui se sont succédé, à partir de la structure actuelle du sous-sol. Pour remonter le temps, ils déforment (« déplient ») les structures existantes en faisant un certain nombre d'hypothèses pour essayer de les aplatis. C'est à cette étape qu'ils ont besoin d'un outil qui permette à terme de valider leurs hypothèses. Représenter l'évolution du sous-sol à travers l'évolution d'une subdivision de l'espace constitue donc la première finalité de cet outil.

Notre application à la géologie fournit des modèles simplifiés de phénomènes agissant au cours du temps sur une scène. Chaque phénomène est décrit à partir d'hypothèses validées par un géologue et peut être facilement étendu par la suite selon le besoin de l'application. Ces modèles sont donc des représentations schématiques de la réalité. La succession de ces phénomènes suffit déjà à décrire un grand nombre de scènes géologiques.

Nos principales contributions sont : (1) de fournir un nouveau modèle générateur d'animation de partition nD de l'espace, basé sur une description textuelle sous forme de scénario ; (2) de proposer un système qui garantit la cohérence de la topologie via une gestion événementielle des changements topologiques ; (3) de détecter les auto-collisions de la partition en exploitant la structure topologique ; (4) d'implanter ce système en $2D$ en incorporant les cas de collisions et leur traitement, cette étude restant indépendante de l'application visée ; (5) de modéliser des phénomènes géologiques à travers des paramètres simples et intuitifs traduits en événements dans notre modèle d'animation.

Ce mémoire est décomposé en sept parties. Le chapitre 2 introduit un état de l'art orienté vers la modélisation à base topologique, les modèles d'animation et vers la composition des deux. Le chapitre 3 présente la partie structurelle de notre modèle et les opérations que l'on peut lui appliquer, dans une optique de *modélisation procédurale* de l'animation. Le chapitre 4 fournit le niveau de contrôle nécessaire à la génération automatique de l'animation à partir d'une description haut-niveau des phénomènes, tout en assurant la cohérence topologique. Le chapitre 5 propose une application en géologie, dans un but démonstratif, en modélisant de manière simplifiée quelques phénomènes géologiques. Enfin, les chapitres 6 et 7 concluent et proposent des orientations futures pour ce travail.

ÉTAT DE L'ART

Comme décrit en introduction, le but de ce mémoire est d'introduire une nouvelle méthode de représentation d'évolutions de structures topologiques à travers le temps permettant d'assurer la cohérence entre le modèle topologique et le modèle géométrique, de représenter l'historique de l'animation et d'y adjoindre des propriétés sémantiques. Les structures topologiques à employer relèvent de la modélisation à base topologique, tandis que leurs évolutions nécessitent le recours à des méthodes d'animation. Aussi, ce chapitre a pour objectif de fournir un tour d'horizon des méthodes de modélisation à base topologique et d'animation. La section sur la modélisation à base topologique a pour vocation de présenter les notions liées à la topologie avant de classer et de survoler les principaux modèles. Le choix du modèle topologique est ensuite détaillé. La section suivante propose une description des grandes classes de méthodes d'animation. Enfin, la dernière section se focalise sur les systèmes d'animation qui se basent sur des modèles topologiques et explique les principales différences entre ces modèles, leurs limites et nos objectifs pour y remédier.

2.1 Modélisation à base topologique

2.1.1 Définitions

La *topologie* (du grec *topos* : lieu et *logos* : étude) combinatoire est une branche des mathématiques qui étudie les relations de voisinage entre objets. Dans le cas de la modélisation géométrique, la topologie s'intéresse à la représentation structurelle d'objets géométriques. Lorsque ces objets sont des polyèdres, les éléments considérés sont les sommets, les arêtes, les faces, les volumes, etc. Ce sont des *cellules topologiques* de dimension croissante. Par définition, des éléments voisins sont dits *adjacents* s'ils sont de même dimension topologique, et *incident*s s'ils sont de dimension différente. De plus, la topologie se focalise sur les caractéristiques structurales ou *invariants topologiques* comme le nombre de trous ou de cavités dans l'objet, l'orientabilité ou la caractéristique d'Euler ($S - A + F$, où S est le nombre de sommets, A le nombre d'arêtes et F le nombre de faces d'un polyèdre) par exemple. Ainsi, la topologie ne tient pas compte de la forme, et de ce fait, classe dans une même catégorie un triangle, un carré ou un

disque, et dans une autre catégorie un cube, un tétraèdre ou une *boule* (sphère volumique). Plus précisément, deux objets appartiennent à la même catégorie s'il existe un isomorphisme pouvant transformer de manière continue (i.e. sans déchirure ni recollement) un objet en un autre de même structure. La figure 2.1 illustre la transformation d'un « doughnut » (tore à un trou) en tasse, où l'anse représente le trou du doughnut [56]. Dans ce cas, une tasse est dite *homéomorphe* (cf. définition 1) à un tore : ils sont de même structure mais ont une forme différente.

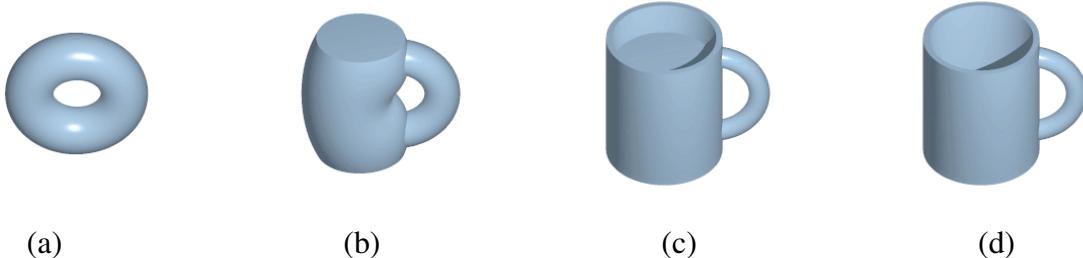


FIG. 2.1 – Une tasse et un tore ont la même topologie : (a) Un tore. (b) Déformation en tasse pleine. (c) La tasse se creuse. (d) Une tasse. (Images extraites du site Wikipédia)

Définition 1 *Une fonction f entre deux espaces topologiques X et Y est dite homéomorphe si et seulement si :*

- f est une bijection ;
- f est continue ;
- la fonction inverse f^{-1} est continue.

Définition 2 *Un espace topologique est un couple (X, T) d'ensembles où T est une collection de sous-ensembles de X , qui satisfait les conditions suivantes :*

- $\emptyset \in T, X \in T$;
- l'union de collections d'ensembles de $T \in T$;
- l'intersection de collections finies d'ensembles de $T \in T$.

La notion d'*homéomorphisme* permet d'introduire la notion de *variété* topologique. Par définition, une variété de dimension n (ou n -variété) est un objet qui, en chaque point, doit être localement homéomorphe à une boule de dimension n (un disque pour les 2-variétés, une boule pour les 3-variétés, voir figure 2.2). Elle appartient à une classe plus large d'objets appelés *quasi-variétés*. Une quasi-variété de dimension n est, de manière intuitive, un objet de dimension n construit par assemblage d'éléments de dimension n le long des cellules de dimension $n-1$ constituant son bord, de telle manière que les cellules de dimension $n-1$ soient incidentes à au plus deux cellules de dimension n .

La figure 2.3 met en exergue le fait que l'assemblage de n -variétés (ici de 3-variétés) ne résulte pas forcément une n -variété, mais peut produire une quasi-variété.

Un objet de dimension n est considéré comme *orientable* si l'objet de dimension $n-1$ représentant son bord sépare l'espace en deux parties distinctes : l'intérieur et l'extérieur de l'objet. Les objets mathématiques non orientables les plus connus sont deux objets surfaciques, la bande de Möbius et la bouteille de Klein. Ces objets ne sont pas représentables en $2D$: pour les visualiser, il faut que leur réalisation géométrique (i.e. leur forme) soit respectivement effectuée en $3D$ et en $4D$ (la bande de Möbius est une surface, d'épaisseur nulle). Les objets de la vie courante sont tous orientables.

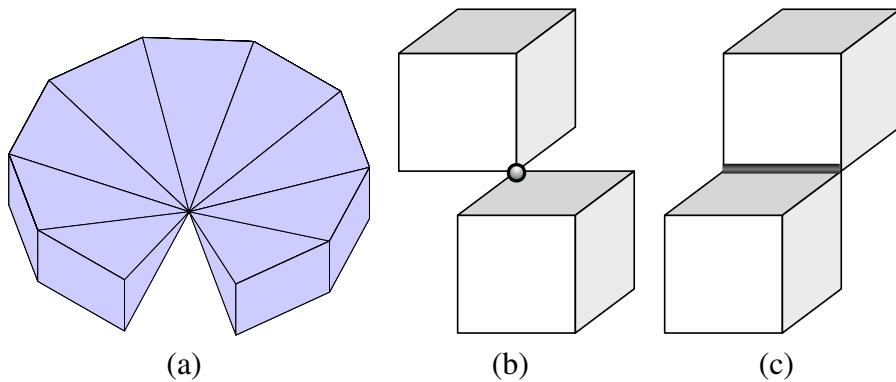


FIG. 2.2 – Exemple d’objets : variété (a) et de non variétés (b) et (c).

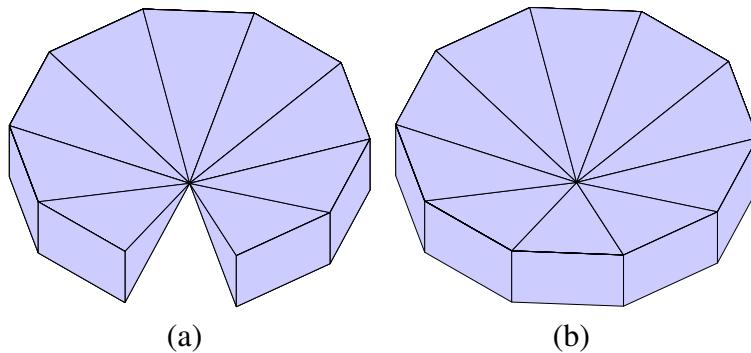


FIG. 2.3 – Exemples de variétés et de non variétés. (a) 3-variété. (b) Après l’ajout du dernier volume, l’objet n’est plus une variété car le point central n’est plus homéomorphe à une boule. Cependant, cet objet appartient toujours à la classe des quasi-variétés.

Un objet subdivisé de dimension n peut être décomposé en éléments appelés *cellules*, de toutes dimensions : sommets (0-cellules), arêtes (1-cellules), faces (2-cellules), volumes (3-cellules), hypervolumes (4-cellules), etc. La modélisation topologique s’intéresse aussi à la représentation des liens entre ces cellules.

Plonger géométriquement une structure topologique dans un espace euclidien \mathbb{R}^n , correspond à lui donner une forme dans cet espace. Un sommet topologique est plongé par un point géométrique à coordonnées dans \mathbb{R}^n . Une arête est soit plongée en traçant un segment entre les points (par abus de langage, les notions de point et de sommet sont confondues dans le présent mémoire) représentant les sommets de son bord (interpolation linéaire), soit en lui associant une courbe (de type Bézier ou spline, par exemple). Un plongement a pour caractéristique de n’admettre aucune auto-intersection. La dimension de l’espace de plongement n’est pas directement liée à la dimension de l’objet modélisé. Par exemple, une sphère creuse, objet de dimension topologique 2, est plongée en 3D.

Pour représenter une scène constituée de plusieurs objets, il est classique d’associer un modèle topologique à chaque objet. Cependant, il est tout à fait possible de mélanger tous les objets dans un même et unique modèle topologique, en se basant en particulier sur les opérations booléennes pour la construction d’une telle structure [33]. Par exemple, de cette manière, une pièce peut être représentée par tous les éléments qu’elle contient : murs, mobilier, livres, ordinateur, air, personnes, etc. L’avantage de cette méthode est que l’on décrit ainsi explicitement les relations de voisinage entre les objets. De plus, l’union des volumes de tous les objets, y compris

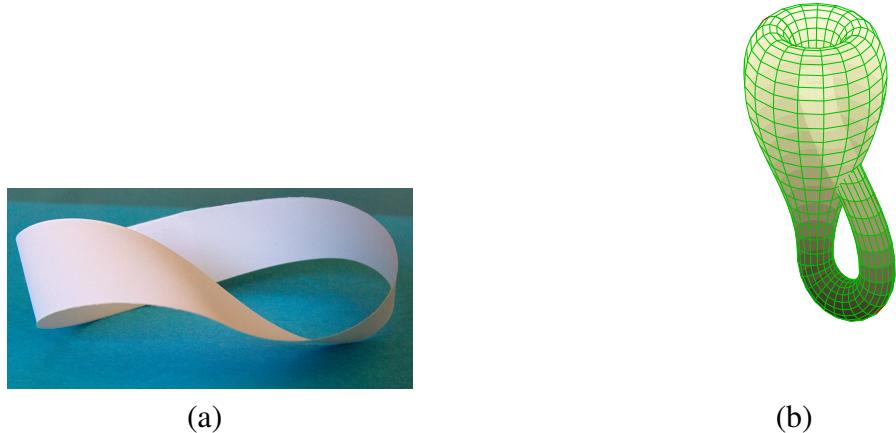


FIG. 2.4 – Objets 2D non orientables : (a) ruban de Möbus, (b) bouteille de Klein.

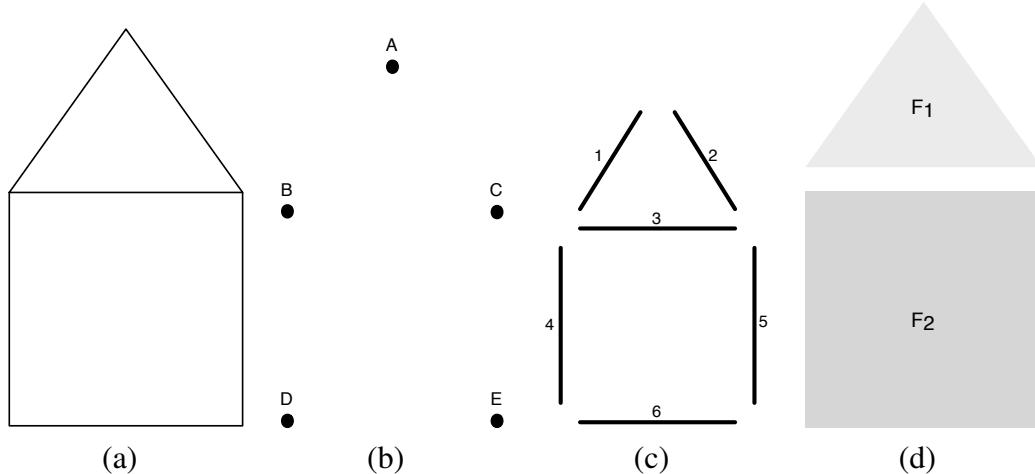


FIG. 2.5 – Décomposition d'un objet en cellules de différentes dimensions. (a) Objet initial constitué d'un triangle collé à un carré. (b) Ensemble des 0-cellules ou sommets. (c) Ensemble des 1-cellules ou arêtes. (d) Ensemble des 2-cellules ou faces.

les zones vides de l'environnement, représente la totalité de l'espace modélisé. Le modèle topologique ne comporte donc qu'une seule composante connexe. Une telle approche revient à modéliser la scène en tant que *partition* de l'espace (cf. définition 3).

Définition 3 Partition d'un ensemble. Une partition d'un ensemble X est un ensemble de sous-ensembles non vides de X tels que tout élément x de X se trouve exactement dans l'un de ces sous-ensembles.

Autrement dit : soit un ensemble X quelconque. Un ensemble P de sous-ensembles de X est une partition de X si :

- $\forall P_i \in P, P_i \neq \emptyset$;
- $\bigcup_{P_i \in P} P_i = X$;
- $\forall P_i, P_j \in P, P_i \cap P_j = \emptyset$.

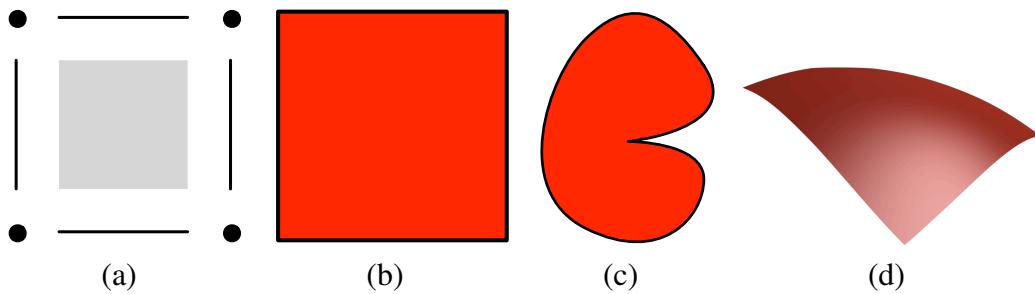


FIG. 2.6 – Exemples de plongements de sommets, arêtes et faces. (a) Structure à plonger composée de quatre arêtes bordant une face et leurs sommets extrémités. (b) Plongement sommet. (c) Plongement courbe 2D. (d) Plongement surfacique 3D.

La figure 2.7 montre un exemple de partition d'un ensemble X en sous-ensembles disjoints P_0 à P_5 . L'union des P_i représente bien la totalité de l'ensemble X .

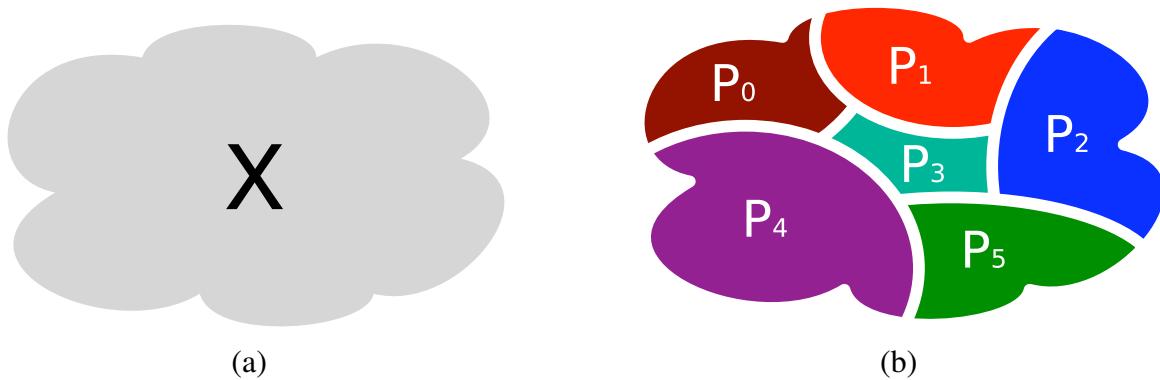


FIG. 2.7 – Partition d'un espace.

2.1.2 Modèles topologiques

Classiquement, deux types de modélisation co-existent : les modèles *CSG* (Constructive Solid Geometry) et les modèles *B-Rep* (Boundary Representation). Le principe des CSG [63] est de définir un objet géométrique comme étant le résultat d'une suite d'opérations booléennes (intersections, unions, différences) appliquées à des primitives géométriques (telles que des sphères ou des prismes). Un objet CSG est stocké sous la forme d'un arbre binaire où les feuilles sont des primitives et dont les autres nœuds sont des opérations booléennes. Un objet CSG peut être visualisé par un processus de lancer de rayons ou en évaluant son bord par analyse de son arbre. L'avantage d'une telle méthode est qu'un objet peut facilement être réévalué en modifiant les paramètres des primitives de base. L'inconvénient est que l'objet n'est pas représenté explicitement, il est donc difficile d'attacher des propriétés (physiques, colorimétriques, etc.) à ses différents composants et de les exploiter directement sans évaluer l'objet dans sa globalité.

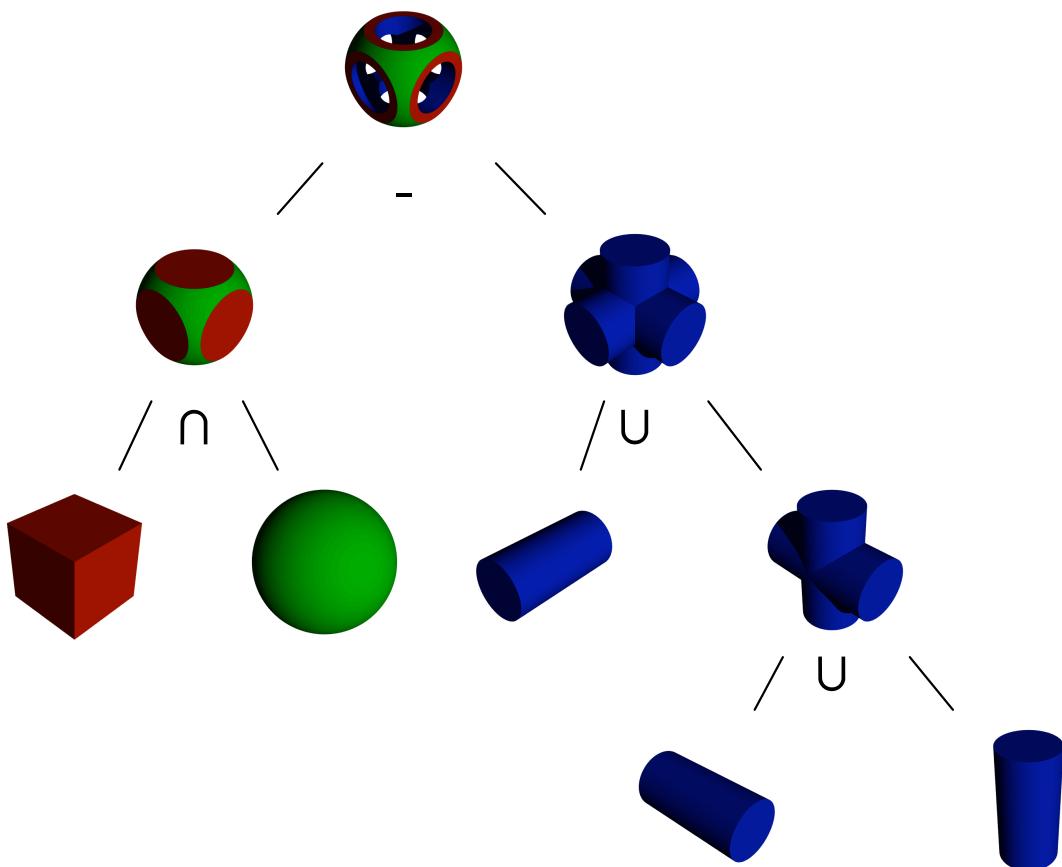


FIG. 2.8 – Exemple de CSG : construction d'un objet à partir de deux unions, une intersection et une différence.

Les modèles topologiques décrivent les structures des objets en cellules de différentes dimensions liées par des relations d'adjacence et d'incidence. Le domaine de la modélisation géométrique à base topologique s'intéresse aux structures représentant ces informations.

Les modèles B-Rep résolvent en partie les problèmes de la CSG en représentant explicitement les objets par leur bord. En B-Rep, un objet 3D est représenté par la surface qui entoure son volume. Les B-Rep sont répartis en deux grandes classes, les *graphes d'incidences* (GI) et les *modèles ordonnés* [13].

Dans le domaine de la géométrie, le plus ancien modèle et le plus utilisé pour supporter la topologie est le graphe d'incidence (GI)[67]. Un GI représente explicitement les cellules de chaque dimension par un ensemble de nœuds liés par des relations d'incidence au travers d'un graphe orienté (cf. figure 2.9). Les relations d'adjacence sont obtenues par le parcours de cet arbre. Un des problèmes des GI est que plusieurs objets peuvent avoir la même représentation. Par exemple, les GI ne permettent pas de représenter la multi-incidence de cellules (cf. figure 2.10). Une méthode pour résoudre ce problème est d'adoindre au modèle une notion d'ordre (au sens littéral du terme) sur les $(i - 1)$ -cellules incidentes à la i -cellule considérée.

Contrairement au GI, les modèles ordonnés utilisent un élément unique, sur lequel s'appliquent des relations de voisinage. Dans ces types de modèle, les i -cellules sont définies implicitement.

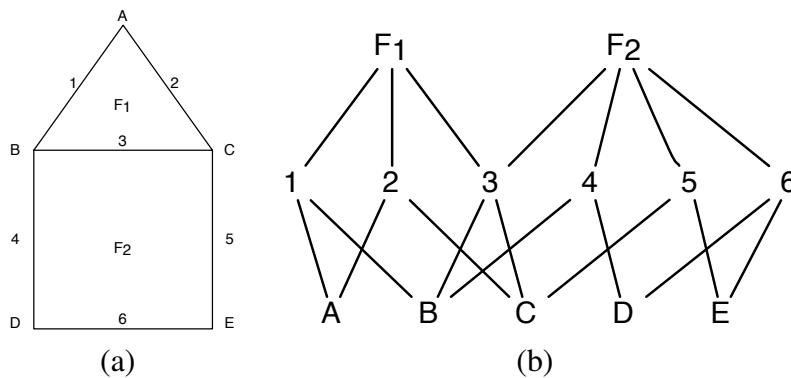


FIG. 2.9 – Exemple de graphe d'incidence.

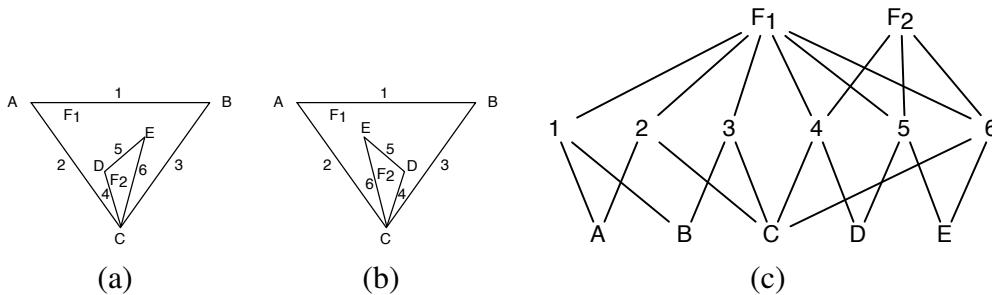


FIG. 2.10 – Exemple d'ambiguïté : un même graphe d'incidence pour deux objets différents.

Dans la suite de ce chapitre, nous proposons de donner une vue d'ensemble des modèles ordonnés. Notre but est de représenter une partition quelconque de l'espace avec des cellules de structure non régulière, c'est pourquoi nous ne détaillons pas les structures topologiques simplciales [51] (à base de sommets, arêtes, triangles et tétraèdres) ou simploïdales [70] (structures construites à partir du produit cartésien de formes simples : simplexes de dimension n).

Un des premiers modèles que l'on peut trouver dans la littérature s'appelle les *arêtes ailées* (*winged edge*) [4]. Ce modèle ordonné permet de représenter des objets 2D orientables sans bord. L'élément topologique de base est l'arête, autour de laquelle l'objet est structuré. Dans ce modèle, une arête est munie de huit pointeurs : deux vers les sommets qui lui sont incidents, deux vers les faces qui lui sont incidentes et quatre vers les arêtes qui lui sont adjacentes et qui appartiennent à ses faces incidentes (cf. figure 2.11(a)). Ce modèle a été étendu avec le modèle des *arêtes radiales* (*radial edges*) [77] afin de représenter des objets 3D orientables qui peuvent

ne pas être variétés. Dans ce modèle, on s'intéresse au classement des faces autour d'une arête. Pour représenter une subdivision 2D, on peut aussi citer le modèle des *demi-arêtes* (*half-edges*) [76]. Dans ce modèle, l'élément de base est la demi-arête. Chaque demi-arête est orientée et possède trois pointeurs, un vers l'arête précédente, un autre vers la suivante de sa face incidente et le dernier vers la demi-arête qui lui est adjacente (cf. figure 2.11(a)).

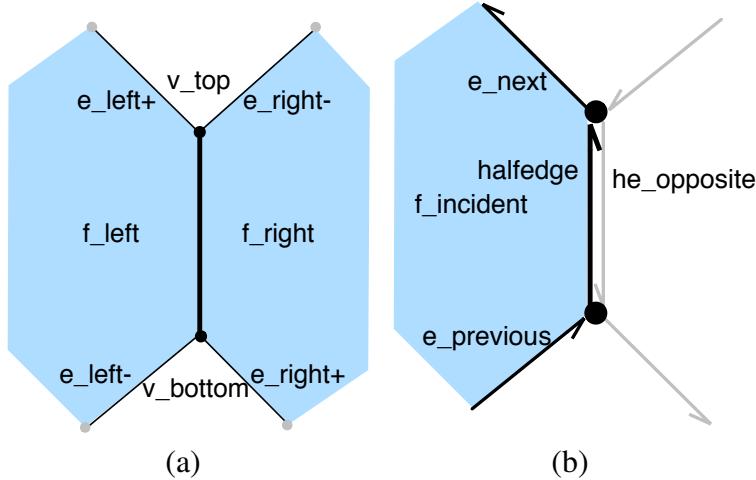


FIG. 2.11 – (a) Schéma de la structure des arêtes ailées. (b) Schéma de la structure des demi-arêtes.

Les modèles cités ci-dessus n'ont pas de définition formelle et sont limités aux dimensions 2 et 3. Un premier modèle formel est décrit par les cartes combinatoires [24] qui se basent sur un élément abstrait appelé *brin*, qui, en 2D, correspond intuitivement à une demi-arête. Les cartes combinatoires permettent de représenter des objets orientés sans bord de dimension n . Ce modèle a été étendu par les cartes généralisées de dimension n (n -g-cartes, également appelées g-cartes pour simplifier la lecture) [42] pour représenter les objets orientables ou non, avec ou sans bord. Les *cells-tuples* [13] sont introduits parallèlement aux g-cartes et sont équivalents aux g-cartes sans bord. Un cell-tuple est un $(n + 1)$ -uplet de cellules (c_0, \dots, c_n) , où c_i représente les cellules de dimension i , muni d'un opérateur $switch(c_{k-1}, c_k, c_{k+1})$ permettant de connaître l'unique cellule c'_k telle que (c_{k-1}, c'_k, c_{k+1}) lui soit incident. Pour étendre ces modèles, sont apparues les chaînes de cartes [26, 44] qui modélisent des objets non variétés en représentant explicitement chaque i -cellule de la scène par une quasi-variété, et en associant chaque i -cellule aux $(i - 1)$ -cellules de son bord par des relations spécifiques.

Le choix d'une structure topologique repose principalement, d'une part sur l'application visée car c'est d'elle dont dépend la complexité des algorithmes (conception, temps de calcul, espace mémoire) et d'autre part sur la classe d'objets à représenter. Dans notre cas, nous voulons décrire l'évolution d'une partition de l'espace en dimension n avec des éléments de type quasi-variété et de forme quelconque. Dans ce cadre, nous avons choisi d'utiliser les g-cartes. La première raison est qu'un plus grand nombre d'opérations a été défini sur les modèles des (g-)cartes que sur celui des cell-tuples. La seconde est que les g-cartes sont définies de manière homogène, à partir d'un seul élément abstrait (brin) et d'un type d'opérateur unique (involution, définition 4), contrairement aux cartes qui utilisent deux types d'opérateurs (permutation et involution) dont l'interprétation diffère suivant les définitions (i.e. suivant la position de la permutation). Ces caractéristiques facilitent la définition des opérations, bien qu'elles augmentent significativement la taille en mémoire de la structure de données (environ par deux : si la g-carte modélise un objet sans bord, ce dernier nécessite deux fois plus de brins qu'en utilisant

une carte, et il y a un pointeur de plus par brin dans le modèle des g-cartes). Le modèle peut être par la suite optimisé en mémoire en transformant les g-cartes en cartes si les scènes étudiées contiennent des structures orientées et sans bord, ou tout autre structure adaptée à la dimension de la scène à générer. De plus, comme cela est décrit plus loin, les g-cartes permettent de désigner de manière naturelle et non ambiguë des cellules de dimension quelconque à partir des brins (i.e., désigner le sommet x incident à l'arête y incidente à la face z). Enfin, puisque nous désirons manipuler des objets quasi-variétés uniquement, il est inutile de recourir aux chaînes de cartes, car ces dernières nécessitent plus de brins que les g-cartes pour représenter une même quasi-variété. Dans la suite de cette section, nous nous focalisons sur les g-cartes car c'est le modèle que nous avons choisi d'utiliser.

2.1.3 Les n-G-Cartes

Les g-cartes [42, 43] de dimension n ou n-g-cartes représentent les objets par leur bord (B-Rep). Elles modélisent les quasi-variétés cellulaires orientées ou non, avec ou sans bord. Les objets géométriques sont subdivisés en cellules (sommets, arêtes, faces, etc.) reliées entre elles par des relations d'adjacence/incidence (Figure 2.12). Les cellules sont modélisées à l'aide d'éléments de base appelés brins, liés entre eux. Intuitivement, on peut voir le brin comme étant un sommet attaché à une arête incidente à ce sommet, elle-même attachée à une face incidente à l'arête, elle-même attachée à un volume incident à la face, etc. La figure 2.12 illustre une telle décomposition sur l'exemple d'une maison schématique constituée d'un triangle et d'un carré. Tout d'abord, l'objet est décomposé en faces en décollant les faces F_1 et F_2 , puis ces faces sont liées par une relation d'adjacence de dimension 2 notée α_2 (cf. figure 2.12(b)). Les faces sont décomposées en arêtes liées par une relation d'adjacence de dimension 1, noté α_1 (cf. figure 2.12(c)). Les arêtes sont enfin décomposées en sommets liés par la relation α_0 (cf. figure 2.12(d)). Sur ce dernier schéma, on observe que par construction, l'arête centrale a été décomposée en quatre éléments liés deux à deux par α_0 et par α_2 : l'objet respecte bien la définition des quasi-variétés cellulaires.

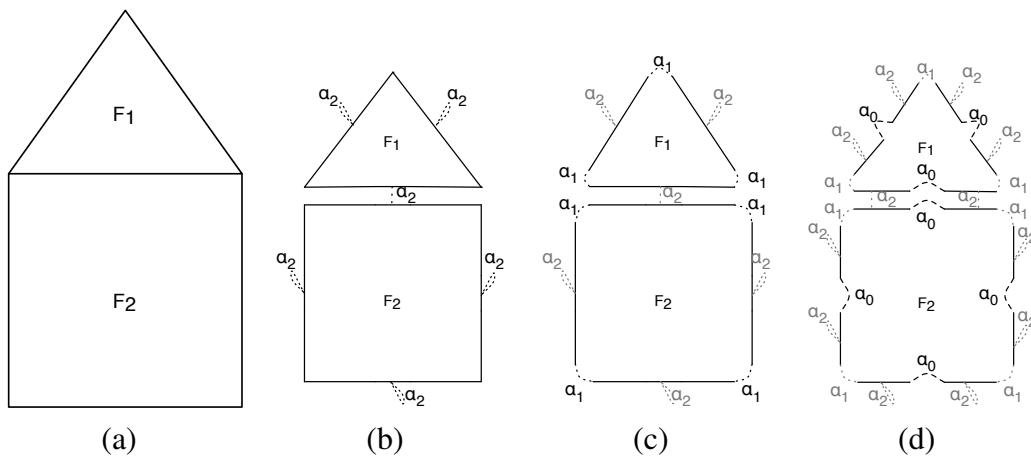


FIG. 2.12 – Décomposition d'une maison schématique (a) vers le modèle g-carte (d). (a) Objet 2D à décomposer. (b) Décomposition de l'objet en faces F_1 et F_2 . (c) Décomposition de chaque face en arêtes. (d) Décomposition de chaque arête en sommets.

De manière plus formelle, une g-carte est définie par un ensemble fini de brins liés entre eux par des involutions α_i , auxquelles on ajoute une contrainte de cohérence assurant que le modèle représente des quasi-variétés cellulaires.

Définition 4 *Involutive.* Soit f une application d'un ensemble E dans lui-même. f est une involution si et seulement si $\forall x \in E, f(f(x)) = x$.

Définition 5 *n-g-cartes.* Une g-carte de dimension n , ou n -g-carte, est un $(n + 2)$ -uplet $G = (B, \alpha_0, \dots, \alpha_n)$ tel que :

- B est un ensemble fini de brins ;
- $\alpha_0 \dots \alpha_n$ sont des involutions sur B ;
- $\alpha_i \alpha_j^{-1}$ est une involution pour $0 \leq i < i + 2 \leq j \leq n$.

Une g-carte est dite sans bord si elle ne comporte pas de point fixe. Autrement dit :

Définition 6 Une n -G-Carte $G = (B, \alpha_0, \dots, \alpha_n)$ est fermée (ou sans bord) si et seulement si $\forall i \in \{0, \dots, n\}, b \alpha_i \neq b$.

Manipuler une g-carte nécessite de parcourir ses brins. Pour cela, la notion d'*orbite* est nécessaire. Intuitivement, une orbite est l'ensemble des brins que l'on peut atteindre à partir d'un brin par un ensemble d'involutions et de permutations (composition d'involutions). Cette notion permet de définir la notion d'*i-cellule*. Les orbites permettent de regrouper les brins en cellules de différentes dimensions (i-cellules).

Définition 7 *Orbite et i-cellule.* Soit $\{\prod_0, \dots, \prod_n\}$ un ensemble de permutations sur B . L'orbite d'un élément $b \in B$ liée à cet ensemble de permutations est $\langle \prod_0, \dots, \prod_n \rangle(b)$, où $\langle \prod_0, \dots, \prod_n \rangle$ correspond au groupe de permutations engendré par \prod_0, \dots, \prod_n .

Soient $b \in B$, $N = \{0, 1, \dots, n\}$ et $i \in N$. L'*i-cellule* incidente à b est l'orbite $\langle \rangle_{N-\{i\}}(d) = \langle \alpha_0, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n \rangle(d)$.

La figure 2.13(b) montre les différentes 0-cellules (sommets) composées des brins obtenus par applications successives des involutions α_1 et α_2 . Les figures 2.13(c) à 2.13(d) illustrent les 1-cellules (arêtes) et les 2-cellules (faces).

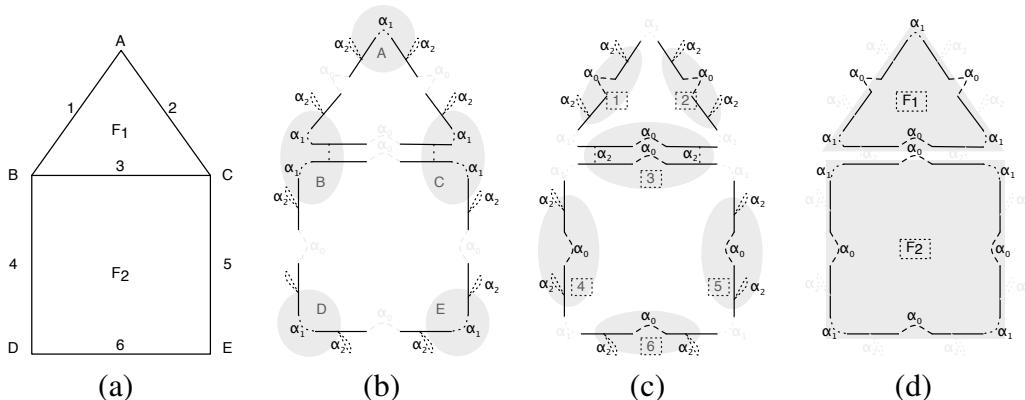


FIG. 2.13 – (a) Une maison schématique. (b) Orbites sommets. (c) Orbites arêtes. (d) Orbites faces.

¹ $\alpha_i \alpha_j$ est la notation qui correspond à la composition $\alpha_j \circ \alpha_i$. $b \alpha_i \alpha_j$ est l'application de cette composition à un élément b de B .

Définition 8 *Orientabilité.* Soit une g-Carte G connexe et fermée. G est orientable si et seulement si elle contient exactement deux orbites pour l'ensemble de permutations [44] : $\{\alpha_0\alpha_1, \alpha_0\alpha_2, \dots, \alpha_0\alpha_n\}$.

La figure 2.14 illustre une partition de l'espace. Elle contient exactement deux orbites (figures 2.14(c) à 2.14(d)) pour l'ensemble de permutations $\{\alpha_0\alpha_1, \alpha_0\alpha_2\}$, elle est donc orientable.

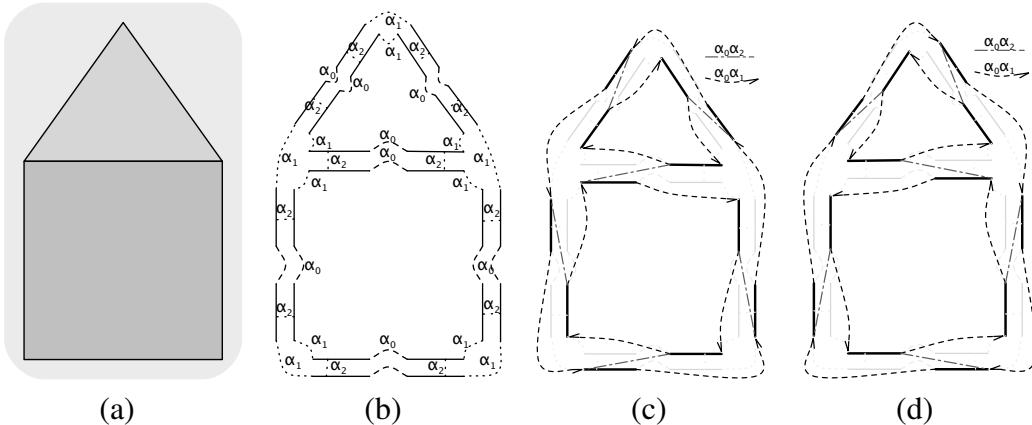


FIG. 2.14 – Orientabilité : (a) une partition de l'espace, (b) la 2-g-carte associée, (c) première composante connexe, (d) seconde composante connexe.

Afin de simplifier et d'améliorer la lisibilité des figures, la convention de la figure 2.15 sera utilisée.

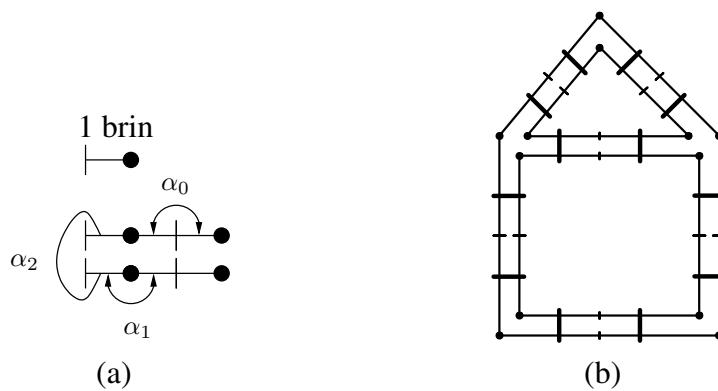


FIG. 2.15 – (a) Convention utilisée pour la représentation d'un brin et des liaisons. (b) Exemple de deux faces collées entre elles et leur représentation par une 2-g-carte fermée.

Une structure topologique représente la structure statique d'un objet. Pour la rendre dynamique, il faut recourir à des modèles d'animation. Nous les étudions dans la section suivante.

2.2 Modèles d'animation

Nous souhaitons animer une structure topologique, il faut donc s'appuyer sur un modèle d'animation, c'est-à-dire un modèle capable de fournir une description de l'évolution des paramètres des objets. Plus précisément, une animation consiste en la modification d'entités (objets, caméra, lumières) d'une scène au cours du temps. Un objet est modifié en faisant évoluer certaines de ses caractéristiques comme sa position (translation), son orientation (rotation), sa taille (homothétie), sa forme (déformation sans déchirure), ses propriétés photométriques (couleur, transparence) ou, ce qui nous intéresse plus particulièrement, sa structure topologique.

Historiquement, les premières animations datent du début du 19ème siècle, obtenues avec des jouets optiques tels que le zootrope (William George Horner - 1834), le folioscope (1834 ou 1868, inventeur et date soumis à débat) et le praxinoscope (Émile Reynaud - 1876, première utilisation d'une bande perforée en 1879). Ces animations sont composées d'une suite chronologique d'images représentant une scène à différentes dates. Leur projection (ou réflexion, etc.) à grande vitesse donnent l'illusion du mouvement. Pour visualiser une animation, il faut donc, a priori, générer une séquence chronologique d'images représentant la scène à intervalles réguliers. Une telle séquence d'animation peut être produite par un ordinateur suivant les directives d'un animateur. Les deux grandes familles de méthodes d'animation s'articulent autour des modèles *descriptifs* et des modèles *générateurs*.

2.2.1 Modèles descriptifs

Les modèles descriptifs reposent sur la description explicite de l'évolution de chaque paramètre des objets au cours du temps. Les modèles de type animation par *images-clefs* et animation *paramétrique* en sont des exemples. Pour pouvoir définir mathématiquement cette évolution, l'animation par images-clefs [15] (ou *keyframe*) consiste à fournir en entrée une série d'images à des dates données et un algorithme se charge de calculer les images intermédiaires entre les images-clefs par déformations de formes libres [19], interpolations ([66], etc.) ou morphismes [69, 71]. [38] proposent un état de l'art sur les métamorphoses d'objets sur lequel s'appuient des travaux plus récents, [39] et [73] par exemple. Cette approche pose essentiellement deux problèmes : d'une part, deux images n'ont pas forcément la même structure et l'interpolation géométrique ne suffit plus ; d'autre part, les mouvements interpolés ne sont généralement pas naturels [37]. L'animation paramétrique consiste à attribuer des valeurs aux propriétés (orientation, position, photométrie, etc.) des entités à différentes dates. Le système calcule alors les valeurs intermédiaires par interpolation. Dans les deux cas, ces méthodes nécessitent de préciser un grand nombre de paramètres, ce qui complexifie particulièrement la tâche de l'animateur. Il est possible de créer et d'exploiter des bibliothèques spécifiant les évolutions cohérentes d'un ensemble de paramètres. La technique du *Motion Capture* permet par exemple de mesurer l'évolution des paramètres sur des personnages réels puis de stocker ces données définitivement.

Ces approches ont comme point commun de nécessiter un nombre relativement élevé de paramètres, tant pour construire que pour contrôler les animations, que l'utilisateur manipule avec plus ou moins de facilité. Un autre ensemble de modèles, dits *générateurs*, s'attache à diminuer le nombre de paramètres mis en jeu.

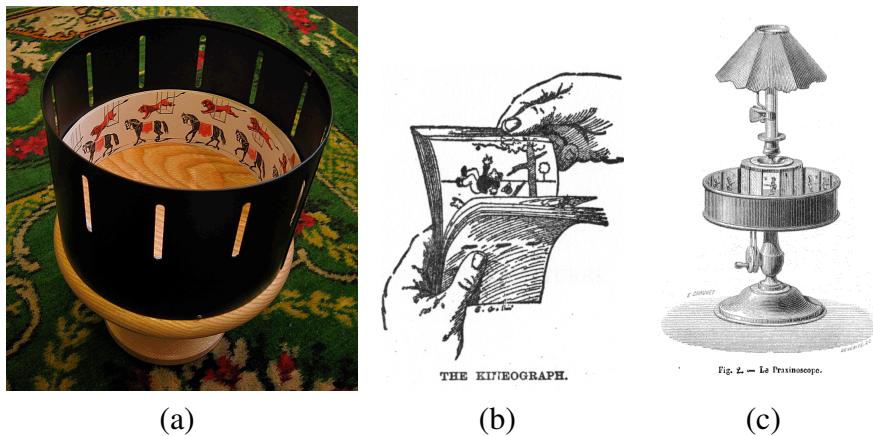


FIG. 2.16 – (a) Réplique d'un zootrope de la période victorienne (photographie d'Andrew Dunn, 2004). (b) Folioscope (illustration de 1886 par John Barnes Linnet). (c) Praxinoscope d'Émile Reynaud (image de la revue des sciences, 1879, num. 296, p. 133).

2.2.2 Modèles générateurs

Comme nous venons de le constater, le principal problème des modèles descriptifs provient de la nécessaire description des évolutions d'un nombre important de paramètres. Pour déterminer ces évolutions, une solution est d'utiliser des modèles, dits *générateurs*, capables d'obtenir ces évolutions par calcul, en ne demandant à l'animateur qu'une quantité restreinte d'information, voire des directives de haut-niveau. Ces informations sont spécifiques à un domaine d'application donné et les directives associées sont automatiquement traduites par le modèle en paramètres adaptés à l'évolution de chaque objet.

Plusieurs approches sont possibles : soit le système est doté d'un ensemble de règles permettant de déterminer l'évolution de certains paramètres à partir d'autres paramètres, on parle alors d'*animation comportementale* ; soit le système utilise des lois, généralement issues de la mécanique, qui calculent les évolutions à partir de conditions initiales, on parle alors d'*animation physique*.

2.2.2.1 L'animation comportementale

L'animation comportementale se base sur un ensemble de règles permettant de déterminer précisément l'évolution de chaque paramètre à partir de l'évolution d'un sous-ensemble de ces paramètres, voire à partir de directives haut-niveau indiquées par l'animateur. Souvent, cette méthode est employée pour faire évoluer des êtres vivants (animaux, plantes), pris individuellement (par exemple, l'animateur indique simplement une trajectoire et l'évolution des paramètres décrivant le déplacement du personnage est automatiquement calculée) ou en groupe [64, 28, 61] (représentation de comportement de masse comme les volées d'oiseaux ou les bancs de poissons). Les lois peuvent avoir des bases mathématiques (telles que la minimisation des distances entre diverses entités) ou s'appuyer sur des outils d'intelligence artificielle. Par exemple, une problématique comme la simulation de foule ou de flux de personnes peut ainsi être abordée selon ces différents angles [54].

2.2.2.2 L'animation physique

L'animation est dite physique si elle repose sur une simulation mécanique de la scène à animer. Ce type d'animation est généralement utilisé pour obtenir des animations les plus réalistes possibles. C'est en spécifiant les données de départ et en résolvant des équations différentielles indiquant l'évolution des paramètres que l'animation est obtenue. Elle nécessite non seulement de savoir résoudre les équations, mais aussi de pouvoir trouver et calculer les interactions entre les objets, i.e. détecter et traiter les collisions. Suivant les propriétés mécaniques des objets, on utilise différents modèles d'évolution.

Les solides rigides ne subissent aucune déformation. En 3D, ils sont définis par six paramètres, appelés degrés de liberté : trois pour décrire la position de l'objet (translation) et trois autres décrivant son orientation (rotation). Ces objets sont généralement soumis à des lois physiques simples (lois de Newton) prenant la forme d'équations du second degré pilotant pour certaines la position et pour les autres l'orientation.

D'autres solides peuvent se déformer, sous l'effet de forces extérieures (contraintes). Certains de ces solides se déforment tout en conservant leurs caractéristiques topologiques. Cette famille comprend en particulier les corps élastiques, qui reprennent leur forme originale une fois les contraintes levées. D'autres corps peuvent subir des modifications topologiques, lors de découpes, déchirures, brisures, etc. Pour calculer l'évolution de ces objets structurés, l'animateur a souvent recours à des modèles particulaires de type masses/ressorts [48, 59, 1, 25] où l'objet est discréteisé en un ensemble de particules (masses) reliées entre elles par des liaisons d'interactions (ressorts), ou bien à des simulations par éléments finis [18]. La figure 2.17 présente un exemple d'assemblage de masses par des ressorts tiré de [59] et capable de simuler par exemple le comportement de tissus.

Enfin, il existe des corps déformables peu structurés (les pâtes par exemple) ou sans structure (sable, fluides, gaz, plasmas). Dans ce cas, il est courant d'avoir recours à un système de particules [62, 21], mais sans maillage (les fonctions d'interaction sont alors définies entre toutes les particules).

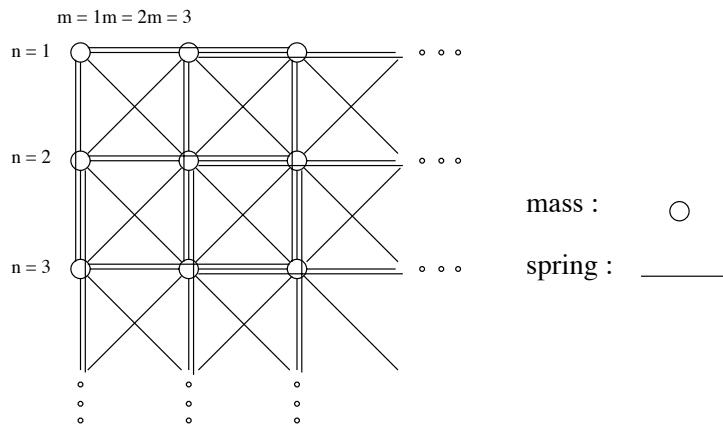


FIG. 2.17 – Exemple de masses-ressorts (image extraite de [59]).

Généralement, durant une animation, les objets interagissent entre eux par des contacts très brefs (chocs) ou prolongés. La tâche de détecter et de traiter ces contacts revient au gestionnaire de collisions. C'est essentiellement par ce biais que le comportement des objets peut être modifié en cours de simulation. Pour traiter les interactions, le système teste l'ensemble des

intersections entre couples d'objets tous les $t + \Delta T$. Le calcul est fastidieux mais peut être optimisé par l'utilisation de structures accélératrices : partition de l'espace (kd-trees, arbres BSP, etc.) limitant le nombre de couples à tester aux objets situés dans une même zone de l'espace ou volumes englobants (boîtes, sphères, enveloppes convexes) simplifiant la géométrie de l'objet et donc la détection de non-collision [75, 31].

Pour simuler un ensemble d'objets, on pose les équations qui gouvernent leur comportement et on résout numériquement ces équations, en calculant l'état du système à intervalles de temps réguliers. La simulation apparaît alors comme une boucle infinie où, à chaque étape, on calcule le nouvel état de chaque objet, en fonction du ou des états précédents [52]. Le gestionnaire de collision intervient à chaque tour de boucle pour déterminer quels sont les objets en collision et calculer les forces engendrées.

Cependant, dans certains cas de simulation, la résolution des équations peut se faire de façon formelle. C'est-à-dire qu'à partir des conditions initiales, et sous réserve que l'objet ne subisse pas de perturbation (collisions), sa trajectoire, i.e. la fonction temporelle décrivant l'évolution des paramètres, est complètement déterminée. L'état d'un objet étant connu tant qu'il n'est pas perturbé, le système va donc essentiellement chercher à prédire quand les collisions surviennent. La simulation s'articule autour d'un système événementiel : ce dernier recense les événements de collision et les traite dans l'ordre chronologique. Le temps de la simulation est donc ici discrétilisé également, mais varie de façon irrégulière, au gré des collisions détectées par le système. Cette technique est souvent utilisée dans les simulations de dynamique moléculaire, par exemple [23, 16]. En pratique, le système débute par une phase d'initialisation où il calcule, comme dans le cas d'une discrétilisation du temps, toutes les interactions possibles entre couples (*acteur, cible*) d'objets. Pour chaque acteur, le système insère l'interaction potentielle entre le couple (*acteur, cible*) dont la date est la plus proche dans une *file à priorité* ordonnée suivant les dates. Cette phase d'initialisation passée, l'animation peut débuter. Le système intervient seulement lorsque la date en cours correspond à la première valeur de la file à priorité. Dans ce cas, on défile l'événement correspondant puis on vérifie sa validité (a t-il vraiment lieu ?). Si l'événement est toujours valide, le système le traite. Dans tous les cas, on recalcule les interactions potentielles pour tous les couples ayant comme acteurs les objets modifiés. Cette méthode est globalement plus efficace que la méthode à pas de temps réguliers car le nombre d'interactions à vérifier est moins important. En revanche, il faut pouvoir prédire les interactions entre objets, ce qui n'est pas toujours possible, suivant leur forme et leur trajectoire.

La figure 2.18 présente deux cas d'événements invalides. Le premier se produit quand l'acteur rencontre une entité avant sa cible (dans la figure 2.18(a), A devrait rencontrer B en premier, mais il rencontre C avant). Le second se produit quand la cible rencontre une entité avant l'acteur (dans la figure 2.18(b), B devrait rencontrer A, mais rencontre C avant). L'invalidité des événements peut être causée pour deux raisons. La première est due à la prédiction des collisions potentielles : dans la première figure, il était prévu que A rentre en collision avec B mais la rencontre avec C annule finalement cette collision (on considère simplement les collisions potentielles, donc on calcule des collisions par paires d'objets). La seconde est due au changement de trajectoire d'un acteur causé par un nouvel événement.

2.2.3 Choix de l'espace de représentation spatio-temporel

Qu'il s'agisse des modèles descriptifs ou générateurs, l'espace de représentation de l'animation a son importance et en particulier, la façon dont cette représentation considère le temps.

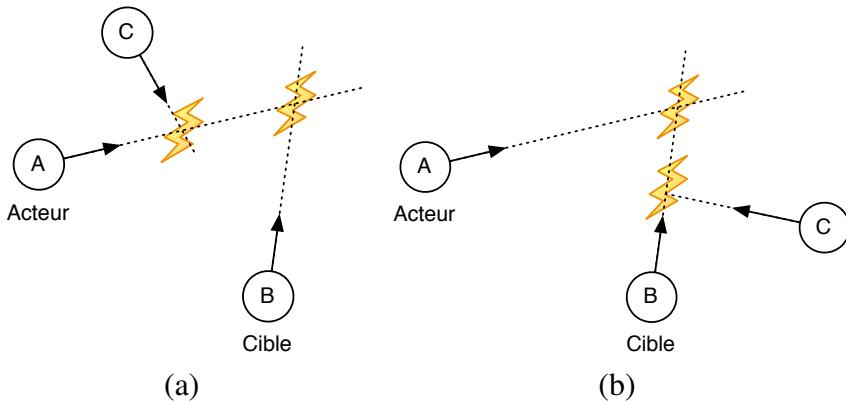


FIG. 2.18 – Deux cas d'événements invalides entre A et B. (a) L'acteur A rencontre C avant B. (b) La cible B rencontre C avant A.

Soit le temps est considéré comme un paramètre des fonctions liées au mouvement. On parle alors d'une représentation « $nD + \text{temps}$ », où l'on décrit l'évolution temporelle d'objets spatiaux. Mais il existe une autre représentation : la forme « $(n + 1)D$ », dans laquelle la variable de temps est considérée de la même manière que les autres variables spatiales.

Dans la représentation « $nD + \text{temps}$ », l'objet est plongé dans un espace à n dimensions spatiales. Le mouvement est alors défini sous forme analytique, en exprimant par exemple la trajectoire des entités considérées.

Dans l'approche $(n + 1)D$, l'objet est plongé dans un espace comportant une dimension supplémentaire représentant le temps [6]. Cette représentation montre l'évolution d'un objet spatial au cours du temps, et englobe à la fois la forme et la trajectoire en représentant toute la portion d'espace couverte par l'objet au cours du temps. L'*objet spatio-temporel* ainsi défini peut être visualisé en des instants donnés, grâce à une *coupe temporelle*. La coupe temporelle correspond à l'intersection d'un hyperplan perpendiculaire à l'axe temporel avec l'objet spatio-temporel (cf. figure 2.19). L'animation consiste alors à faire évoluer l'hyperplan de coupe progressivement le long de l'axe du temps dans le sens croissant.

Par exemple, une animation simple consiste à représenter un objet spatial immobile durant un certain intervalle de temps. Il suffit d'extruder cet objet par un « segment temporel », i.e. un segment parallèle à l'axe temporel défini entre les instants t_0 et t_1 : l'animation résultante fait apparaître l'objet spatial à t_0 , qui reste immobile avant de disparaître à t_1 . Si l'animation doit représenter l'objet spatial en mouvement, l'extrusion se fait le long d'une courbe décrivant la trajectoire souhaitée. En associant des coefficients d'homothétie et de rotation aux points de la courbe d'extrusion, on élargit la gamme d'effets appliqués à l'objet en mouvement. Plusieurs types d'animations sont possibles et dépendent de la manière de construire et de transformer l'objet $(n + 1)D$. Mais les opérations d'extrusion ne permettent pas de modifier la structure topologique d'un objet.

Pour ce faire, le principe général consiste à construire un objet $(n + 1)D$ à l'aide d'opérations reprises de la modélisation nD et appliquées en $(n + 1)D$, en considérant la dimension temporelle comme une dimension spatiale classique. Ensuite, l'application de la coupe temporelle sur l'objet $(n + 1)D$ permet de faire évoluer la structure de l'objet nD sous-jacent. Prenons l'exemple de la figure 2.19 qui modélise l'animation $(1 + 1)D$ d'un point se transformant en segment. L'objet $(n + 1)D$ est ici un triangle spatio-temporel dont une arête est parallèle à l'axe du temps t , et une autre parallèle à l'autre axe x . En effectuant des coupes temporelles,

on observe un point qui se dilate au cours du temps. Si un côté du triangle n'est pas parallèle à l'axe x , le point va se dilater en segment, puis se contracter. La principale difficulté de cette approche est de contrôler la synthèse de l'animation, i.e. construire l'objet $(n + 1)D$ représentant précisément l'animation que l'on souhaite obtenir. Plusieurs méthodes s'attaquent à ce problème et proposent des solutions reposant sur des déformations [8, 5], des constructions par épaississement [9, 10] ou par produit cartésien [45].

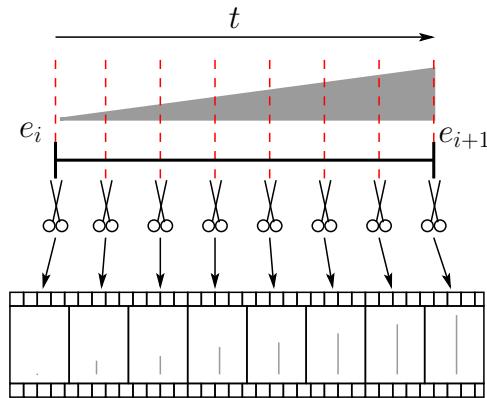


FIG. 2.19 – Exemple d'un objet spatio-temporel $(1+1)D$. Un triangle, représentant la dilatation d'une arête au cours du temps, est défini entre l'événement de début e_i et l'événement de fin e_{i+1} . Les paires de ciseaux sont placées aux instants de coupe. La pellicule contient les images à afficher.

2.3 L'animation à base topologique

Certains travaux ont cherché à exploiter les structures topologiques dans le cadre de l'animation. Ils étudient les objets à structure dynamique, c'est-à-dire qui impliquent des créations ou disparitions d'entités, des fusions, des déchirures, etc. Pour cela, de nombreux travaux se sont appuyés sur la théorie des langages pour la représentation de la structure, et considèrent l'animation en termes de transformations successives, en ayant recours à des systèmes de réécriture. Nous présentons diverses approches dans cette section.

2.3.1 les L-Systèmes

Un L-système [46, 47, 60] est une grammaire formelle utilisée afin de modéliser des processus de développement et de prolifération de plantes et de bactéries. Cette grammaire formelle comprend un alphabet V , un ensemble de constantes S , un axiome de départ ω et un ensemble de règles P d'évolution du système. Les images de la figure 2.20 sont obtenues par une interprétation géométrique de type *turtle* du L-système 1 défini ci-dessous. L'interprétation *turtle* consiste à associer un symbole à chaque mouvement d'une tortue. La tortue peut avancer (le mouvement est généralement représenté par une lettre, ici "F"), mémoriser une position (symbole "["), y retourner (symbole "]") ou changer de direction (symboles "+" ou "-" pour tourner à gauche et tourner à droite).

L-Système 1 (*Arbre 1*)

- $V = "F"$, $S = "]", [$, $+, -"$,
- $\omega = "F"$,
- $P = "F" \longrightarrow F[+F]F[-F]F"$.

L'image de la figure 2.21 est obtenue par l'interprétation du L-système 2 sur laquelle on peut observer des ensembles de branches qui s'auto-intersectent. En effet, les L-systèmes ne déterminent que la réécriture de mots et ne traitent pas de la cohérence géométrique.

L-Système 2 (*Arbre 2*)

- $V = "F"$, $S = "]", [$, $+, -"$,
- $\omega = "F"$,
- $P = "F" \longrightarrow F[+F]F[-F][F]"$.

Les variantes de ce modèle portent principalement sur l'application des règles (utilisation du contexte, de conditions et de probabilités). Les map-L-systèmes appliquent le principe des L-systèmes mais génèrent des graphes. Le processus débute par l'application des règles de transformation, suivies de la liaison des arêtes pendantes pour créer les faces. Les cells-L-systèmes [60] fonctionnent de même avec l'ajout de modifications de faces dans les règles.

Notons que cette approche de type L-système a pu être utilisée dans le cadre des g-cartes : les g-map-L-systèmes. Cette méthode a en particulier été employée pour modéliser la croissance interne du bois [74]. Ce modèle est comparable aux L-systèmes classiques mais s'applique à des volumes. La croissance est guidée par l'application séquentielle de règles choisies suivant le contexte local. Comme pour les L-systèmes, les transformations consistent essentiellement en subdivisions hiérarchiques d'un maillage par l'éclatement de faces/volumes, la création de faces/volumes et l'assemblage d'arêtes/volumes.

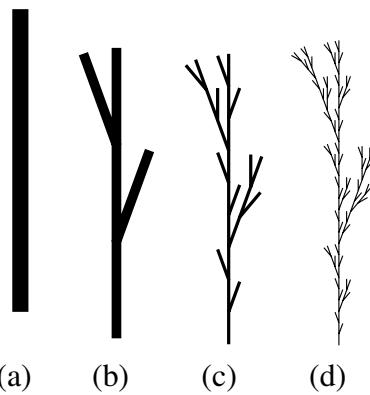


FIG. 2.20 – (a-d) Interprétation turtle du L-système 1 après zéro, une, deux et trois itérations.

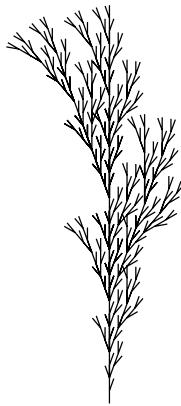


FIG. 2.21 – L-système 2 après quatre itérations, on observe des branches qui se croisent.

2.3.2 Les systèmes vertex-vertex

Les systèmes vertex-vertex [72] utilisent un modèle topologique 2D correspondant à un graphe non orienté, où les sommets représentent les sommets de la structure, et les liens représentent une permutation des arêtes autour des sommets. Cette structure est comparable aux cartes combinatoires de dimension 2. La subdivision 2D évolue par un ensemble de modifications appliquées au cours du temps. Le langage, une sur-couche du C++, est de plus bas niveau que les L-systèmes. Il permet d'accéder directement à la structure topologique, de la parcourir et de la modifier. Les modifications topologiques portent sur l'ordre des sommets autour du sommet courant, avec la possibilité d'en ajouter / supprimer. Des informations sont ajoutées au graphe, telles que le plongement géométrique, ou des marqueurs permettant aux fonctions de pouvoir simuler des règles d'évolutions.

Un exemple typique de ce modèle est la subdivision de maillage 2D. La fonction *polyhedral* est extraite de [72]. Elle correspond au plus simple schéma de subdivision d'un maillage triangulaire. L'algorithme 2.1 insère un sommet au milieu de chaque arête du triangle (lignes 5 à 13, figure 2.23(a)) et divise chaque triangle en quatre triangles coplanaires en modifiant les relations d'adjacence de chaque nouveau sommet (lignes 15 à 22, figures 2.23(b) à 2.23(c)).

Listing 2.1 – Subdivision polyédrique (algorithme extrait de [72])

```
void polyhedral(mesh & V) {
    mesh N ;
    synchronise V ; // Crée une copie de la structure V accessible par l'opérateur '
```

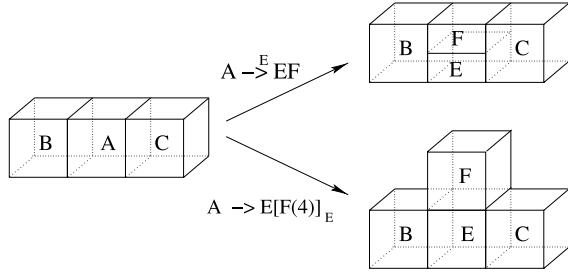


FIG. 2.22 – Exemple d’opérations sur les 3-G-Map-L-systèmes : transformation du volume A en volumes EF par ajout et par découpe.

```

5   // Insertion d'un nouveau sommet au centre de chaque paire de sommets adjacents
forall v in V {
    forall u in 'v { // pour tous les voisins du sommet 'v
        if (u < v) continue; // assure que chaque arête n'est traitée qu'une seule fois
        vertex x = insert(u, v); // insérer un sommet entre u et v
        x$pos = 0.5 * (u$pos + v$pos); // plongement
        add x to N; // N = N Union {x}
    }
}

15  // Ajout des relations de voisinage à chaque nouveau sommet
forall v in N {
    // Prendre les deux sommets adjacents au nouveau sommet
    vertex a = any in v; // prendre un sommet dans le voisinage de v
    vertex b = nextto a in v; // prendre le sommet suivant de a dans le voisinage de v
    20 // Assigner les relations de voisinage avec le nouveau sommet v
    make { nextto v in a, a, prevto v in a, nextto v in b, b, prevto v in b } nb_of v;
}
merge V with N ;
}

```

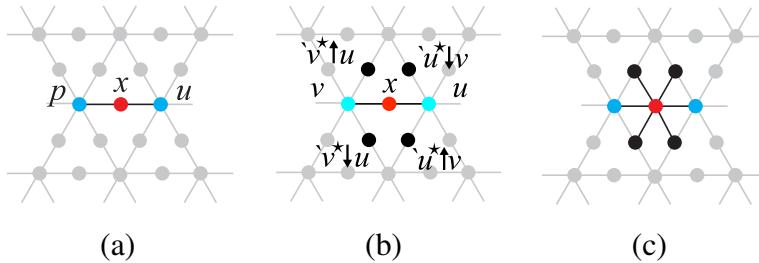


FIG. 2.23 – (a) Insertion d’un nouveau sommet x entre p et u. (b) Recherche des nouveaux sommets insérés dans les faces incidentes au sommet x. (c) Création des relations d’adjacence entre x et ces sommets (images extraites de [72]).

2.3.3 MGS

MGS [34] est un langage de programmation fonctionnelle pour la transformation de structures, essentiellement basé sur un système à base de règles de transformation. MGS décrit les transformations de chemins composés d’une suite de labels/types d’entités/répétitions de sous-chemins.

Le listing 5.4 correspond au même schéma de subdivision décrit ci-dessus. Les lignes 3 à 11 correspondent à l'insertion d'un sommet ' v ' sur chaque arête délimitée par $v1$ et $v2$ (cf. figure 2.24(a)). Cette insertion est réalisée par la règle *insert_vertex*, qui débute par la recherche d'un motif correspondant à deux sommets ($v1$ et $v2$) liés par une arête (e) et qui le transforme en trois sommets ($v1$, $v2$ et ' v ') liés par deux arêtes (' $e1$ et ' $e2$ '). Les lignes 13 à 25 correspondent à la recherche d'une face f constituée de six arêtes $e1$ à $e6$. f est transformée en quatre faces $f1$ à $f4$ dont les arêtes $e1$ à $e6$ et $e24$, $e46$, $e62$ constituent les bords.

Listing 2.2 – Polyhedral subdivision (algorithme extrait du projet MGS)

```

record Point = { x:float , y:float , z:float , gen:int } ;;

patch insert_vertex[genNb] = {
    ~v1 < e:[dim=1] > ~v2 =>
    5      'e1:[dim=1,faces=(v1,'v),`edge]
          'v:[dim=0,cofaces=(`e1,'e2),{gen = genNb,
          x   = (v1.x + v2.x) / 2,
          y   = (v1.y + v2.y) / 2,
          z   = (v1.z + v2.z) / 2}]
10     'e2:[dim=1,faces=(v2,'v),`edge]
} ;;

patch build_triangles[genNb] = {
    f:[dim=2,(e1,e2,e3,e4,e5,e6) in faces]
15   ~v1 < ~e1 > ~v2:[v2.gen == genNb] < ~e2 >
    ~v3 < ~e3 > ~v4:[v4.gen == genNb] < ~e4 >
    ~v5 < ~e5 > ~v6:[v6.gen == genNb] < ~e6 > ~v1 =>
    'e24:[dim=1,faces=(v2,v4),`edge]
    'e46:[dim=1,faces=(v4,v6),`edge]
20   'e62:[dim=1,faces=(v6,v2),`edge]
    'f1:[dim=2,faces=(e6,e1,'e62),`face]
    'f2:[dim=2,faces=(e2,e3,'e24),`face]
    'f3:[dim=2,faces=(e4,e5,'e46),`face]
    'f4:[dim=2,faces=(e24,e46,'e62),`face]
25 } ;;

```

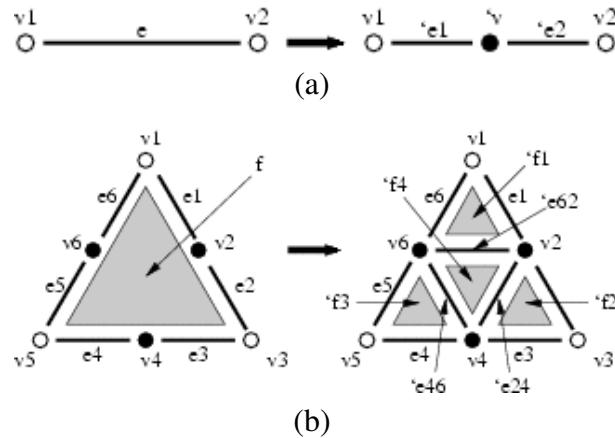


FIG. 2.24 – (a) Insertion des nouveaux sommets sur les arêtes de la face. (b) Création des arêtes et des faces ainsi que leurs relations d'adjacence et d'incidence (images extraites du projet MGS).

2.3.4 Discussion

Les structures présentées dans cette section permettent effectivement de faire évoluer la topologie d'objets au cours du temps (pour des applications spécifiques dans la plupart des cas). Cependant, en considérant nos objectifs, ces approches présentent des limites. Tout d'abord, elles n'ont pas réellement pour but de générer une animation mais peuvent plutôt être considérées comme un processus de modélisation comportant une succession d'opérations de construction. On parle de « modélisation procédurale ». Le résultat n'est donc pas une représentation continue de l'évolution du système, mais une succession d'étapes d'évolution. De plus, la cohérence topologie/géométrie n'est la plupart du temps pas assurée. Les règles de transformation ont seulement pour objectif de complexifier la structure et non de traiter les problèmes géométriques et topologiques. Or, nous souhaitons que les règles ne contrôlent pas seulement une évolution, mais qu'elles permettent au système de réagir aux collisions et aux changements topologiques de manière adaptée.

Un autre inconvénient est à souligner : ces systèmes offrent un contrôle haut-niveau assez limité par rapport au résultat obtenu. Pour obtenir un résultat donné ou même lorsqu'une incohérence survient (une collision par exemple), les règles doivent être modifiées de façon peu intuitive, ce qui requiert une expertise certaine. Il n'est ainsi pas garanti qu'un jeu de règles permette d'aboutir à des structures attendues et/ou cohérentes.

2.4 Conclusion

Dans ce chapitre, nous avons traité successivement les travaux antérieurs concernant les modèles topologiques, les modèles d'animation, puis les modèles topologiques dans le cadre de systèmes évolutifs ou animés. Nous avons montré que les systèmes proposés visaient essentiellement une description de la structure au cours du temps, mais ne garantissaient pas la cohérence géométrique globale.

Or, dans la section traitant des modèles topologiques, nous avons présenté le cas des partitions de l'espace, où tous les objets considérés sont englobés dans la même structure. Dans ce type d'approche, on suppose que chaque volume est associé à un seul objet, ce qui implique que deux objets ne sont pas autorisés à s'interpénétrer. Dans le cas d'objets animés, suivre l'évolution de la partition permet d'examiner les changements des voisinages entre objets et détecter plus facilement les intersections. Il semble ainsi opportun de nous focaliser sur des partitions de l'espace, tout d'abord parce qu'elles permettent de représenter les structures des objets réels dont nous désirons représenter l'évolution (en biologie, en géologie, etc.), mais aussi parce que ce type de structures nous offre un cadre idéal pour garantir la cohérence de la topologie et de la géométrie.

Par conséquent, notre objectif est de concevoir un système d'animation de structures topologiques, et en particulier de partitions, paramétrable et adaptable à des applications données, qui maintienne la cohérence topologique de la scène à chaque instant. Nous souhaitons également représenter l'historique de l'animation et y adjoindre des propriétés sémantiques relatives au contexte étudié.

Ce modèle d'animation est obligatoirement générateur, afin que l'animateur n'ait pas à traiter explicitement chaque modification de la structure. Les modèles topologiques présentés fournissent de nombreuses informations sur les relations de voisinage entre les entités topologiques

et il ne semble pas utile de concevoir un nouveau type de structure. Il nous faut plutôt pouvoir contrôler ces structures dans un système d'animation dédié.

MODÈLE TOPOLOGIQUE SPATIO-TEMPOREL

Dans ce chapitre, nous définissons un modèle d'animation d'objets basé sur la topologie. En d'autres termes, il s'agit ici de doter les modèles topologiques d'une dimension temporelle supplémentaire. Ceci doit cependant se faire en autorisant un contrôle de cette animation, c'est-à-dire en concevant des opérations adaptées, sur le plan spatial et temporel. Pour cela, nous étudions d'abord deux pistes différentes pour concevoir le modèle. La première, dite « continue », considère le temps comme une dimension similaire aux dimensions spatiales, d'où son appellation $(n+1)D$. L'autre, dite « discrète », considère la nature instantanée des modifications topologiques et fonde l'aspect temporel sur une succession de structures topologiques définies pour chacune par rapport à la structure précédente. Dans la suite du chapitre, nous étudions, dans le cas particulier 2D, les opérations nécessaires à la manipulation de ces modèles afin, d'une part, de définir l'ensemble des primitives de transformation nécessaires à la création des animations et, d'autre part, de fixer notre choix sur l'un des modèles proposés.

3.1 Structure topologique temporelle

Pour concevoir un modèle structurel capable de représenter l'évolution d'un système, il nous faut doter un modèle topologique d'une dimension temporelle supplémentaire. Se contenter d'ajouter un aspect temporel au plongement n'est pas suffisant. En effet, la structure n'étant pas fixe, elle se doit d'être définie, elle-même, au cours du temps. Nous avons donc besoin d'une structure spatio-temporelle. Nous avons le choix entre les deux possibilités de représentation d'animation présentées au paragraphe 2.2.3, mais nous les considérons ici sous un angle purement topologique. Soit nous choisissons de représenter le temps de façon continue, comme les autres dimensions spatiales, et obtenons ainsi un modèle $(n+1)D$; soit nous considérons le temps de façon discrète, i.e., nous nous intéressons exclusivement aux *instants* où surviennent des modifications topologiques de la structure. La différence fondamentale entre ces deux domaines revient à savoir si le temps n'est pris en compte qu'au niveau des plongements (cas

$nD + temps$ où la structure topologique reste nD), ou au contraire si la structure topologique est intrinsèquement spatio-temporelle.

3.1.1 Approche $(n + 1)D$

Nous avons décrit dans la section 2.2.3 le principe de l'approche $(n + 1)D$, dans laquelle les animations sont représentées sous forme d'objets spatio-temporels et les étapes de l'animation sont obtenues par la « coupe temporelle » de ces objets. Nous nous intéressons à cette approche pour le suivi topologique de l'évolution d'une partition : nous considérons ainsi une structure $(n + 1)D$ (impliquant en particulier des entités topologiques de dimension $n + 1$).

Nous avons défini dans [40] une méthode d'animation $(2 + 1)D$ reposant sur cette approche que nous décrivons ici en dimension $(n + 1)D$. Cette méthode se base sur des modifications locales d'un volume spatio-temporel résultant d'une extrusion temporelle. Le principe est le suivant. Nous commençons par construire une scène spatiale nD (le plongement temporel de la scène est égal à t_{min}). Cette scène est extrudée suivant l'axe temporel sur un intervalle $[t_{min}, t_{max}]$. À ce stade, les volumes spatio-temporels résultant de l'extrusion représentent des objets nD immobiles au cours de l'intervalle de temps. Des modifications locales sont ainsi appliquées sur les sommets de la scène dont le plongement temporel vaut t_{max} . Ces sommets étant reliés à leurs homologues de plongement temporel t_{min} , la coupe temporelle des volumes spatio-temporels permet d'obtenir les états intermédiaires de l'animation entre t_{min} et t_{max} .

Si l'objet doit subir des modifications topologiques, nous appliquons des transformations, issues d'opérations définies en nD et réinterprétées en $(n + 1)D$. Par exemple, pour éclater une arête en deux arêtes (insertion d'une face) dans l'objet initial, nous éclatons la face correspondant à l'évolution de l'arête dans le modèle spatio-temporel et faisons apparaître un nouveau volume. L'objet $(n+1)D$ est donc modifié, mais également ses coupes temporelles, donc l'objet nD dont on représente l'évolution.

Le processus est réitéré : extrusion de la face supérieure (le long de l'axe temporel), modification des plongements géométriques et/ou application des transformations ayant lieu entre les deux instants de l'extrusion. L'utilisateur indique, de façon interactive, la succession d'opérations topologiques que la structure doit subir. Cette démarche implique que l'objet $(n + 1)D$ est défini par intervalles temporels, chaque intervalle correspondant aux extrusions successives. Précisons que l'opération d'extrusion a pour principal objectif d'obtenir rapidement un objet $(n + 1)D$ à partir de la scène nD , en affectant à toutes les cellules de la scène nD une évolution par défaut (les cellules restent immobiles) sur l'intervalle de temps défini lors de l'extrusion. On modifie ensuite le plongement de certaines cellules définies à t_{max} pour modifier l'évolution par défaut.

La figure 3.1 illustre un exemple d'objet spatio-temporel $(2 + 1)D$ modélisé avec les opérations décrites dans [40] et au paragraphe 3.2.1. L'objet représente l'évolution d'une subdivision $2D$ composée au départ de deux rectangles accolés. Une première, puis une seconde face, s'insèrent entre ces deux entités. L'insertion de ces faces est réalisée par l'opération d'éclatement d'un *chemin d'arêtes* (une succession d'arêtes consécutives dans la subdivision). L'éclatement d'un chemin d'arêtes consiste à insérer un volume spatio-temporel dont une extrémité spatiale est une ligne polygonale représentant le chemin d'arêtes, tandis que l'autre extrémité est une face représentant la face à insérer : l'animation décrite par le volume représente donc le chemin d'arêtes qui se dilate en face.

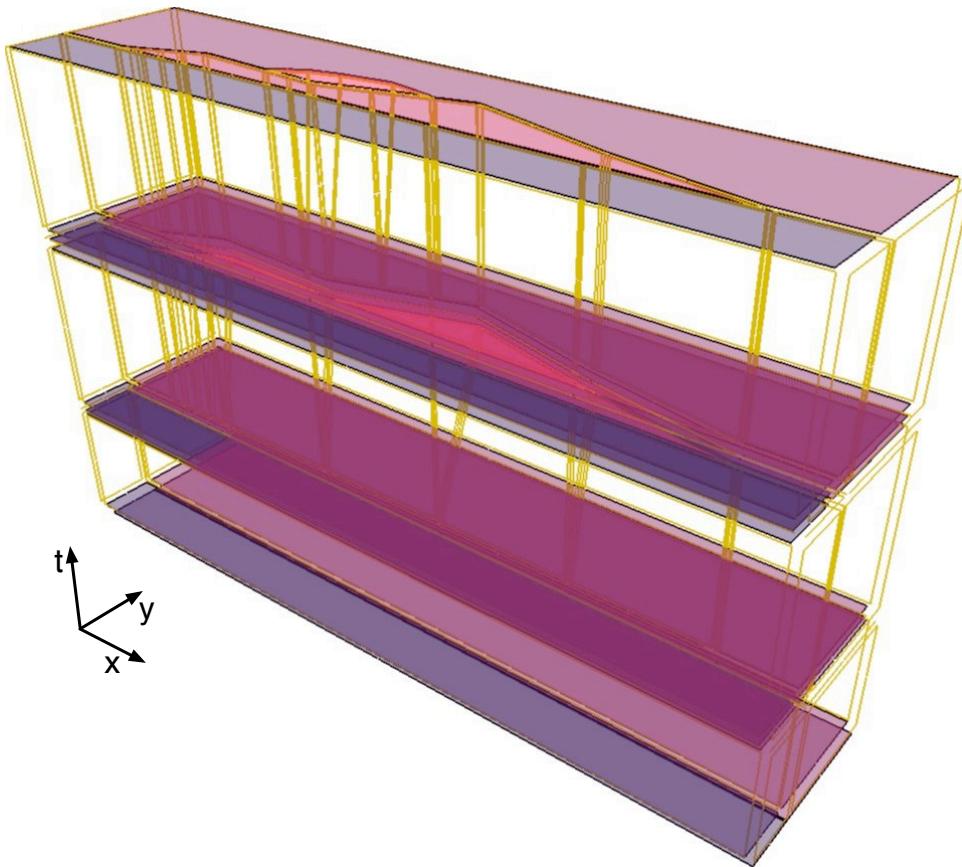


FIG. 3.1 – Volume spatio-temporel représentant deux rectangles accolés subissant deux insertions successives de faces au cours du temps. Les faces latérales du volume sont transparentes pour faciliter la compréhension de l’animation. Le début de l’animation (côté inférieur du volume) montre les deux rectangles initiaux. La dernière étape (côté supérieur du volume) affiche les deux nouvelles faces insérées entre celles du départ. L’insertion des nouvelles faces est réalisée par l’éclatement de chemins d’arêtes. Les deux étapes intermédiaires affichées dans le volume spatio-temporel illustrent l’évolution de la subdivision.

Notons qu’en pratique, nous avons utilisé un plongement linéaire, aussi bien pour la partie spatiale que temporelle, ce qui oblige également à ajouter des blocs à chaque instant comportant des modifications de plongement.

En résumé, notre modèle peut être vu comme une suite de cartes nD purement spatiales (orthogonales à l’axe du temps), englobées dans une carte $(n+1)D$ faisant les liens temporels entre ces cartes. Notre modèle nous permet de manipuler trois types de cellules contenus dans les volumes (pour les cellules de dimensions 1 à n) : les cellules « purement » spatiales (sommets, arêtes, faces perpendiculaires à l’axe temporel) ; les cellules « purement temporelles » (des cellules de dimension i résultant respectivement de l’extrusion des cellules (immobiles) de dimension $i-1$ spatiales, le long de l’axe temporel) ; et, naturellement, les cellules spatio-temporelles, résultant de la modification de plongement d’une extrémité de cellules temporelles.

Cependant, ce modèle présente une limite importante. Nous utilisons des transformations topologiques définies sur une extrusion, ce qui permet de borner dans le temps, l’application de cette transformation. D’une part, avant t_{min} , la transformation n’a pas lieu. D’autre part, la date de fin de transformation n’est pas forcément définie (par exemple, une arête éclatée reste éclatée définitivement, sauf si une nouvelle transformation vient modifier la face créée lors de

l'éclatement). Dans le cas où la limite temporelle supérieure n'est pas fixée, nous choisissons comme t_{max} , non pas la fin du phénomène, mais le début du phénomène suivant (ou la date d'un changement de plongement temporel lorsque la contrainte de linéarité l'oblige). L'évolution du phénomène peut cependant continuer au delà de t_{max} , il suffit alors de poursuivre les modifications des plongements. Ceci aboutit à une remarque importante : la modification topologique n'a réellement lieu qu'à l'instant t_{min} (dans notre exemple, le moment où l'arête s'éclate). Lorsque la transformation est traduite en $(n + 1)D$ et effectuée sur l'extrusion, elle vise à seulement indiquer qu'il y a eu une transformation topologique initiale, qui perdure dans le temps.

De ce fait, le problème revient surtout à déterminer quand ont lieu les transformations topologiques : leur commencement et leur fin. Si le commencement est généralement piloté par l'application, la fin est tributaire de nombreux paramètres (par exemple, en $2D$, lorsqu'une face qui est incisée progressivement, jusqu'à se couper définitivement en deux, la date de fin correspond au moment où l'incision atteint un bord de la face). Il est donc naturel de se focaliser sur ces instants.

En outre, l'interprétation en $(n + 1)D$ d'une transformation nD peut parfois être assez délicate à définir (par exemple, l'interprétation $(2 + 1)D$ d'une face qui se coupe en deux, voir figure 3.2). Or, il est légitime de se poser la question de la pertinence de la définition de l'opération $(n + 1)D$, souvent complexe, là où les transformations topologiques se définissent beaucoup plus simplement en nD .

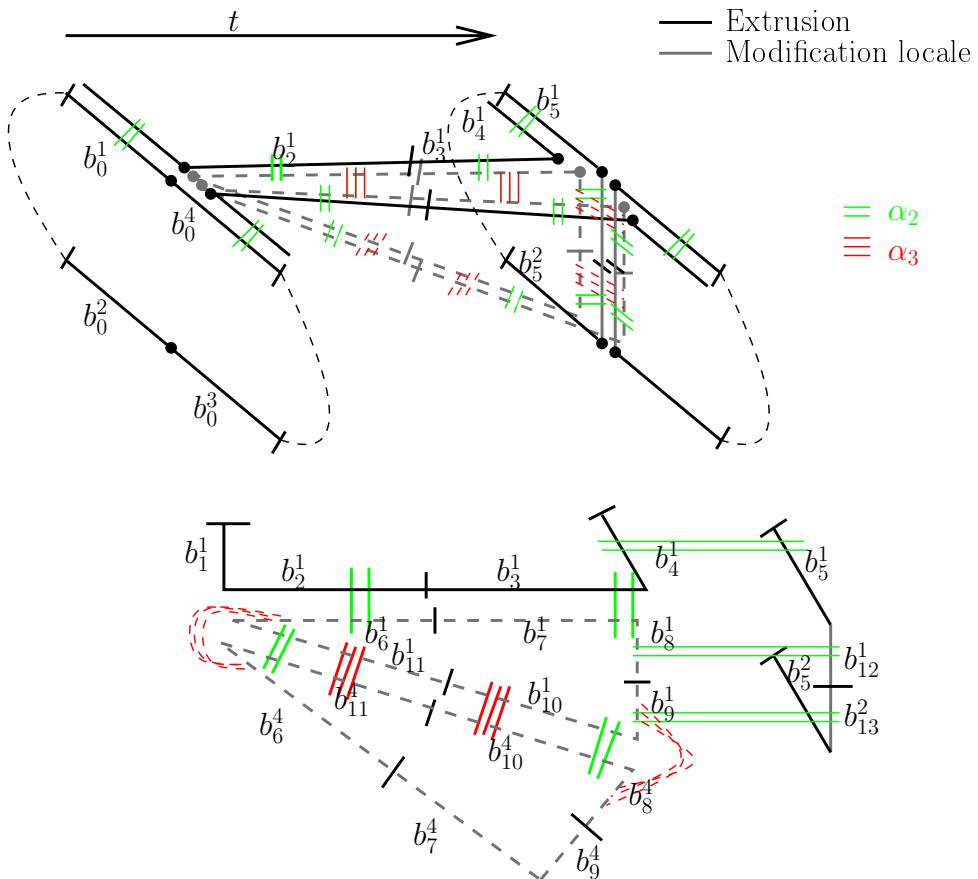


FIG. 3.2 – Opération de scission d'une face.

Aussi, puisque la dimension $n + 1$ n'a pour rôle que celui de représenter l'évolution du plongement géométrique et qu'elle est souvent délicate à construire, il nous semble opportun d'examiner un autre modèle, qui se focalise sur les instants de modifications topologiques seulement. Nous l'examinons dans le paragraphe suivant.

3.1.2 Modèle par images clefs

La modélisation par images-clefs ou *keyframes* consiste à créer une séquence d'images représentatives (dites clefs) de l'animation [15]. L'animation est obtenue grâce à un algorithme qui construit les images intermédiaires par interpolation entre deux images-clefs successives. Cet algorithme doit effectuer une correspondance entre les éléments de l'image courante et ceux de la suivante. Si l'algorithme n'est pas en mesure d'apparier tous les éléments, il insère généralement des entités fictives. Les éléments étant assortis, les positions et directions des objets peuvent être interpolées par diverses méthodes (linéaire, lagrangienne, spline, etc.) [14] qui fournissent des résultats plus ou moins naturels (cf. figure 3.3).

Pour adapter cette technique à notre problème d'animation topologique et pour limiter la taille de la structure, nous avons fait le choix de positionner des images-clefs aux seuls instants où se produisent les changements topologiques. En effet, une modification topologique apparaît comme une discontinuité dans l'animation et elle est nécessairement instantanée : on peut lui associer une date et créer une image-clef à cette date. Par cette méthode, une image-clef regroupe l'ensemble des modifications topologiques simultanées et se définit par rapport à l'image-clef précédente. Le mouvement des entités se traite par interpolation ou tout autre méthode adéquate et cette évolution est portée uniquement par le plongement, sans recours à des images-clefs. La géométrie des entités est ainsi définie par une fonction f dans \mathcal{R}^n . En résumé, la structure globale est une succession d'images-clefs (cf. figure 3.4). Elle correspond à une n -g-carte fermée et connexe. Entre deux n -g-cartes, aucune modification topologique n'intervient. Extraire des images de l'animation est assez simple : on parcourt l'image-clef antérieure la plus proche de l'instant t considéré et on extrait le plongement temporel des entités topologiques à cette date (cf. figure 3.4).

Le grand avantage de cette technique est que les opérations de manipulation du modèle sont celles de la dimension de l'objet initial et sont donc généralement bien maîtrisées. En d'autres termes, à partir d'un résultat souhaité, il est plus facile de déterminer les paramètres à utiliser pour l'obtenir. Le contrôle de l'animation en est donc facilité.

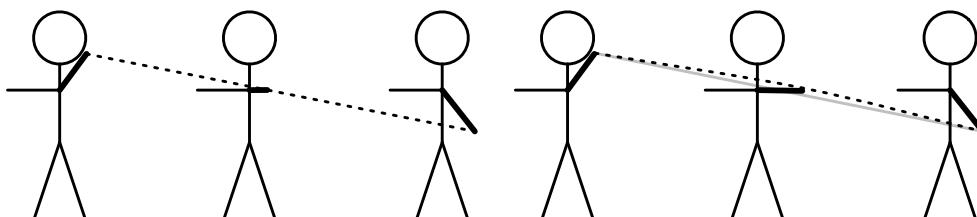


FIG. 3.3 – L'interpolation linéaire ne donne pas des résultats naturels. (a) Avec une interpolation linéaire (ligne en pointillés), le bras du personnage ne conserve pas sa taille. (b) Avec une interpolation non linéaire, le bras conserve sa taille, l'animation est plus naturelle.

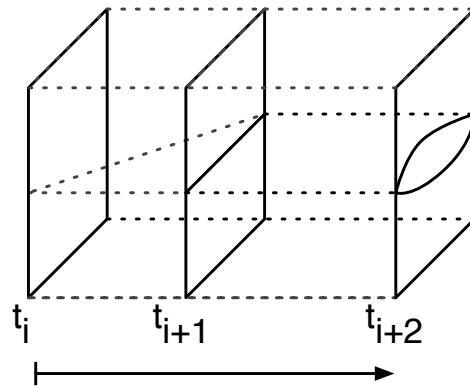


FIG. 3.4 – Modèle par images-clefs : chaque image (en trait plein) correspond à un ensemble de modifications topologiques : à l'instant t_i , un sommet sur le côté de la face est éclaté en segment de longueur nulle. Entre t_i et t_{i+1} , le segment s'allonge (seul le plongement est modifié). À t_{i+1} , le segment touche l'autre côté de la face, la coupe en deux et éclate en face d'aire nulle. Le plongement de cette nouvelle face évolue ensuite pour adopter la forme d'une face elliptique jusqu'à t_{i+2} , date de la modification topologique suivante dans l'animation.

3.1.3 Discussion

Comme nous venons de le voir, la représentation des informations structurelles peut être décrite soit par un modèle $(n + 1)D$, soit par un modèle à base d'*images-clefs*. Au cours de nos travaux, nous avons étudié et implanté ces deux approches [40]. L'approche basée *images-clefs* permettant de représenter la même information que l'approche $(n + 1)D$, les deux modèles sont équivalents. En effet, on peut construire un modèle à base d'*images-clefs* à partir d'un modèle espace-temps en utilisant des coupes temporelles sur ce dernier. Tant que la topologie des différentes intersections successives d'un hyperplan avec la structure n'est pas modifiée, nous nous trouvons entre deux images-clefs où seuls les plongements géométriques changent en fonction de t . Inversement, le passage du modèle basé *images-clefs* vers un modèle spatio-temporel s'effectue d'abord par extrusion des n -g-cartes le long de l'axe temporel, puis par l'union des hyper-volumes ainsi obtenus et disposés l'un après l'autre le long de l'axe temporel (cf. figure 3.5).

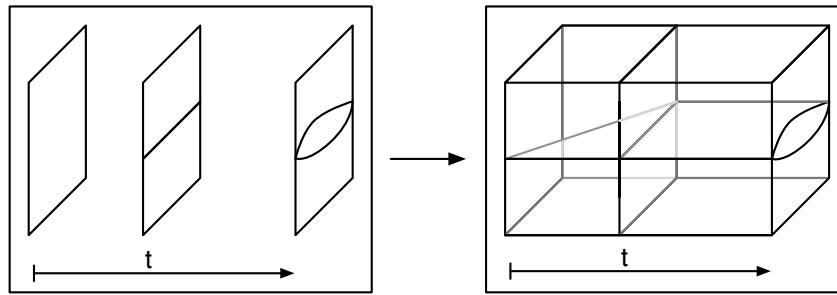


FIG. 3.5 – Passage du modèle temporel à un modèle spatio-temporel par l'union des extrusions des 2-g-cartes suivant le plongement des sommets.

Bien que ces modèles soient équivalents, l'approche $(n + 1)D$ nécessite de disposer en permanence d'un objet continu, ce qui peut être considéré comme un historique de l'évolution de l'objet au cours du temps. Ce fonctionnement implique de penser les opérations topologiques en conséquence car il faut prendre en compte la continuité temporelle. En effet, les opérations

topologiques se traduisent par deux types de modifications : spatiales et temporelles. L'information de continuité portée par le modèle $(2+1)D$ avec les 3-g-cartes est redondante et coûteuse en termes de mémoire (une arête est un ensemble de n brins avec $n > 4$). Le modèle étant continu, chaque opération se décrit à la fois dans le plan « spatial », mais également en termes de liens « historiques » le long de l'axe temporel. Cette gestion des voisinages spatiaux et historiques alourdit considérablement la définition des opérations car il y a un nombre plus important de relations de voisinage à gérer (cf. figure 3.6).

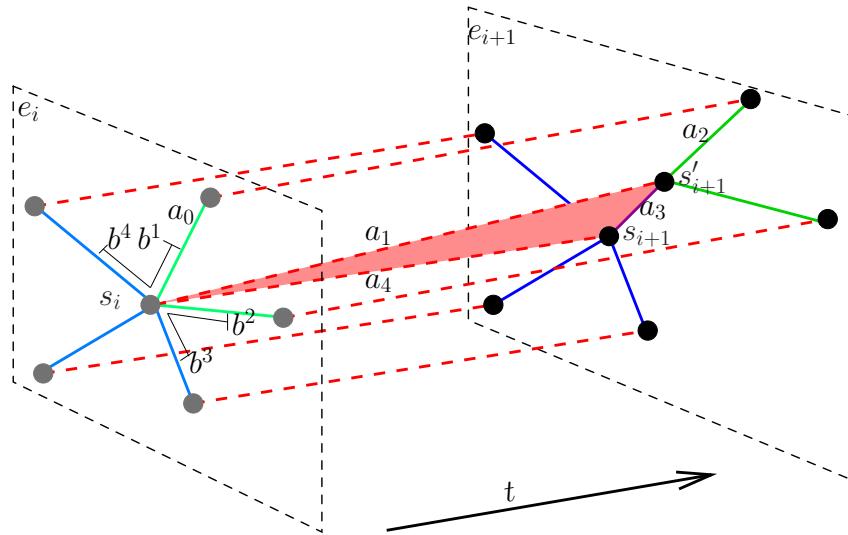


FIG. 3.6 – Exemple de l'opération de transformation d'un sommet en arête spatio-temporelle (Traits plein : composante spatiale ; pointillés : composante temporelle/historique).

En résumé, nous avons choisi dans ce contexte l'approche basée *images-clefs* pour représenter l'historique des modifications topologiques.

3.2 Étude du cas particulier 2D

Cette section se focalise sur l'étude de notre modèle dans le cas particulier de la 2D. Le but est ici de représenter l'évolution d'une subdivision de l'espace 2D. L'évolution peut être décrite de plusieurs manières, par exemple, de manière interactive à l'aide d'une interface graphique ou à l'aide d'une description textuelle. Nous avons choisi d'utiliser un script pour maîtriser tous les paramètres nécessaires à la réalisation de l'animation. Ce script décrit les étapes de construction de chaque image-clef ainsi que le plongement géométrique associé à chaque entité. Les entités manipulées par le script sont identifiées par un nom. Les étapes de construction sont une suite d'opérations topologiques appliquées à des entités désignées.

3.2.1 Opérations topologiques 2D

Dans le cadre de l'animation d'une subdivision 2D, nous utilisons huit opérations topologiques principales : l'éclatement et l'insertion de sommet, l'identification de deux sommets, la création, la déconnexion, la contraction et la suppression d'arête et l'éclatement d'un chemin d'arêtes. La contraction et la suppression d'arête sont définies de manière générale en dimension n par Damiand et al. [20]. Nous considérons que ces opérations permettent de générer toutes les

animations possibles en 2D. En effet, nous pouvons comparer une subdivision 2d à un graphe. Les opérations nécessaires pour construire un graphe sont l'ajout/suppression de sommets et le changement des liaisons entre ces sommets.

3.2.1.1 Éclatement d'un sommet et création d'arête

L'opération d'éclatement de sommet est une manière de créer une arête. Elle sépare un ensemble d'arêtes incidentes à un sommet en deux ensembles distincts d'arêtes (cf. figure 3.7(a)). Nous distinguons deux arêtes dans l'ensemble, appelées a_1 et a_2 . Ces arêtes délimitent un des sous-ensembles (représenté en noir), tandis que l'autre sous-ensemble contient toutes les autres arêtes (affichées en gris). L'éclatement de sommet produit deux sommets respectivement incidents à chaque sous-ensemble (cf. figure 3.7(b)). Notons que cette opération peut également être écrite à l'aide des opérations de création d'arêtes, d'identification de sommets et de déconnection d'arête.

Définition 9 Soit $G = (B, \alpha_0, \alpha_1, \alpha_2)$ une 2-g-carte fermée. Soient b_1^1, b_1^2, b_1^3 et b_1^4 des brins de B tels que :

- $\exists p > 0, b_1^1 = b_1^2(\alpha_2\alpha_1)^p\alpha_2$;
- $b_1^3 = b_1^2\alpha_1, b_1^4 = b_1^1\alpha_1$.

En dimension 2, un éclatement de sommet en arête engendre une 2-g-carte $G' = (B', \alpha'_0, \alpha'_1, \alpha'_2)$ telle que :

- $B' = B \cup \{b_2^1, b_2^2, b_2^3, b_2^4\}$;
- $\forall b \in B, b\alpha'_0 = b\alpha_0, b\alpha'_2 = b\alpha_2$;
- $\forall b \in B - \{b_1^1, b_1^2, b_1^3, b_1^4\}, b\alpha'_1 = b\alpha_1$;
- $b_2^1\alpha'_0 = b_2^4, b_2^2\alpha'_0 = b_2^3, b_2^1\alpha'_2 = b_2^2, b_2^3\alpha'_2 = b_2^4$;
- $\forall i \in 1, \dots, 4, b_2^i\alpha'_1 = b_1^i$.

Dans la définition 9a, b_1^1 et b_1^2 sont les brins bordant un sous-ensemble d'arêtes, b_2^1, b_2^2, b_2^3 et b_2^4 sont les brins créés par l'opération d'éclatement de sommet.

3.2.1.2 Éclatement d'un chemin d'arêtes et création de face

L'éclatement d'un chemin d'arêtes crée une nouvelle face insérée entre les faces incidentes au chemin (cf. figure 3.8). Cette opération correspond à l'ajout d'une face. Un chemin d'arêtes est une suite d'arêtes sélectionnées le long du bord d'une ou plusieurs faces. Cette opération peut aussi être décrite à l'aide des opérations de création d'arêtes et d'identification de sommets.

Définition 10 En dimension 2, un chemin d'arêtes \mathcal{C} est une suite de brins $(b^0, b^1, \dots, b^{2k+1})$ telle que :

- $b^{2p+1} = b^{2p}\alpha_0$;
- $b^{2q} = b^{2q-1}(\alpha_1\alpha_2)^m\alpha_1$ avec $1 \leq q \leq k$ et $m \geq 0$.

Définition 11 Soit $G = (B, \alpha_0, \alpha_1, \alpha_2)$ une 2-g-carte fermée. Soient $\mathcal{C} = \{b_1^0, \dots, b_1^{2k+1}\}$ un chemin d'arêtes et $\mathcal{C}' = \{b_1^{2k+2}, \dots, b_1^{4k+3}\}$ son image par α_2 tels que :

- $\mathcal{C} \subset B, \mathcal{C}' \subset B$ et $\mathcal{C} \cap \mathcal{C}' = \emptyset$;
- $\forall i \in [2k+2 \dots 4k+3], b_1^i = b_1^{i-(2k+2)}\alpha_2$.

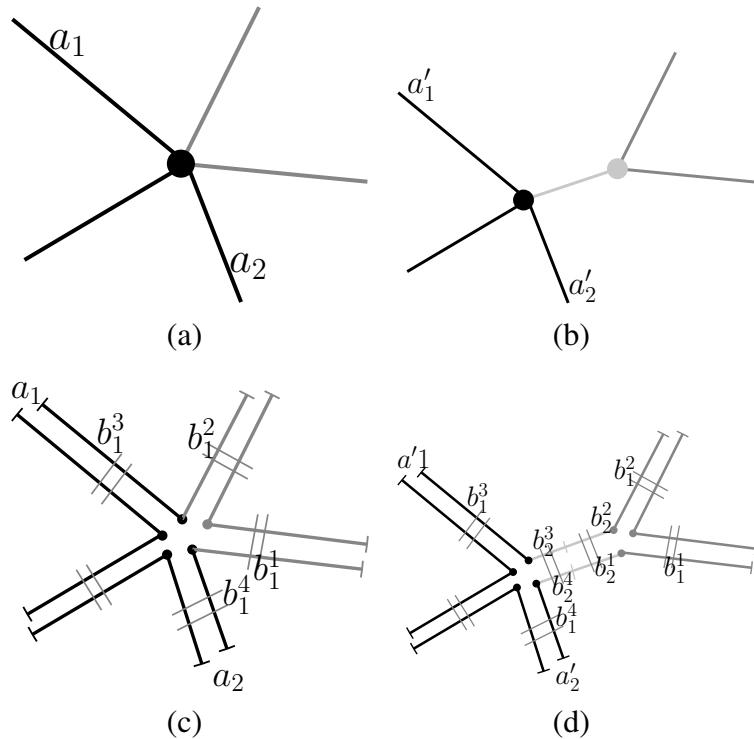
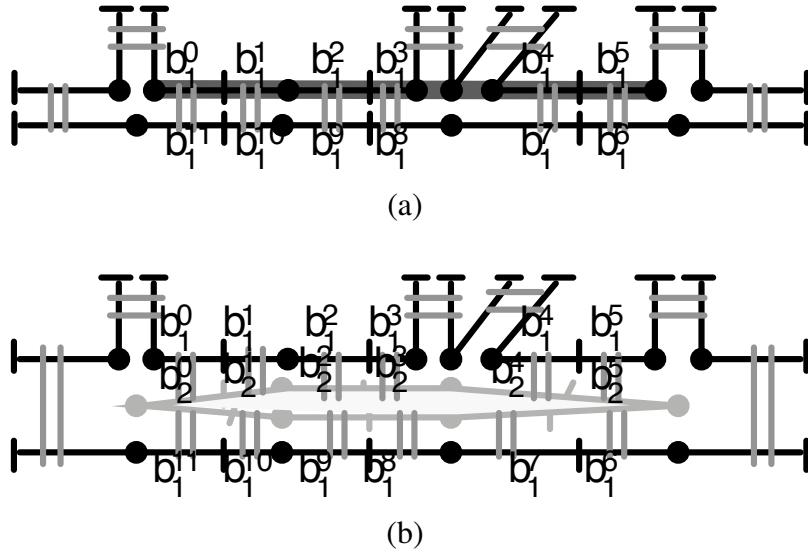


FIG. 3.7 – (a,c) Avant l'éclatement du sommet. (b,d) Après l'éclatement du sommet.

FIG. 3.8 – Un chemin d'arêtes défini entre les brins b_1^0 et b_1^5 est éclaté pour engendrer la face grise.

En dimension 2, l'éclatement d'un chemin d'arêtes engendre une 2-g-carte $G' = (B', \alpha'_0, \alpha'_1, \alpha'_2)$ telle que :

- $B' = B \cup \{b_2^0, \dots, b_2^{4k+3}\}$;
- $\forall b \in B, b\alpha'_0 = b\alpha_0$ and $b\alpha'_1 = b\alpha_1$;
- $\forall b \in B - (\mathcal{C} \cup \mathcal{C}'), b\alpha'_2 = b\alpha_2$;
- $b_2^{2a}\alpha'_0 = b_2^{2a+1}$ avec $0 \leq a < 2k + 1$;
- $b_2^i\alpha'_2 = b_1^i$ avec $0 \leq i \leq 4k + 3$;

$$- b_2^{2a+1} \alpha'_1 = b_2^{(2a+2) \text{mod}(4k+4)} \text{ avec } 0 \leq a \leq 2k+1.$$

Dans la définition 11a, les brins b_2^0, \dots, b_{4k+3}^4 représentent la face créée.

3.2.1.3 Identification de sommets

Intuitivement, l'identification de sommets correspond à leur fusion : les arêtes incidentes aux deux sommets sont par la suite incidentes à un seul et même sommet. Dans le formalisme des 2-g-cartes, ceci correspond à la modification des liaisons α_1 . Cette opération est décrite dans la définition 12 et implantée grâce à l'algorithme 1.

Définition 12 Soient $G = (B, \alpha_0, \alpha_1, \alpha_2)$ une 2-g-carte fermée et $D \subset B$ un ensemble de brins représentant une ligne polygonale, soient $b_3, b_4 \in D$ et $b_1, b_2 \in B - D$ tels que :

- $\forall b \in D, b\alpha_2 \neq b \wedge b\alpha_0 \neq b \wedge (b(\alpha_1\alpha_2)^2 = b \vee b\alpha_1\alpha_2 = b \vee b\alpha_1 = b);$ (a)
- $b_1\alpha_1 = b_2 \wedge b_3\alpha_2 = b_4 \wedge (b_3\alpha_1 = b_3 \vee b_3\alpha_1\alpha_2 = b_3).$ (b)

En dimension 2, l'identification de d_1 et d_3 engendre une 2-g-carte $G' = (B, \alpha_0, \alpha'_1, \alpha_2)$ telle que :

- $\forall b \in B - \{b_1, b_2, b_3, b_4\}, b\alpha'_1 = b\alpha_1;$
- $b_1\alpha'_1 = b_3, b_2\alpha'_1 = b_4.$

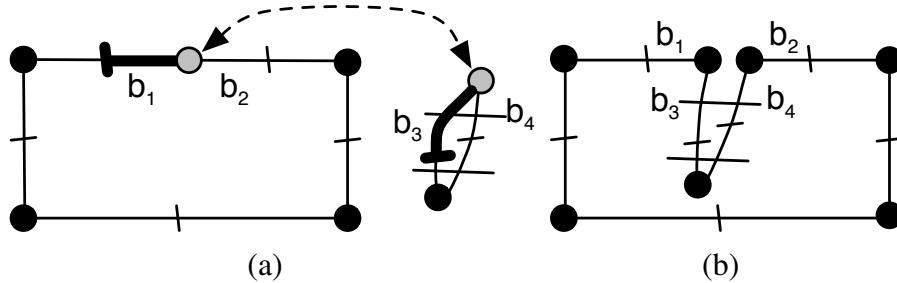


FIG. 3.9 – (a) Identification des deux sommets gris. (b) Pour identifier deux sommets, il suffit de modifier leurs liaisons α_1 .

Dans la définition 12, l'équation (a) définit la ligne polygonale. Dans l'équation (b), b_3 et b_4 représentent une des extrémités de la ligne polygonale. La figure 3.10(d) montre que l'opération d'identification peut aboutir à un objet non orientable lors de l'identification de la dernière extrémité de la ligne polygonale. Pour obtenir un objet orientable, il suffit que $b_3 \in < \beta_1 > (b_1\alpha_0)$ avec $\beta_1 = \alpha_0\alpha_1$.

Entrées : $b_1, b_2 \in G_2$

$$ba_1 \leftarrow b_1\alpha_1; ba_2 \leftarrow b_2\alpha_1$$

$$b_1\alpha_1 = b_2$$

$$ba_1\alpha_1 = ba_2$$

Algorithme 1 : Identification de deux sommets (2D)

Dans notre modèle, nous utilisons une marque d'orientation qui permet d'identifier de manière automatique une ligne polygonale dont une extrémité est libre (i.e. non liée) et dont l'autre appartient à la face contenant le sommet auquel on veut la relier (cf. figure 3.11). Dans cette configuration, deux cas peuvent se présenter : soit le sommet de la face est incident à une arête pendante (i.e. arête dont une extrémité n'est liée à aucune autre) contenue dans la face, soit

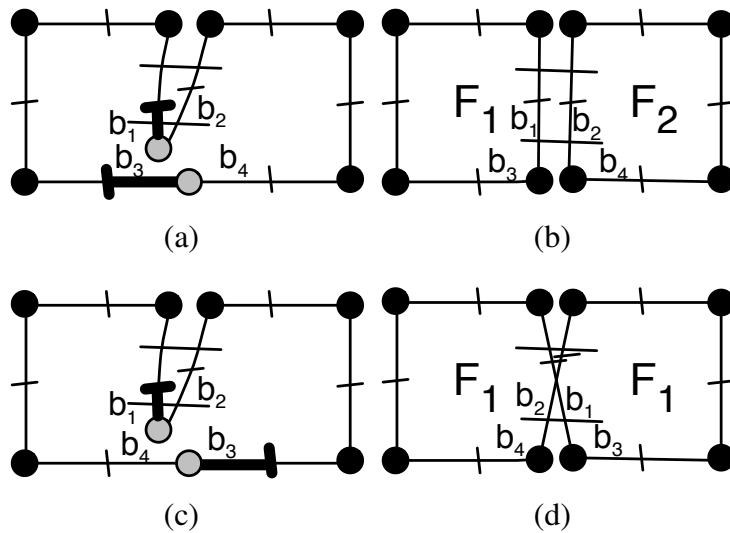


FIG. 3.10 – Deux solutions pour identifier deux sommets gris : (a-b) Solution orientable : la face est divisée, (c-d) Solution non orientable : la face devient non orientable et n'est plus plongeable en 2D.

c'est une configuration que nous qualifions d'idéale. Dans le premier cas, pour connaître les brins à identifier (cf. algorithme 2) (cf. figure 3.11(a)), on doit classer les arêtes autour du sommet. Cette méthode fait appel à la géométrie et peut donc introduire des erreurs causées par les arrondis. Dans le second cas, on réalise un simple parcours des sommets pour rechercher la face commune et on utilise la marque d'orientation (cf. algorithme 3) (cf. figure 3.12). Cette méthode fait appel à la topologie, elle est donc insensible aux problèmes géométriques.

Entrées : $b_1, b_2 \in G_2$

Sorties : bool

```

pour tous les  $bc_1 \in \langle 1, 2 \rangle(b_1)$  faire
  pour tous les  $bc_2 \in \langle 1, 2 \rangle(b_2)$  faire
    si  $bc_1 \in \langle 0, 1 \rangle(bc_2)$  alors
      classer_arêtes_autour_du_sommet()
      identifier les arêtes suivant l'ordre du classement
    retourner true
  retourner false

```

Algorithme 2 : Identification de deux sommets appartenant à la même face en respectant l'orientation dans le cas où les sommets appartiennent à une arête pendante (méthode géométrique 2D)

3.2.1.4 Crédation d'une arête libre, d'une face et de sommets

En deux dimensions, une arête est représentée par quatre brins liés par α_0, α_1 et α_2 (cf. figure 3.13), afin d'obtenir un modèle fermé (cf. définition 13).

Définition 13 Soient $G = (B, \alpha_0, \alpha_1, \alpha_2)$ une 2-g-carte fermée et $d \in B$. En dimension 2, la création d'une arête engendre une 2-g-carte $G' = (B \cup \{b_1, b_2, b_3, b_4\}, \alpha'_0, \alpha'_1, \alpha'_2)$ telle que :

- $\forall b \in B - \{b_1, b_2, b_3, b_4\}$, $ba'_0 = ba_0$, $ba'_1 = ba_1$, $ba'_2 = ba_2$

Entrées : $b_1, b_2 \in G_2$

Sorties : bool

pour tous les $bc_1 \in \langle 1, 2 \rangle(b_1)$ **faire**

pour tous les $bc_2 \in \langle 1, 2 \rangle(b_2)$ **faire**

si $bc_1 \in \langle 0, 1 \rangle(bc_2)$ **alors**

si pas_de_marque_orientation(bc₁) alors $bc_1 = bc_1\alpha_1$

si orientee(bc₂) alors $bc_2 = bc_2\alpha_1$

identifier(bc₁, bc₂)

retourner true

retourner false

Algorithme 3 : Identification de deux sommets appartenant à la même face en respectant l'orientation dans le cas où les sommets n'appartiennent pas à une arête pendante (méthode topologique 2D)

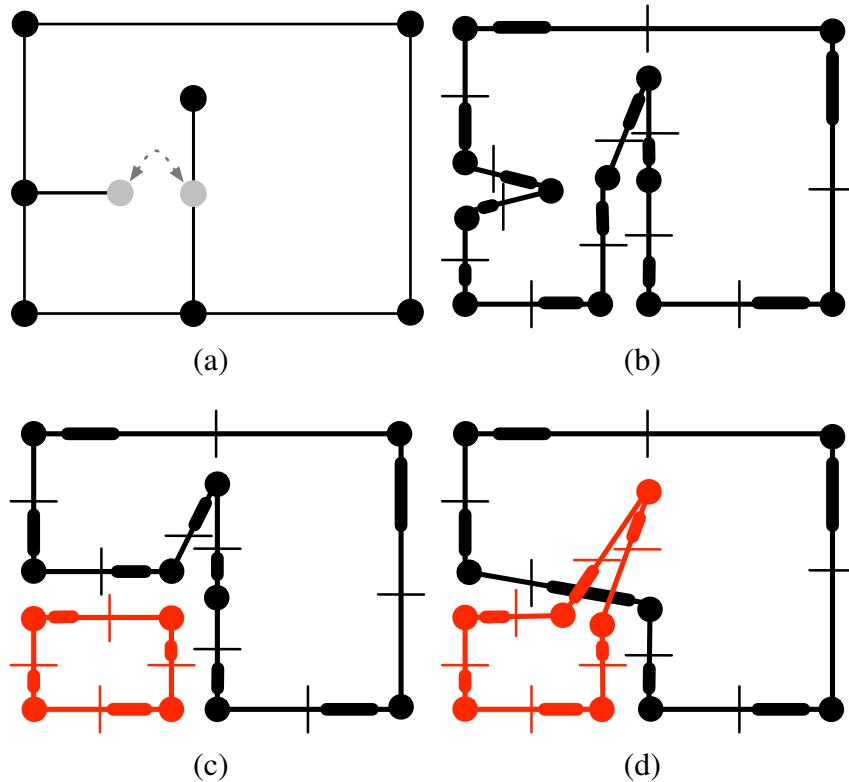


FIG. 3.11 – (a) Identification des deux sommets gris. (b) Vue éclatée. (c) Résultat voulu de l'opération. (d) Résultat non souhaité car il ne respecte pas l'ordre des arêtes autour du sommet résultant de la fusion (les brins en gras représentent des marques d'orientation).

- $b_0\alpha'_0 = b_1, b_2\alpha'_0 = b_3$
- $b_0\alpha'_1 = b_2, b_1\alpha'_1 = b_3$
- $b_0\alpha'_2 = b_2, b_1\alpha'_2 = b_3$

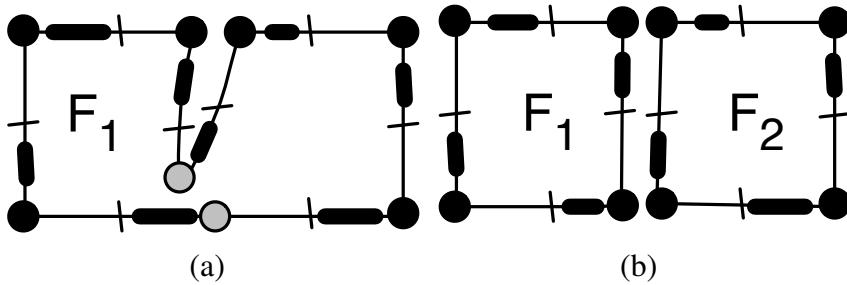


FIG. 3.12 – (a-b) Identification automatique des deux sommets gris en utilisant la marque d'orientation (les brins en gras représentent des marques d'orientation).

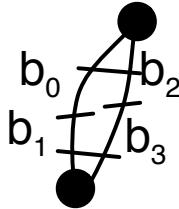


FIG. 3.13 – Création d'une arête libre : quatre brins liés par α_0 , α_1 et α_2 .

3.2.1.5 Déconnexion d'une arête

Déconnecter une extrémité d'arête consiste à séparer une arête d'un ensemble d'arêtes liées à un seul et même sommet (cf. figure 3.14). Pour cela, il suffit de modifier la liaison α_1 des brins passés en paramètres comme indiqué en définition 14.

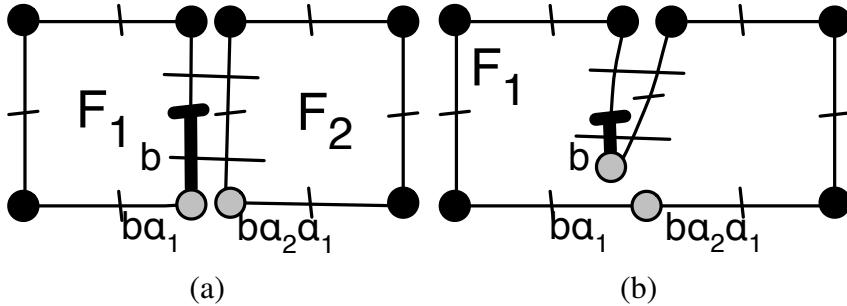


FIG. 3.14 – (a) Délier l'arête marquée en gras du sommet gris. (b) Résultat après déconnexion.

Définition 14 Soient $G = (B, \alpha_0, \alpha_1, \alpha_2)$ une 2-g-carte fermée et $d \in B$ tel que $d\alpha_1 \neq d\alpha_2$. Soient $da_1 = d\alpha_1$ et $da_2a_1 = d\alpha_2\alpha_1$. En dimension 2, déconnecter l'extrémité d'une arête engendre une 2-g-carte $G' = (B, \alpha_0, \alpha'_1, \alpha_2)$ telle que :

- $\forall b \in B - \{d, d\alpha_1, d\alpha_2, da_2a_1\}$, $ba'_1 = ba_1$
- $da_1\alpha'_1 = da_2a_1$
- $d\alpha'_1 = da_2$

3.2.1.6 Insertion d'un sommet sur une arête

Intuitivement, l'insertion d'un sommet sur une arête correspond au découpage d'une arête. Cette opération est généralement utilisée pour lier des entités à une arête ou pour lui donner une

apparence plus lisse dans le cas d'un plongement sommet en la subdivisant. L'insertion d'un sommet s'effectue en insérant quatre nouveaux brins à la g-carte, accompagnés de modifications locales des liaisons α_0 des brins déjà présents, ainsi que l'ajout de liaison α_1 et α_2 sur les nouveaux brins (cf. figure 3.15(b)).

Définition 15 Soient $G = (B, \alpha_0, \alpha_1, \alpha_2)$ une 2-g-carte fermée et $d \in B$. En dimension 2, l'insertion d'un sommet sur une arête engendre une 2-g-carte $G' = (B \cup \{b_1, b_2, b_3, b_4\}, \alpha'_0, \alpha'_1, \alpha'_2)$ telle que :

- $\forall b \in B - \{b_1, b_2, b_3, b_4\}, b\alpha'_1 = b\alpha_1, b\alpha'_2 = b\alpha_2$
- $\forall b \in B - (\{b_1, b_2, b_3, b_4\} \cup \langle\rangle_{N-\{1\}}(d)), b\alpha'_0 = b\alpha_0$
- $d\alpha'_0 = b_1, d\alpha_2\alpha'_0 = b_3$
- $b_2\alpha'_0 = d\alpha_0, b_4\alpha'_0 = d\alpha_2\alpha_0$
- $b_1\alpha'_1 = b_2, b_3\alpha'_1 = b_4$
- $b_1\alpha'_2 = b_3, b_2\alpha'_2 = b_4$

3.2.1.7 Suppression et contraction d'arêtes

Les opérations de suppression et de contraction de cellules sont définies dans [20] pour la dimension n .

Intuitivement, la suppression d'arête (identifiée par d sur la figure 3.15(a)) correspond généralement à la fusion de deux faces (cf. figure 3.15(c)). En 2D, elle correspond à la suppression des quatre brins composant l'orbite arête, ainsi que la modification des liaisons α_1 qui liaient l'arête supprimée au reste du modèle, afin d'obtenir des brins libres par α_1 .

La contraction d'arête (identifiée d sur la figure 3.15(a)) est l'opération duale de la suppression (cf. figure 3.15(d)). En 2D, elle correspond à la suppression des quatre brins de l'arête, ainsi que la modification des liaisons α_1 qui liaient l'arête contractée au modèle, à partir d'un parcours $\alpha_1\alpha_0$ qui permet de rechercher le plus proche brin n'appartenant pas aux brins supprimés. La contraction d'arête peut également être interprétée comme la combinaison des deux opérations précédemment décrites. Dans ce cas, la contraction se résume à la suppression de l'arête à contracter suivie de l'identification éventuelle des sommets qui lui étaient incidents.

3.2.2 Plongement géométrique

Le but du plongement géométrique est d'associer une forme géométrique à chaque entité. Dans le cas de l'animation 2D, différents plongements peuvent être utilisés : plongement sommet et plongement arête.

Le plongement sommet consiste à attribuer une position dans l'espace à chaque 0-cellule suivant une date donnée. Une telle position peut s'exprimer sous la forme d'une fonction temporelle $f : t \rightarrow \mathbb{R}^2$. La forme d'une arête liant deux sommets est déduite par interpolation linéaire de leurs coordonnées. La forme des faces est déduite par l'assemblage des arêtes la constituant (cf. figure 3.16(b)). Ce plongement est le plus simple que l'on puisse utiliser, mais les arêtes de l'objet sont forcément des segments et les faces des morceaux de plan.

Pour lever cette limitation et fournir des formes « gauches », il est possible d'utiliser le plongement arête. Ce plongement consiste à attribuer une fonction temporelle $f : t, p \rightarrow \mathbb{R}^2$ à chaque 1-cellule où $p \in [0; 1]$ correspond à une coordonnée curviligne. La forme d'une face est déduite par l'assemblage des arêtes constituant son bord (cf. figure 3.16(c)). Ce plongement doit

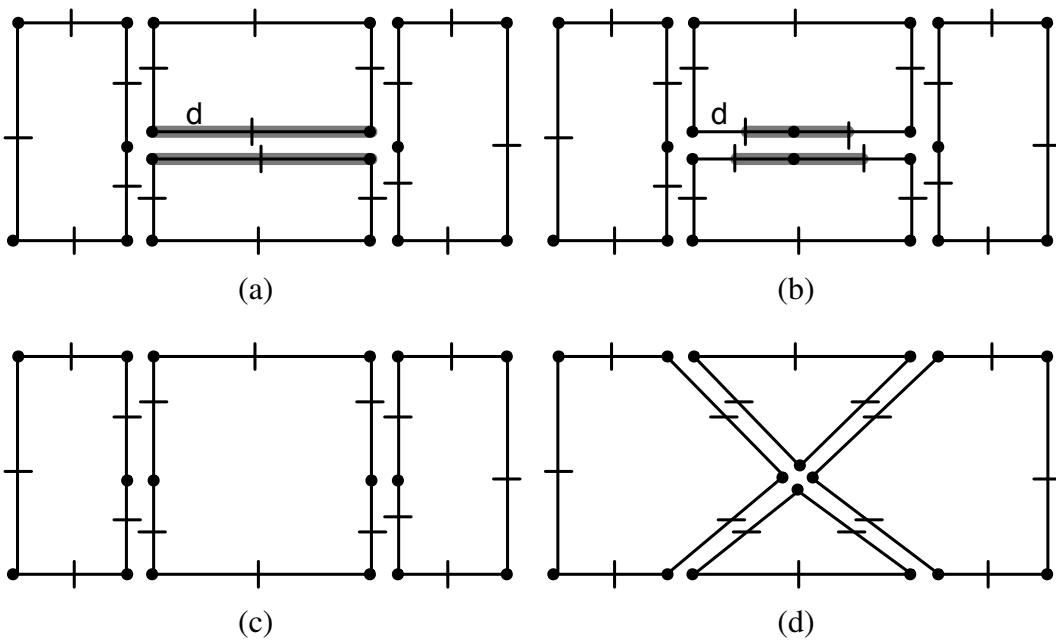


FIG. 3.15 – (a) 2-g-map de départ, les opérations s’effectuent à partir du brin d , l’arête grise est concernée (les relations α_2 ne sont pas affichées pour alléger la représentation). (b) Insertion d’un sommet (création de quatre nouveaux brins). (c) Suppression d’une arête (suppression de quatre brins). (d) Contraction d’une arête (suppression de quatre brins).

respecter quelques contraintes. Tout d’abord, toutes les arêtes incidentes à un sommet doivent admettre un point d’intersection commun correspondant à la coordonnée curviligne 0 ou 1 de chaque plongement (cf. figure 3.16(d)). Ensuite, par définition, le plongement de l’arête ne doit pas admettre de boucle (cf. figure 3.16(e)). Enfin, les arêtes constituant une face ont un point d’intersection correspondant à une extrémité seulement si elles se succèdent (cf. figure 3.16(f)).

Notre script permet de manipuler les deux types de plongements : dans le cas du 0-plongement, l’utilisateur fournit la courbe 3D des points en mouvement de la ligne polyédrique ; pour le 1-plongement, il devra fournir une surface représentant la ligne en mouvement.

3.2.3 Contrôle de l’animation

Jusqu’à maintenant, nous avons décrit notre structure, ses propriétés et ses opérations de construction. Cependant, il nous faut des méthodes pour construire l’animation. Pour cela, nous proposons une méthode qui énumère dans un script les opérations à appliquer au modèle. Cependant, un tel script nécessite un mécanisme pour désigner les entités et indiquer ainsi les entités concernées par une opération. Nous décrivons au préalable ce mécanisme.

3.2.3.1 Désignation des entités

Les entités géométriques sont désignées par un nom. Ce nom sert de paramètre au script. Nous pouvons décider de désigner indépendamment les sommets, les arêtes et les faces. Cependant, en 2D, la désignation des arêtes suffit à désigner toutes les entités. En effet, une arête est incidente à au plus deux faces, désignées *left* et *right* suivant la direction relative à sa désigna-

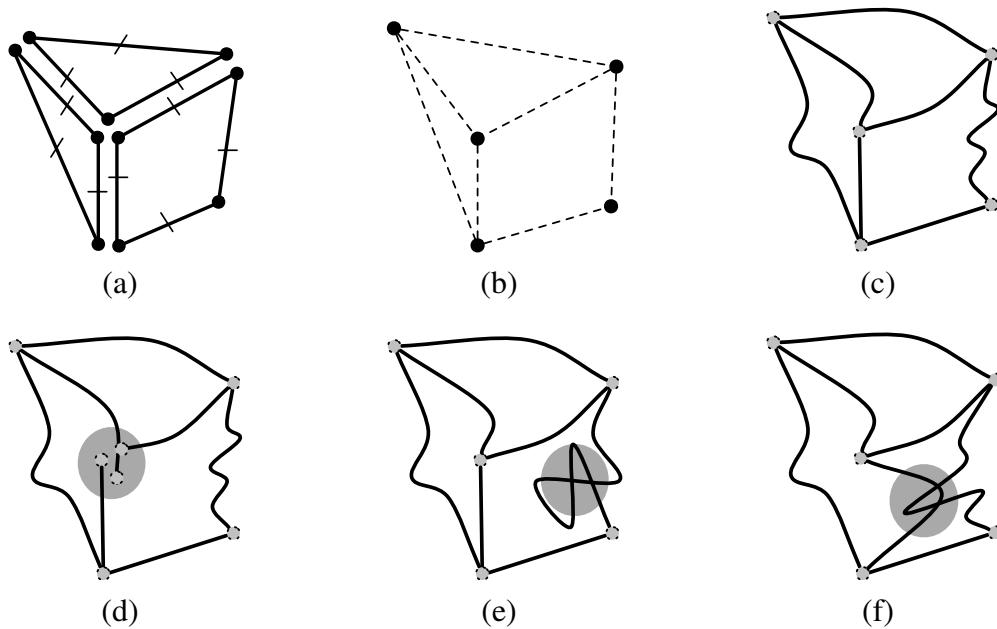


FIG. 3.16 – (a) Exemple de 2-g carte à plonger. (b) Plongement sommet : les sommets sont associés à des points de l'espace puis sont reliés entre eux par un segment de droite. (c) Plongement arête : les arêtes sont associées à des courbes, leurs intersections fournissent les sommets. Problèmes de plongement : (d) les arêtes incidentes au sommet central n'ont pas un point d'intersection unique, (e) la courbe associée à l'une des arêtes s'auto-intersecte, (f) les deux courbes associées à deux arêtes d'une face commune s'intersectent.

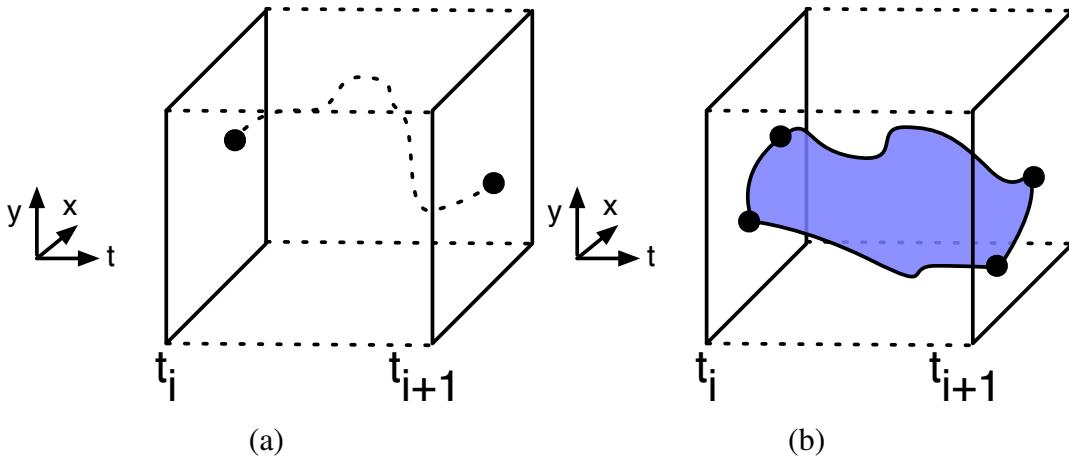


FIG. 3.17 – En 2D : (a) 0-plongement temporel = courbe 3D. (b) 1-plongement temporel = surface 3D.

tion et deux sommets *begin* et *end*. Chaque arête est donc associée à un identifiant qui permet de la désigner (Figure 3.18). Notons que puisque la désignation s'opère à partir d'une arête, il existe plusieurs moyens de désigner les sommets et les faces. Le nombre de façons de les désigner est respectivement égal au degré du sommet et au nombre d'arêtes formant le bord de la face.

Topologiquement, en 2D, dans le contexte d'une 2-g-carte fermée, l'orbite arête est constituée de quatre brins. Un nom est associé à un seul d'entre eux et le sommet associé à ce brin reçoit la désignation *begin*, permettant ainsi de préciser l'orientation de l'arête.

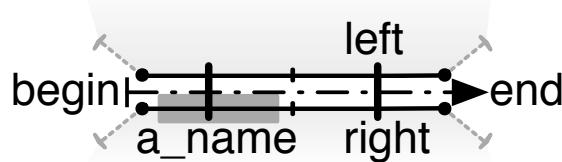


FIG. 3.18 – Mécanisme de désignation des 0, 1, 2-cellules à partir du brin portant la désignation de la 1-cellule.

3.2.3.2 Script bas niveau pour la 2D

Pour manipuler la partie structurelle de notre modèle d'animation, nous avons opté pour un script bas niveau permettant de modifier la structure topologique et la structure temporelle. Ce script est écrit en langage Lua (<http://www.lua.org>) interprété par le noyau de l'application. Les fonctions appelées s'appliquent sur une ou plusieurs entités d'une des 2-g-cartes du modèle structurel. Aucun contrôle de cohérence géométrique n'est effectué à ce niveau, seul le modèle topologique est assuré d'être cohérent car toutes les opérations topologiques utilisées assurent la préservation de cette cohérence au cours de l'animation. Notons que cette démarche est indépendante de la dimension du plongement : en utilisant les mêmes fonctions, il est tout à fait possible d'écrire des programmes d'évolutions de surfaces.

Dans le script, il est possible de préciser les plongements géométriques associés aux cellules. Ils sont de deux types : $f : t \rightarrow \mathbb{R}^2$ pour les 0-cellules ou $f : s, t \rightarrow \mathbb{R}^2$ (s une coordonnée curviligne) pour les 1-cellules [41].

En résumé, pour contrôler l'évolution d'une subdivision, les opérations topologiques suffisantes contenues dans le script sont :

- créer une arête désignée ;
- contracter une arête ;
- supprimer une arête ;
- identifier des sommets ;
- déconnecter une extrémité d'arête ;
- insérer n sommets sur une arête ;
- éclater un sommet ;
- éclater un chemin d'arêtes.

Ces fonctions assurent la cohérence entre les modèles topologique et sémantique. Cependant, l'utilisation seule des primitives *déconnecter* et *créer* une arête pourrait engendrer un modèle qui ne remplit pas la contrainte de connexité des g-cartes. Pour cette raison, ces deux fonctions sont seulement utilisées dans des étapes intermédiaires des modifications topologiques et la contrainte de connexité est toujours assurée à la fin de ces étapes. En effet, ces fonctions sont associées à l'utilisation des fonctions *identifier*, *supprimer* et *contracter*.

Dans l'implantation de l'instanciation 2D de notre modèle, nous avons décidé de créer par défaut quatre entités de départ – désignées *border_left*, *border_top*, *border_right* et *border_bottom* – qui constituent le cadre de la scène et sont intégrées au modèle topologique.

Le script permet de créer toutes sortes d'animations 2D. Le listing 3.1 présente l'exemple d'une balle qui rebondit sans fin contre quatre murs. L'idée est de représenter la balle par une simple face plongée à l'aide d'une courbe en forme de cercle. Pour que la balle appartienne à la face constituant la pièce, on insère une arête dite fictive qui relie la balle et le bord de la pièce.

La figure 3.19 présente les étapes du script 3.1. Lignes 1 à 10, création d'une animation avec un cadre d'une largeur de 50 et d'une hauteur de 20 3.19(a). Ligne 13, création de l'arête *fictive_edge* qui reliera la face représentant la balle au bord afin de l'inclure dans la face représentant la scène. Ligne 14, identification d'un sommet de l'arête (*fictive_edge, begin*) à un des sommets du cadre (*border_left, begin*) (cf. figure 3.19(b)). Sur cette ligne, on remarque le paramètre *left*, qui indique la face à laquelle on veut identifier les deux sommets. Ligne 17, création de l'arête *ball* représentant le bord de la balle. Ligne 20, pour que la balle appartienne à la face de la scène, il faut l'identifier au sommet libre (*fictive_edge, end*) de l'arête fictive. Ligne 21, il faut ensuite créer une face pour représenter la balle. Pour cela, nous identifions l'extrémité libre de l'arête avec le sommet (*fictive_edge, end*). Les lignes 24 à 29 définissent un plongement de type cercle. Les lignes 32 à 53 définissent le plongement de l'arête représentant la balle. Ce plongement simule le comportement d'une balle qui rebondit sur le cadre de la scène par l'intermédiaire d'un comportement simple qui consiste à changer de direction quand la balle touche le bord. La ligne 56 affecte le plongement à l'arête *ball*. La figure 3.20 illustre quelques étapes de l'animation.

Listing 3.1 – Script d'animation d'une balle qui rebondit contre quatre murs

```

1 — Bibliothèque
2 require 'utils'

— création d'une animation de taille 50*20
5 width      = 50
height     = 20
animation  = CAnimation:new(gmv, width, height)

— création de la première image clef
10 firstFrame = animation:addFirstDiscretePlane2d(0)

— création de l'arête fictive liant la balle au bord
13 firstFrame:createInterface("fictive_edge")
firstFrame:identification("border_left", begin, "fictive_edge", begin, left)
15
— création de l'arête représentant la balle
17 firstFrame:createInterface("ball")

— liaison de la balle à l'arête fictive
20 firstFrame:identification("fictive_edge", end, "ball", begin, left)
firstFrame:identification("fictive_edge", end, "ball", end, left)

— fonction de plongement d'un cercle quelconque
24 circle = function (x, y, r)
    return function(p, _)
        — la forme ne varie pas au cours du temps
25        p = p * math.pi * 2
        return math.cos(p) * r + x, math.sin(p) * r + y
    end
end
30
— fonction de plongement qui simule de manière très simple les rebonds
32 rebond = function()
    local lastt = 0
    local v     = .5
35    local a     = -math.pi / 6
    local r     = 1
    local x, y = width/2, height - r
    return function (p, t)
        if lastt ~= t then
            x, y = x + math.cos(a) * v, y + math.sin(a) * v
40        if x - r <= 0 or x + r >= width then
            a = -a + math.pi
        end
    end
end

```

```

x = clamp(x, 0, width)
end
45 if y - r <= 0 or y + r >= height then
    a = -a
    y = clamp(y, 0, height)
end
lastt = t
50 end
    return circle(x, y, r)(p, t)
end
end

— Affectation du plongement de l'arête ball
55 firstFrame :setSymboleEdgeEmbedding("ball", rebond())

```

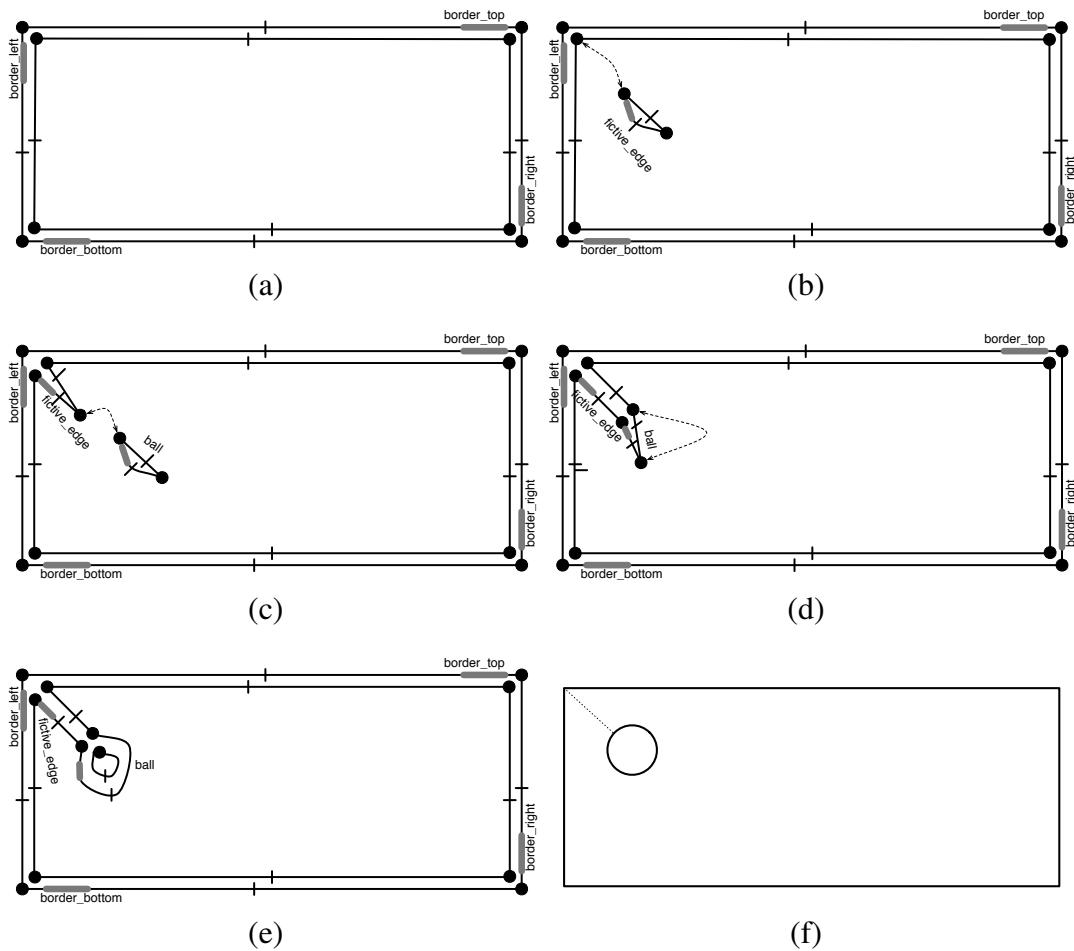


FIG. 3.19 – Étapes de création de la scène de la balle. (a) Scène de départ. (b) Crédit et identification d'une arête désignée "fictive_edge" avec le bord du cadre. (c) Crédit et identification de l'arête représentant le bord non fermé de la balle avec l'extrémité libre de l'arête fictive. (d) Identification de l'extrémité libre de l'arête représentant la balle avec l'arête fictive afin de créer la face représentant la balle. (e-f) Résultat final en vue éclatée et compacte.

3.3 Bilan

Ce chapitre a d'abord proposé deux modèles topologiques permettant de représenter l'évolution temporelle d'une structure. Ils se différencient essentiellement par le fait que le temps



FIG. 3.20 – Images de l’animation d’une balle rebondissant contre un mur.

est représenté de façon continue ou discrète. Nous avons montré que ces deux modèles sont équivalents, mais que la version discrète permet des manipulations beaucoup plus simples, en se basant sur une succession de cartes sans liens topologiques entre elles. Ce chapitre propose une instantiation 2D de ce modèle. L’ensemble des opérations nécessaires à la représentation des animations a été exposé. Le calcul de l’animation peut alors se faire au travers d’un script qui indique l’ensemble des modifications que subit la carte initiale et permet ainsi de générer la succession de cartes. Ce script repose sur un mécanisme de désignation des entités, subordonné aux modifications qu’elles subissent. Ce script offre ainsi une approche de type *modélisation procédurale* où l’animation est décrite en termes de modifications successives de la structure. Cependant, l’écriture d’un tel script est très délicate et fastidieuse. En effet, le script est de type « bas-niveau » et ne masque pas la complexité de la structure. Il est plus pratique pour l’animateur de spécifier des évolutions dans un format plus abstrait (« haut-niveau »). Il nous faut donc fournir une sur-couche à notre modèle d’animation visant à décrire des évolutions de façon globale, tout en permettant à ces évolutions d’être transcris automatiquement en des descriptions locales telles que nous les avons définies dans le script. Cette sur-couche vise également à assurer la cohérence topologique et géométrique, c’est-à-dire contrôler la génération de l’animation. Au-delà d’un modèle de type modélisation procédurale, c’est donc bien un système d’animation à part entière que nous souhaitons concevoir et qui fait l’objet du chapitre suivant.

UN SYSTÈME D’ANIMATION TOPOLOGIQUE

Dans le chapitre précédent, nous avons présenté un modèle topologique animé, représentant l’évolution d’une structure au cours du temps. Ce chapitre propose un système permettant de construire l’animation en la contrôlant, c’est-à-dire en assurant en particulier la cohérence topologique. Deux approches sont possibles. Soit nous abordons le problème sous l’angle de la modélisation géométrique : dans ce cas, nous cherchons des opérations haut-niveau, généralement dépendantes de l’application finale, permettant de construire progressivement le modèle. Soit nous abordons le problème sous l’angle de l’animation pure, voire de la simulation ; en se basant sur des modèles de phénomènes haut-niveau, nous proposons un système générant automatiquement les évolutions topologiques de bas-niveau. C’est cette seconde approche que nous avons choisi d’explorer. En effet, elle nous assure de traiter les phénomènes dans l’ordre de leur apparition, ce qui est une garantie précieuse pour le contrôle et la cohérence du résultat. De plus, elle nous permet de faire le lien entre des phénomènes haut-niveau et les primitives de modifications bas-niveau.

Ce chapitre est organisé de la façon suivante : nous présentons d’abord notre modèle d’animation, en précisant ses objectifs puis ses trois principales composantes : structurelle, sémantique et événementielle. Nous décrivons ensuite l’instanciation de ce modèle en 2D.

4.1 Le modèle d’animation

Dans notre modèle d’animation, nous souhaitons modéliser des phénomènes haut-niveau, spécifiques à l’application finale. Ces phénomènes sont retranscrits en primitives de transformations bas-niveau sur une structure topologique spatio-temporelle. Dans l’idéal, les phénomènes sont issus d’une simulation ou d’un système d’animation comportementale pour assurer un certain réalisme des évolutions calculées. Cependant, pour valider le système, nous avons opté pour une approche plus immédiate de type scénario, où l’ensemble des phénomènes sont décrits dans l’ordre de leur apparition. Le scénario est orienté métier et directement manipulé par l’utilisateur.

Notre système d'animation se base tout d'abord sur une composante structurelle, en l'occurrence, le modèle topologique spatio-temporel présenté au chapitre précédent.

Chaque phénomène est associé à des évolutions géométriques et des modifications topologiques de la structure, qu'il s'agit de détecter. Notre modèle comprend donc nécessairement une composante permettant de reconnaître les conditions d'un changement et d'appliquer les modifications. Cette composante est qualifiée d'*événementielle*. C'est cette composante qui nous garantit la cohérence topologique du système.

Par ailleurs, l'animation construite doit pouvoir être interprétée par l'utilisateur, ou plus précisément par l'expert du domaine d'application. Pour cela, une autre composante est nécessaire : la composante sémantique. Elle revêt ici deux fonctions. Elle inclut tout d'abord le script présenté au chapitre précédent qui donne un aperçu en termes d'opérations topologiques élémentaires subies par la structure initiale. Il faut cependant coupler cette séquence de transformations par un système de suivi de l'historique de chaque entité. Pour cela, il faut pouvoir déterminer l'origine des entités. Nous étendons alors le système de désignation présenté au chapitre précédent.

4.1.1 La composante événementielle

La composante structurelle présentée au chapitre précédent suffit, à elle seule, à décrire une animation par une suite d'opérations topologiques et d'affectations de plongements sommets. Le problème est de déterminer à quel instant a lieu chaque modification topologique, les opérations topologiques à appliquer pour mettre en œuvre les modifications et les nouveaux plongements. Pour répondre à ces questions, un système à événements discrets est adjoint au modèle. Les événements générés peuvent provenir de diverses origines. Certains sont générés directement par le scénario, i.e. aux évolutions haut-niveau du scénario peuvent correspondre des événements dit *initiaux*. Les autres événements, généralement plus nombreux, résultent de l'évolution topologique du système. Pour ces derniers événements, une approche prédictive permet de déterminer, en fonction des évolutions de la géométrie, la date d'un événement topologique, c'est-à-dire l'instant où une entité rencontre une autre entité. Par exemple, si un point se déplace le long d'une arête, un événement est déclenché lorsque le point atteint l'extrémité de cette arête. Enfin, certains événements résultent de la collision entre des entités initialement indépendantes.

La gestion par événements discrets est similaire à celle utilisée habituellement en animation [23]. Une fois détectés, les événements sont ajoutés dans une file à priorité triée suivant le temps. Dans l'ordre de leur apparition, les événements sont extraits de la file puis traités. Le traitement d'un événement débute par la vérification de sa validité (en effet, des modifications topologiques précédentes ont pu rendre caduque l'apparition d'un événement précédemment détecté [23]). Si l'événement est valide, il est traité suivant le contexte de la scène et entraîne une série de transformations topologiques, géométriques et de désignations (voir la composante sémantique). En effet, les modifications à appliquer dépendent de l'application et généralement du phénomène simulé. Ces transformations sont appliquées soit à la *n-g*-carte courante si la date qui lui est associée est égale à la date de l'événement, soit à une nouvelle *n-g*-carte créée par duplication de la précédente et dont la date est celle de l'événement. Les transformations de plongements incluent les trajectoires des objets. En analysant ces mouvements, de nouveaux événements sont détectés et sont rajoutés à la file. Ce processus est alors répété aussi souvent que nécessaire et permet de calculer l'animation.

L'algorithme 4 suivant résume le traitement de la file d'événements.

```

lastDate ← -∞
tant que non file_evenement.vide() faire
    evt = file_evenement.defiler()
    si lastDate > evt.date() alors
        Afficher message d'erreur
        Sortir du programme
    lastDate ← evt.date()
    // est_valide = entités toujours existantes ? et
    // collision toujours valide ?
    si evt.est_valide() alors
        traiter_collision_suivant_contexte(evt, scène)
        pour chaque sommet s dont le plongement a changé faire
            file_evenement.ajouter(detecter_plus_proche_collision_potentielle(s),
            evt.date())
    
```

Algorithme 4 : Traitement de la file d'événements (nD)

Pour une interception des phénomènes topologiques, nous nous basons sur une détection des auto-collisions de la partition, à partir des trajectoires des entités. Plus encore, dans une partition, les collisions ne peuvent apparaître qu'entre des entités qui sont incidentes à une même cellule. Par exemple, dans une partition 3D, un sommet ne peut rencontrer qu'un sommet, une arête ou un face délimitant un volume dont il est déjà sommet. Ceci nous permet de restreindre la recherche des entités à inspecter lors de la phase de prédition d'une collision. Une approche similaire de collision prédictive basée sur la topologie a été proposée récemment [36], mais ne se place pas dans le cas des auto-collisions. De ce fait, seule une partie des accélérations proposées est adaptable à notre système. En outre, un mécanisme permettant de pister la position topologique des points dans une carte est proposé dans ces travaux, mais n'est pas adapté au cas des auto-collisions (car tous les points de notre partition que nous manipulons font partie de la carte).

Ensuite, lorsqu'un événement survient, son traitement n'est généralement pas unique pour une application donnée. Suivant le phénomène reproduit, le contexte de l'événement et le type des entités impliquées, le traitement diffère. Une fois l'événement détecté et validé, il est nécessaire d'identifier le type de traitement à appliquer. Pour cela, la structure locale des entités impliquées est analysée. Nous nous basons sur des identifications de motifs auxquels est associée la suite d'opérations à appliquer au modèle pour traiter l'événement. Si l'on considère le graphe que représente une structure topologique (une n -g-carte est considérée comme un graphe étiqueté dont les arêtes représentent les liaisons α_i), la recherche de motif peut paraître coûteuse (cf. algorithme 6). Cependant, nous profitons des entités impliquées dans l'événement pour définir des appariements obligatoires entre la structure manipulée et les motifs testés. L'identification du type de traitement est donc une simple vérification d'isomorphisme et non une recherche.

Il faut noter que cette définition des traitements alourdit considérablement l'utilisation de notre système pour une application donnée, puisque cette partie est actuellement programmée dans le moteur. Une approche plus souple est souhaitable. Plusieurs travaux utilisant un cadre formel à base de règles de réécritures de graphes [57, 29] pourraient permettre de généraliser la description de ces modifications. Ces approches décrivent la transformation d'un graphe à partir

d'un motif décrit dans une règle. Ces règles permettent donc de décrire une structure recherchée avec des propriétés (prédicats) à vérifier et leurs transformations. Ces règles sont généralement définies à partir d'une grammaire et sont donc facilement manipulables par un utilisateur qui pourrait adapter les règles à l'application visée. Cependant, dans notre cas, la phase d'identification du motif est beaucoup plus rapide que le recours à la détection d'homéomorphisme de graphe.

Pour déterminer l'ensemble des événements nécessaires à une application, la première étape est d'identifier tous les cas de collisions possibles dans la dimension considérée (l'étude des collisions dans le cas 2D est décrite en section 4.2.1). On définit ensuite les algorithmes permettant de sélectionner les transformations à appliquer, selon des informations sémantiques supplémentaires propres à l'application. Finalement, les algorithmes de transformation sont à écrire.

4.1.2 La composante sémantique

La composante sémantique vise deux objectifs : représenter de façon explicite les modifications subies par la structure au cours du temps et fournir une représentation de l'historique des entités topologiques. Pour remplir ces deux objectifs, nous utilisons un mécanisme de désignation des entités et un script d'opérations topologiques.

La désignation consiste à associer des noms aux cellules de la scène. Un nom est soit choisi par l'utilisateur lors des créations de cellules, soit déterminé automatiquement selon l'enchaînement des opérations mises en jeu. Ces noms sont utilisés comme paramètres des descriptions (fournies dans le scénario de l'utilisateur) et des opérations topologiques et géométriques issues de ces descriptions. La désignation est hiérarchique, ce qui facilite la description de la genèse d'une entité en lui associant comme parent l'entité dont elle est issue. Par exemple, l'opération d'insertion de n sommets sur une arête résulte en $n + 1$ arêtes dont les désignations sont filles de la désignation de l'arête d'origine et ont ainsi comme préfixe la désignation de cette arête.

La composante structurelle représente seulement le résultat géométrique et topologique de l'évolution. Elle ne décrit qu'implicitement les transformations topologiques subies alors que le script de bas-niveau, décrit dans le chapitre précédent, les décrit explicitement. Nous avons donc choisi que la composante sémantique puisse produire automatiquement le script, car il retrace l'historique de construction du modèle et permet d'analyser, a posteriori, le résultat de l'animation. Pour fabriquer ce script, la composante sémantique récupère, lors du traitement des événements, la liste des opérations topologiques appliquées aux entités altérées. Cette liste forme une séquence chronologique d'opérations élémentaires qui s'appliquent sur la n-g-carte initiale au cours du temps. Chaque opération élémentaire est alors décrite dans le script par un appel de fonction, fonction qui s'applique sur une ou plusieurs entités d'une des 2-g-cartes du modèle temporel. Aucun contrôle de cohérence géométrique n'est effectué à ce niveau (ce contrôle est effectué dans l'étape précédente de génération du script), seul le modèle topologique est assuré d'être cohérent car toutes les opérations topologiques utilisées assurent la préservation de cette cohérence tout au long de l'animation. Notons que ce script permet, à partir de la carte initiale, de générer toute l'animation.

Ce mécanisme forme ainsi un pont entre l'aspect « haut-niveau » (descriptif) de notre modèle, dans lequel l'utilisateur contrôle l'animation, et l'aspect « bas-niveau » (structurel) sur lequel s'appliquent les décisions de l'utilisateur.

4.1.3 Interaction entre les composantes

La figure 4.1 montre le processus général de l'animation. En (a-b), l'utilisateur écrit un scénario pour décrire une animation (cette description est donc spécifique à l'application). Le scénario est décomposé en informations structurelles et en opérations sur des entités désignées. En (b-c), la description est analysée et convertie en une séquence d'événements. Le traitement de ces événements génère un script d'évolutions bas-niveau où chaque modification topologique est enregistrée. Ce processus utilise les informations structurelles et sémantiques. En (c-d), le script bas-niveau est interprété par le modèle topologique avec l'assistance de la composante sémantique qui lie les objets désignés aux entités topologiques. En (d), les opérations structurelles sont appliquées.

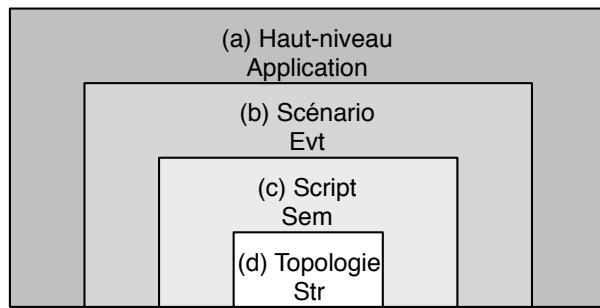


FIG. 4.1 – Utilisation des trois composantes (événementielle, sémantique, structurelle) dans le processus de création d'animations (Evt : Événementielle ; Sem : Sémantique ; Str : Structurelle)

La figure 4.2 résume les interactions entre les différentes composantes : le scénario est analysé et envoyé à la composante événementielle ; celle-ci réalise des opérations sur les entités désignées au travers de la composante sémantique, en réponse aux événements utilisateur déduits du scénario et des règles de transformations dédiées à l'application ; la composante sémantique crée un historique des modifications géométriques et topologiques et applique ces modifications sur la composante structurelle. Cette dernière joue un double rôle : non seulement elle correspond à une évaluation de l'historique des événements passés, mais elle est indispensable à la prédiction des collisions à venir entre les entités de la scène.

4.2 Étude du cas particulier 2D

La section précédente a décrit l'architecture globale de notre système d'animation. Cependant, la définition des événements est restée très abstraite. Il faut en effet se placer dans une dimension donnée pour étudier quels sont les événements possibles. Cette section décrit comment les événements liés à l'évolution d'une subdivision 2D sont générés et traités.

4.2.1 Types et résolution d'événements de collisions

Nous utilisons un plongement linéaire en associant un plongement aux 0-cellules uniquement. Les sommets de la scène évoluent librement dans un espace 2D, ce qui peut entraîner des collisions entre cellules, constituant avec les événements initiaux définis par l'utilisateur l'ensemble des événements possibles. En deux dimensions, seuls deux types de collisions peuvent

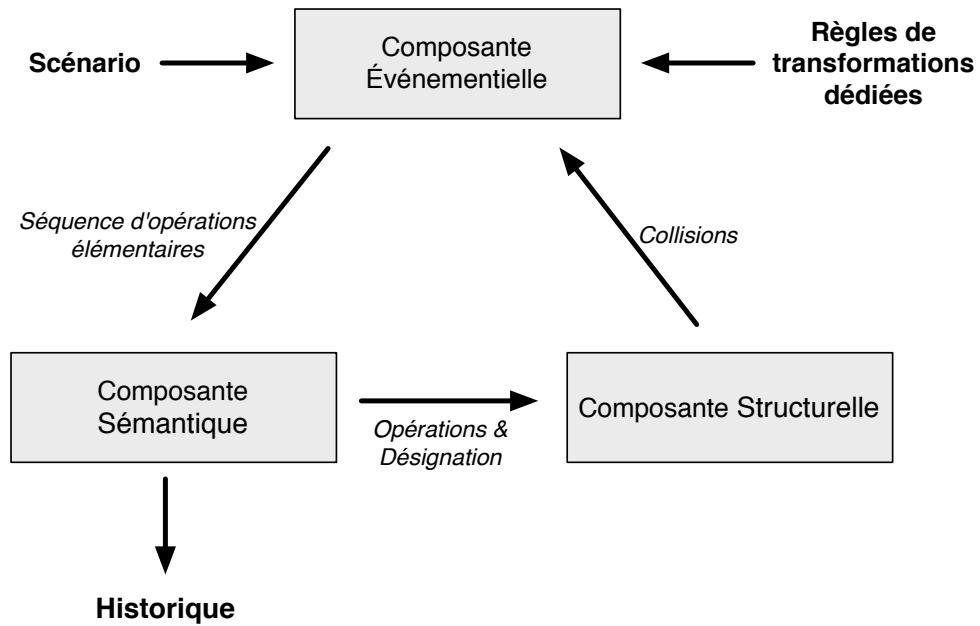


FIG. 4.2 – Interactions entre les composantes.

se produire, les collisions sommet/sommet et les collisions sommet/arête. En effet, avec un plongement temporel de sommets et un système dans un état cohérent, les collisions arête/arête n’ont pas besoin d’être détectées, car des collisions sommet/sommet ou sommet/arête sont détectées avant ; et ce, même dans le cas dégénéré où les arêtes sont parallèles (dans le cas d’un plongement non linéaire, les collisions arête/arête devraient être détectées). Nous séparons le traitement des collisions sommet/sommet, des collisions sommet/arête pour traiter les cas dégénérés de façon explicite [53]. De plus, le système de collision se basant sur une résolution formelle d’équations, la différenciation entre le cas sommet/sommet et sommet/arête est immédiate. Partant de ce constat, nous pouvons définir des classes de traitement associées aux différents cas. Cette classification permet de généraliser et de simplifier la description des traitements associés aux événements.

4.2.1.1 Détection des autocollisions

La détection consiste, pour tout sommet sur lequel un mouvement est appliqué, à calculer ses potentielles interactions avec son voisinage direct, c’est-à-dire avec les faces qui lui sont incidentes. On utilise ainsi l’information topologique pour limiter le nombre de couples d’entités à tester puisque nous nous sommes placés dans le cadre d’une partition. Les tests d’intersection se résument à chercher la plus proche collision (dans le temps) (1) entre chaque sommet s_0 modifié et les arêtes appartenant aux faces incidentes à s_0 ($\text{faces_sommet}(s_0)$), (2) entre chaque arête a incidente à s_0 ($\text{aretes_sommet}(s_0)$) et chaque sommet appartenant aux faces incidentes à s_0 ($\text{sommets_face}(\text{faces_aretes}(a))$), et (3) entre chaque sommet adjacent au sommet s_0 ($\text{sommets_sommet}(s_0)$) (i.e. longueur de l’arête courante devenant nulle). L’algorithme 5) récapitule ces différentes étapes. Des fonctions de la forme x_y sont utilisées, où x et y peuvent représenter des sommets, des arêtes ou des faces, et retournent l’ensemble des cellules x incidentes à une cellule y considérée. $dmintrouee$ correspond à la date de la plus proche intersection, $dtmp$ correspond à la date trouvée en cas d’intersection et $inter_trouee$ correspond à la

plus proche intersection trouvée. *inter_trouvee* est un triplet contenant le type d'intersection trouvé (*ve* : sommet/arête ; *vv* : sommet/sommet) ainsi que les deux cellules mises en cause.

```

Entrées :  $s_0 \in G_2, t_{min}$ 
inter_trouvee  $\leftarrow$  nil
// (1) recherche les intersections du sommet  $s_0$  avec les
    arêtes appartenant aux faces incidentes à  $s_0$ 
dmintrouee  $\leftarrow \infty$ 
pour chaque face  $f \in faces\_sommet(s_0)$  faire
    pour chaque arête  $a \in (aretes\_face(f) - aretes\_sommet(s_0))$  faire
        si intersection_arete_sommet(a, s0, dmintrouee, dtmp) alors
            si dmintrouee > dtmp alors inter_trouvee  $\leftarrow ("ve", s_0, a)$ 
    // (2) recherche les intersections des arêtes  $a$  incidentes
        au sommet  $s_0$  par rapport aux sommets appartenant aux
        faces incidentes à  $s_0$ 
    pour chaque arête  $a \in aretes\_sommet(s_0)$  faire
        pour chaque sommet  $s \in sommets\_face(faces\_arete(a)) - sommets\_arete(a)$ 
            faire
            si intersection_arete_sommet(a, s, dmintrouee, dtmp) alors
                si dmintrouee > dtmp alors inter_trouvee  $\leftarrow ("ve", s, a)$ 
    // (3) recherche les intersections des sommets  $s$  adjacents
        au sommet  $s_0$ 
    pour chaque sommet  $s \in sommets\_sommet(s_0)$  faire
        si intersection_sommet_sommet(s0, s, dmintrouee, dtmp) alors
            si dmintrouee > dtmp alors inter_trouvee  $\leftarrow ("vv", s_0, s)$ 
retourner inter_trouvee
```

Algorithme 5 : Détection de la plus proche collision potentielle (plongement sommet, 2D).

Compte tenu du 0-plongement, déterminer l'instant de collision entre un sommet C et une arête AB revient à détecter l'intersection entre un point et un segment de taille variable (déformable). Provot [58] a proposé de détecter cette intersection en vérifiant les instants t où les points A , B et C sont alignés. En 2D, cela revient à détecter que l'angle entre la normale $\vec{n}(t)$ à $\vec{AB}(t)$ et $\vec{AC}(t)$ doit être nul, autrement dit que $\vec{n}(t) \cdot \vec{AC}(t) = 0$. Cette formulation permet de ramener le calcul à une équation à une inconnue. De plus, l'équation peut facilement être ramenée à une inéquation permettant de traiter les erreurs numériques en recherchant des solutions pour $\vec{n}(t) \cdot \vec{AC}(t) < \epsilon$. Cependant, pour mener analytiquement le calcul à terme, il faut faire des hypothèses sur le mouvement des points, généralement un mouvement rectiligne uniforme. Cette hypothèse sur une certaine classe de mouvement des points est cependant trop restrictive. Par exemple, un point se déplaçant le long d'une arête dont les sommets ont un mouvement rectiligne uniforme n'a généralement pas un mouvement rectiligne uniforme, le point considéré sort donc de la classe des mouvements autorisés. Nous avons donc opté pour une approche plus générale : nous examinons s'il existe une intersection entre C et la droite D support de AB , puis testons si l'intersection appartient à l'arête. Il faut alors résoudre $A(t) + \alpha_1 \vec{AB}(t) = C(t)$, i.e. trouver les instants t^i tels qu'il existe α_1 vérifiant l'équation, puis vérifier pour chaque t^i que $0 \leq \alpha_1^i \leq 1$. Pour résoudre l'équation dans le cas général, nous utilisons actuellement un moteur de résolution d'équations formelles.

Concernant le cas dégénéré de collision entre les sommets A et B , l'approche est similaire. Nous recherchons les instants t^i respectant $A(t) = B(t)$, via le même moteur de résolution.

4.2.1.2 Traitement

Les collisions sommet/arête(Figure 4.4(a)) engendrent trois types de réponses. La première est de simplement coller les entités en collision (Figure 4.4(b) - Cas 1). La deuxième est de donner la priorité au sommet qui traverse l'arête en la découplant (Figure 4.4(c) - Cas 2). La troisième est de donner la priorité à l'arête qui sépare les arêtes incidentes au sommet (Figure 4.4(d) - Cas 3). L'affectation de la priorité au sommet ou à l'arête considérée est déterminée par l'utilisateur, en fonction du cadre d'application.

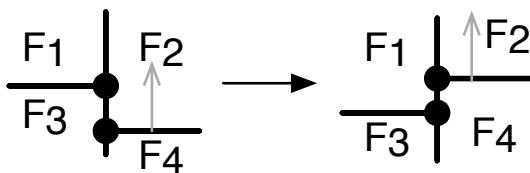


FIG. 4.3 – Glissement d'un sommet sur un autre avec changement des relations d'adjacence entre les quatre faces.

Les collisions sommet/sommet entraînent également trois types de réponses. Si les deux sommets sont liés par une arête, un glissement topologique d'un sommet sur un autre est réalisé (Figure 4.3). Sinon, soit les entités en collision sont collées comme dans le cas d'une collision sommet/arête (Figure 4.5(b) - Cas 1); soit une priorité est donnée à l'un des deux sommets (Figure 4.5(c) - Cas 2). Cette seconde approche pose un problème de décision si le sommet non prioritaire est de degré supérieur à deux, car il est difficile de prévoir la structure topologique résultante ainsi que le plongement des nouveaux sommets dans le cas de trajectoires non rectilignes (Figure 4.5(d) - Cas 3). En effet, dans le cas d'une trajectoire rectiligne, nous pouvons calculer le résultat de l'intersection au moment $t + \epsilon$ et en déduire la répartition des arêtes incidentes au sommet v sur les arêtes $e1$ et $e2$ (Figure 4.6). Dans le cas d'une trajectoire non rectiligne, le calcul des nouveaux plongements de sommets et de la topologie est délicat car la répartition des arêtes peut changer au cours du temps comme par exemple pour l'arête e (Figure 4.7). Pour déterminer comment séparer l'ensemble des arêtes, on peut utiliser la tangente de la trajectoire au point d'impact. Le changement de répartition d'arêtes s'effectue par des sommets après les collisions sommet/sommet liés par une arête.

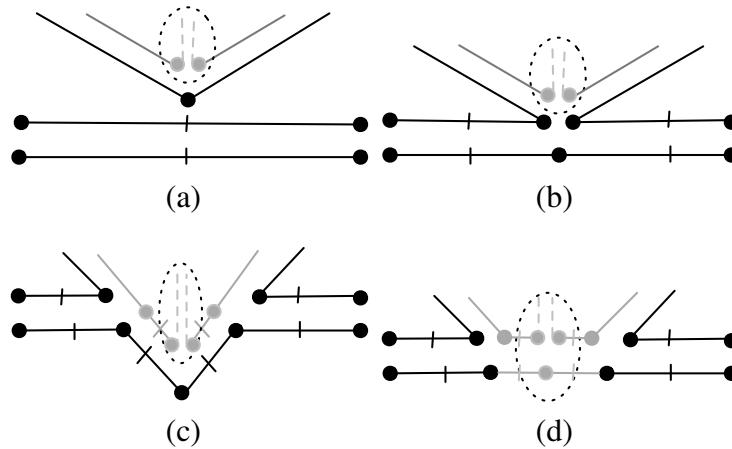


FIG. 4.4 – Cas de collisions sommet/arête. Les pointillés représentent la répétition du motif encerclé. Par exemple, dans la figure (a), la zone en pointillés représente un nombre quelconque d’arêtes incidentes au sommet qui vont entrer en collision avec l’arête horizontale. (a) État avant collision. (b) Cas 1 : le sommet se colle à l’arête. (c) Cas 2 : le sommet est prioritaire sur l’arête. (d) Cas 3 : l’arête est prioritaire sur le sommet.

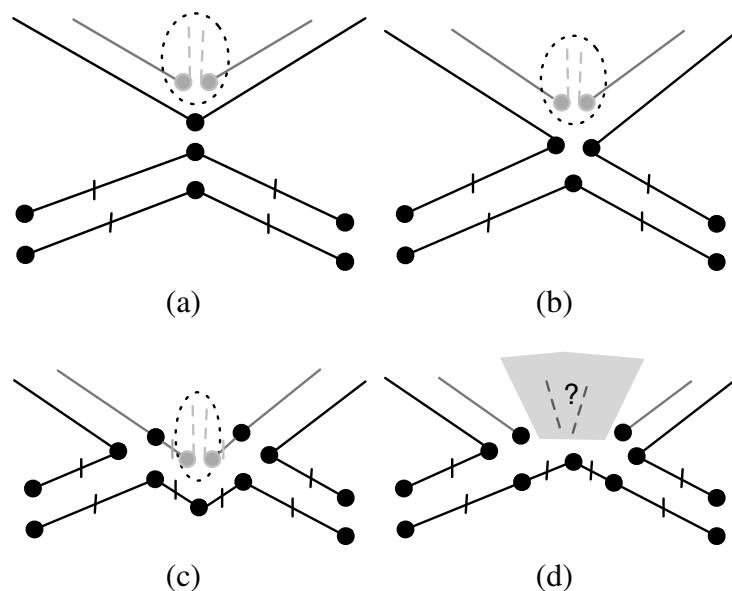


FIG. 4.5 – Cas de collisions sommet/sommet (les pointillés représentent la répétition du motif encerclé). (a) État avant collision. (b) Cas 1 : le sommet se colle à l’autre sommet. (c) Cas 2 : le sommet supérieur est prioritaire sur l’autre de degré ≤ 2 . (d) Cas 3 : le sommet inférieur est prioritaire sur l’autre de degré > 2 : les modifications topologiques du sommet supérieur dépendent du nombre d’arêtes qui lui sont incidentes et des trajectoires des sommets inférieur et supérieur.

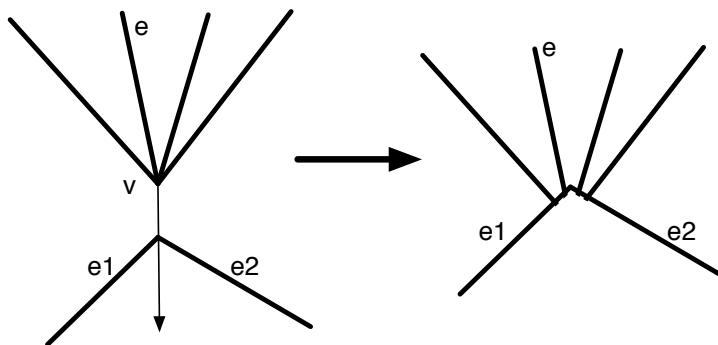


FIG. 4.6 – Résolution du cas 3 de la figure 4.5(d) dans le cas rectiligne.

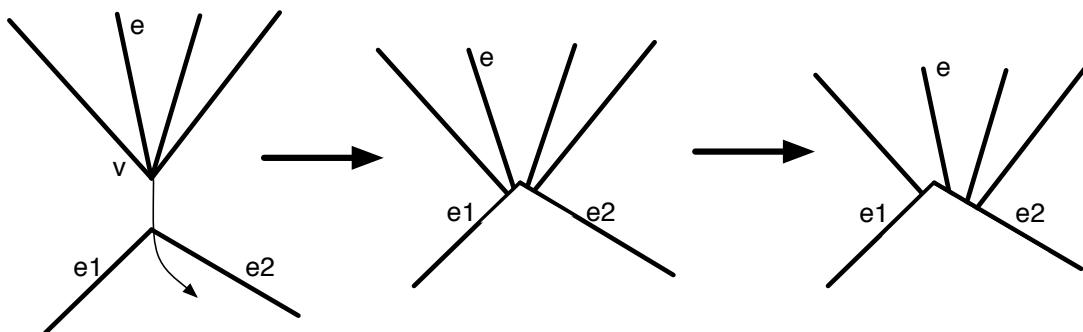


FIG. 4.7 – Problème du cas 3 de la figure 4.5(d) dans le cas non rectiligne. Après l'intersection de v (dont la trajectoire est symbolisée par la flèche) à la jonction des arêtes $e_1 - e_2$, un glissement des arêtes s'effectue en suivant la trajectoire de v .

4.2.2 Calcul de trajectoires

Pour implanter notre modèle en 2D et réaliser quelques animations, nous avons défini plusieurs types de trajectoires.

4.2.2.1 Trajectoire simple

La trajectoire la plus simple est un déplacement entre deux points $P_1(t)$ et $P_2(t)$ de l'espace débutant à une date t_{begin} et s'arrêtant à une date t_{end} (cf. figure 4.8) : $P(t) = P_1(t) * (1 - p) + P_2(t) * p$ avec $p = \frac{t - t_{begin}}{t_{end} - t_{begin}}$

Avec cette formulation, si $P_1(t) = cst$ et $P_2(t) = cst$, alors la vitesse du point est constante et la trajectoire rectiligne.

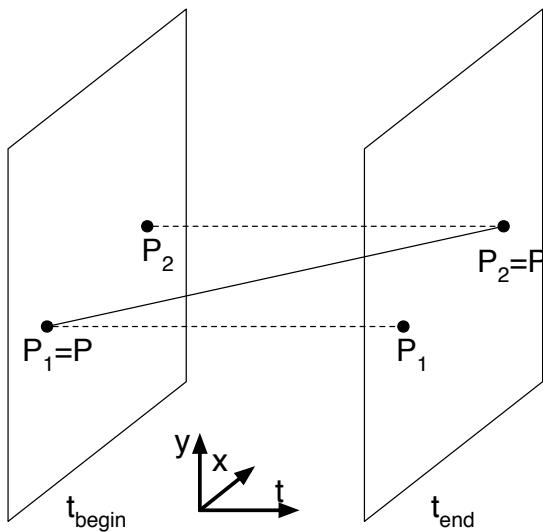


FIG. 4.8 – Interpolation entre deux sommets au cours du temps.

4.2.2.2 Trajectoire résultant de l'intersection de deux droites

L'intersection entre deux droites peut classiquement se résoudre en formulant les équations de ces droites et en résolvant un système d'équations. Par exemple, si $D_1(t)$ est définie par les points $A(t)$ et $B(t)$ et $D_2(t)$ par les points $C(t)$ et $D(t)$, $D_1(t)$ et $D_2(t)$ s'intersectent si on trouve un temps t_x tel que le point d'intersection $M(t_x)$ vérifie :

$$\begin{cases} M(t_x) = A(t_x) + \alpha_1 \vec{AB}(t_x) \\ M(t_x) = C(t_x) + \alpha_2 \vec{CD}(t_x) \end{cases}$$

Pour résoudre l'équation, il faut déterminer α_1 , α_2 et le temps t_x de collision. Cependant, cela pose des problèmes dans le cas où les droites sont colinéaires. Une autre méthode consiste à utiliser les coordonnées homogènes. On peut ainsi associer les droites $D_1(t)$ et $D_2(t)$ à des plans passant par elles et l'origine (cf. figure 4.9). Pour définir ces plans, on recherche leur normale \vec{n}_1 et \vec{n}_2 . Chaque normale est déduite à partir de deux points distincts de chaque droite en effectuant le produit vectoriel de leurs coordonnées homogènes. Tout point p d'une des droites D_i vérifie $\vec{n}_i \cdot p = 0$. Un point p d'intersection entre D_1 et D_2 va donc respecter l'équation $\vec{n}_1 \cdot p = \vec{n}_2 \cdot p = 0$, en d'autres termes, le vecteur \vec{Op} (où O est le centre du repère) est orthogonal

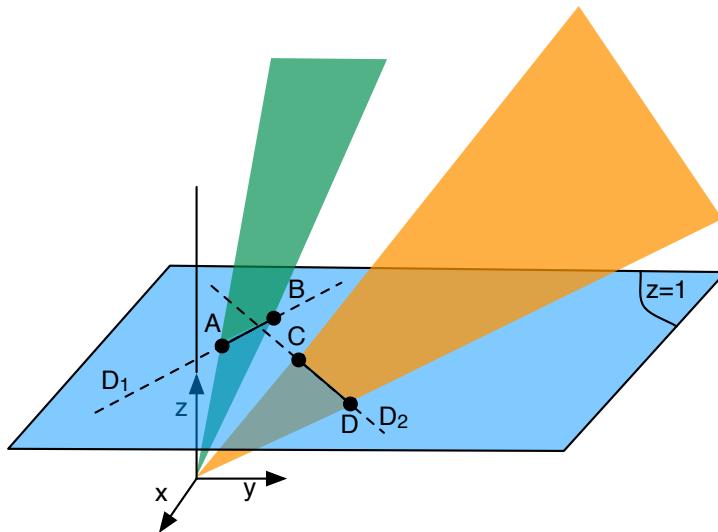


FIG. 4.9 – Intersection de deux droites D_1 et D_2 en utilisant les coordonnées homogènes.

à n_1 et n_2 . Un tel vecteur est immédiatement obtenu par le produit vectoriel $n_1 \wedge n_2$. Si la coordonnée de $p_z = 0$, alors les droites sont colinéaires. Sinon, le point p dans \mathbb{R}^2 est de coordonnées $(p_x/p_z, p_y/p_z)$. En résumé :

$$\vec{p}(t) = \left(\begin{pmatrix} A_x(t) \\ A_y(t) \\ 1 \end{pmatrix} \wedge \begin{pmatrix} B_x(t) \\ B_y(t) \\ 1 \end{pmatrix} \right) \wedge \left(\begin{pmatrix} C_x(t) \\ C_y(t) \\ 1 \end{pmatrix} \wedge \begin{pmatrix} D_x(t) \\ D_y(t) \\ 1 \end{pmatrix} \right)$$

donc dans $\mathbb{R}^2 : p(t) = \begin{pmatrix} \frac{p_x(t)}{p_z(t)} \\ \frac{p_y(t)}{p_z(t)} \end{pmatrix}$ pour $p_z(t) \neq 0$

4.2.3 Implantation d'un prototype 2D

L'instanciation 2D de ce modèle a été prototypée à l'aide d'une surcouche du noyau du modeleur Moka, un modeleur géométrique 3D à base topologique. Son noyau est à base de 3-g-cartes et comporte de nombreuses opérations topologiques et géométriques. Notre modèle étant en 2D, nous ignorons simplement les liaisons α_3 utiles seulement en 3D, ce qui revient à appliquer l'opération identité sur chaque brin par α_3 ($\forall b \in B, b\alpha_3 = b$).

D'un point de vue pratique, le noyau de Moka est écrit en C++ et il est organisé en classes dont la hiérarchie sépare l'implantation purement topologique du plongement. L'implantation réalisée ici s'appuie sur les 3-cartes plongées, tout en redéfinissant le plongement. En effet, Moka est un modeleur 3D d'objets statiques, sans plongement temporel. À cet effet, le prototype utilise la bibliothèque C++ GiNaC (GiNaC = GiNaC is Not a CAS¹). GiNaC est une librairie de calcul symbolique, ce qui s'adapte bien à la représentation d'un plongement temporel dépendant d'un paramètre t avec des équations linéaires. Cependant, nous ne souhaitons faire aucune hypothèse de linéarité sur le mouvement des entités. Par exemple, le point d'intersection de deux segments dont les sommets se déplacent sur des trajectoires rectilignes uniformes (hypothèse de [58]) ne décrit généralement pas une trajectoire rectiligne uniforme.

¹Computer Algebra System

Les équations sont alors, dans le cas général, non linéaires. Pour traiter ce type de cas, nous utilisons Maxima, un CAS (Computer Algebra System) écrit en LISP. Son langage d'origine ne s'interface pas de manière intuitive avec les outils utilisés pour le prototype, nous l'utilisons donc en tant que programme externe, ce qui ralentit considérablement les résolutions d'équations. Cependant, nous ne prenons pas en compte pour le moment les aspects performance dans nos algorithmes. Des approches de résolution non linéaire permettront à terme de fournir des temps de calcul plus adaptés.

4.3 Conclusion

Dans ce chapitre, nous avons décrit un système d'animation topologique générant une animation de la structure tout en garantissant sa cohérence topologique. Ce système produit en outre un script des opérations appliquées au modèle et un historique des entités topologiques. Ce modèle contient trois composantes : une composante structurelle, une composante événementielle et une composante sémantique. La composante sémantique est en charge de la mémorisation du processus constructif, tandis que la composante événementielle garantit la cohérence topologique. Le système d'animation fait ainsi l'hypothèse que l'instant des collisions peut être détecté. Nous avons fait le choix de fournir, en entrée du système, un scénario haut-niveau. Ce dernier est spécifique à l'application visée. Cependant, si le système est dédié à une application, il faut également y définir les phénomènes haut-niveau que l'on souhaite simuler. Plus précisément, à chaque phénomène correspondent des événements initiaux et des règles de transformation à appliquer aux événements détectés. Le chapitre suivant propose une application de notre système d'animation au cas particulier de la géologie et montre comment le système s'adapte à cette application.

APPLICATION À LA GÉOLOGIE

Notre système d'animation a été présenté dans le chapitre précédent. Cependant, pour pouvoir reproduire l'évolution d'une structure, il nécessite une configuration préalable. En effet, le principe de transformer des phénomènes haut-niveau en une série d'opérations topologiques bas-niveau requiert de définir des événements initiaux, le comportement des entités topologiques et les transformations à appliquer pour chaque type d'événement et chaque phénomène. Nous devons donc choisir un domaine d'application donné où l'animation de structures en 2D présente un intérêt pour l'expert. Pour ce domaine, nous choisissons des phénomènes et configurons le système en conséquence. Cette mise en application nous permet ainsi de montrer que notre approche est effective et adaptable à une simulation donnée.

Le domaine choisi est la géologie. En effet, dans ce domaine, l'utilisation de la 2D est courante. Les sous-sols sont souvent étudiés avec une cartographie 2D. Les relevés de certains instruments de mesure fournissent des informations planaires. Nous considérons alors qu'une évolution purement 2D fournit déjà des résultats significatifs pour les géologues. Il nous faut ainsi déterminer un ensemble de phénomènes courants en géologie et les transcrire dans notre système. Ces phénomènes sont la sédimentation, l'érosion, l'apparition de failles et le glissement. Dans ce chapitre, nous commençons d'abord par examiner la problématique générale des géologues puis étudions successivement chacun des phénomènes choisis, c'est-à-dire leur définition, les événements initiaux du phénomène, les événements rencontrés en cours d'évolution et les modifications topologiques engendrées. Nous décrivons alors l'application développée à cet effet et les résultats d'animation obtenus en combinant les différents phénomènes étudiés.

5.1 Problématique

La géologie (du grec *gê* : terre et *logos* : étude) étudie la structure, la composition, l'histoire et l'évolution de la Terre. L'évolution de la Terre peut être décrite à travers l'évolution des couches géologiques qui la composent ainsi qu'au travers des processus internes et externes qui les modifient. Ces processus induisent des modifications structurelles qui causent des métamorphoses chimiques des couches et des migrations du sous-sol dues aux conditions

de température et de pression. L'étude de la topographie du sol fournit des informations sur la possibilité de couches réservoirs qui piègent et accumulent certains matériaux.

Connaître l'évolution de la structure du sous-sol permet de déterminer l'existence de couches géologiques ainsi que leur composition. Cette étude passe par la construction de modèles théoriques 2D ou 3D. Un modèle du sous-sol se base sur l'interprétation des relevés topographiques ou sismiques, des carottages, etc. par des géologues experts ou des logiciels métiers. À partir de ce modèle, les géologues peuvent émettre des hypothèses sur la genèse de telles structures ainsi que sur leur évolution ; ils peuvent alors prédire l'emplacement de certaines couches particulièrement recherchées (comportant par exemple des poches de gaz ou des réservoirs de matériau fossile).

Notre système d'animation se prête bien à la représentation de la structure du sous-sol et la reproduction des phénomènes géologiques principaux. En effet, le sous-sol est structuré à partir d'un ensemble de couches de formes et de propriétés différentes. Une couche géologique est formée de matériaux sédimentaires accumulés sur une aire de dépôt plus ou moins étendue durant un intervalle de temps défini. Une couche peut être fragmentée en une multitude de blocs non adjacents, ce qui rend la structure du sous-sol complexe. La modélisation d'une coupe du sous-sol peut être réalisée à l'aide d'une subdivision de l'espace 2D. Un premier problème d'échelle intervient ici : les dimensions d'une couche se mesurent du millimètre au kilomètre. Cette structure est le résultat d'évolutions mal maîtrisées (l'historique de l'évolution et toutes les conditions ne sont pas connus) reposant sur les hypothèses de structures et d'évolutions du sous-sol. Ces évolutions sont le résultat de phénomènes géologiques qui sont des processus affectant la surface du sol (pluie, vent, activités humaines, etc.) ou des mouvements de plaques tectoniques (cf. figure 5.1). Un deuxième problème d'échelle intervient ici : les phénomènes mis en jeu peuvent tout aussi bien durer une fraction de seconde comme se dérouler sur des milliers d'années. On peut prendre pour exemple une jeune chaîne de montagnes (à forte activité), située à Taiwan, qui est le résultat de la convergence de deux plaques dont la vitesse peut aller jusqu'à 10 cm / an entraînant une surrection (élévation) pouvant aller jusqu'à 3 cm / an. On peut aussi prendre l'exemple de la création d'une faille qui se déroule en moins d'une seconde suite à un tremblement de terre. Dans ce cas, on peut considérer que la création de la faille est instantanée par rapport à la création d'une chaîne de montagnes.

5.2 État de l'art en modélisation et simulation géologique

On distingue la modélisation dite analogique de la modélisation numérique. La modélisation analogique consiste à réaliser une maquette d'une scène géologique avec des matériaux (sable, pâte à modeler, etc.) ayant des propriétés similaires aux couches à modéliser. Un exemple de bac à sable est donné en figure 5.2. Cette figure représente l'état d'une chaîne de montagnes résultant du choc de deux plaques tectoniques. Pour obtenir l'animation d'un tel modèle, il faut construire chaque étape de la simulation mécanique et l'afficher. Changer un paramètre dans la simulation implique de recommencer toute l'expérience.

Un modèle numérique du sous-sol est généralement construit à partir de mesures brutes issues de sondages, forages, etc. Ces données bruitées doivent cependant être traitées et améliorées [3]. De nombreux logiciels industriels exploitent les mesures effectuées, par exemple GOCAD[30, 49], GSI3D et RML[65, 27] mais ces logiciels ne génèrent pas d'animation de l'évolution du sous-sol.

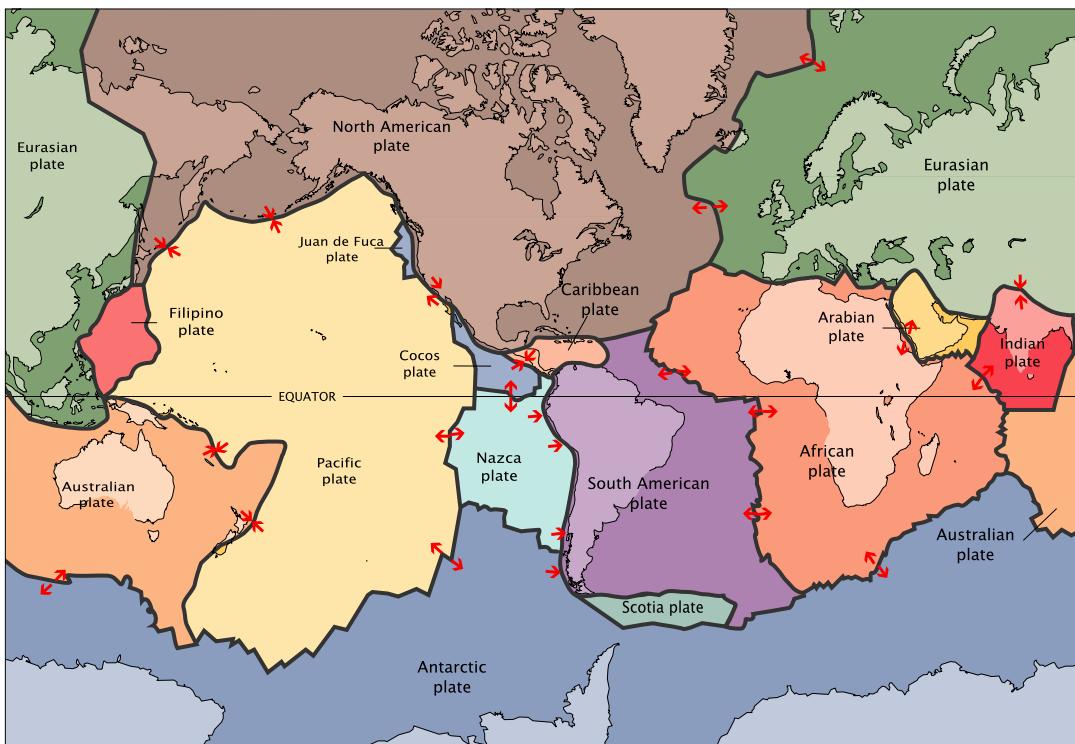


FIG. 5.1 – Carte des plaques tectoniques terrestres. Les flèches indiquent les mouvements relatifs de chaque plaque. (Carte extraite de wikipedia)

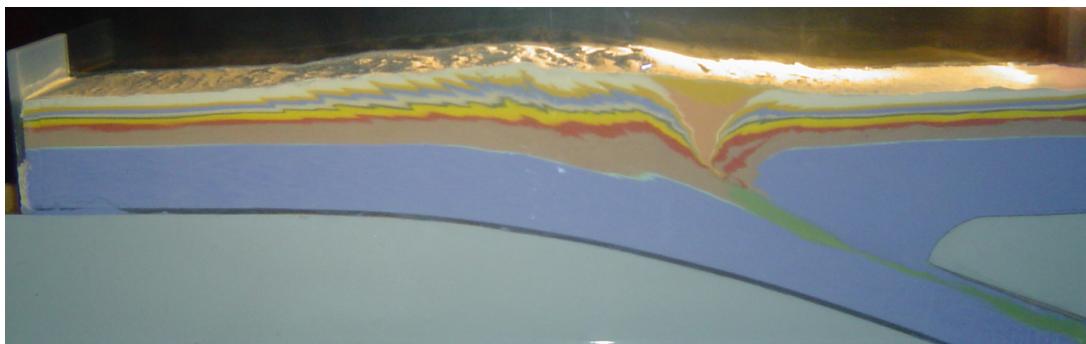


FIG. 5.2 – Simulation analogique : utilisation d'un bac à sable par Jacques Malavienne pour simuler la formation d'une montagne (Cité des Sciences à Paris).

Schneider [68] représente le sous-sol par un ensemble de surfaces qui s'intersectent. Ces surfaces correspondent aux *horizons* et aux *failles* qui délimitent les blocs géologiques. Ce calcul d'intersections s'effectue en utilisant la bibliothèque CAS.CADE sur un plongement en surfaces paramétriques. Dans cette approche, les horizons et les failles traversent la scène de part en part. Après calcul des intersections entre les surfaces paramétriques, Schneider utilise une extension du modèle des n-g-cartes conçue pour représenter la topologie des blocs géologiques et pour relier topologiquement les blocs appartenant aux mêmes couches géologiques, même si ces blocs sont géométriquement disjoints.

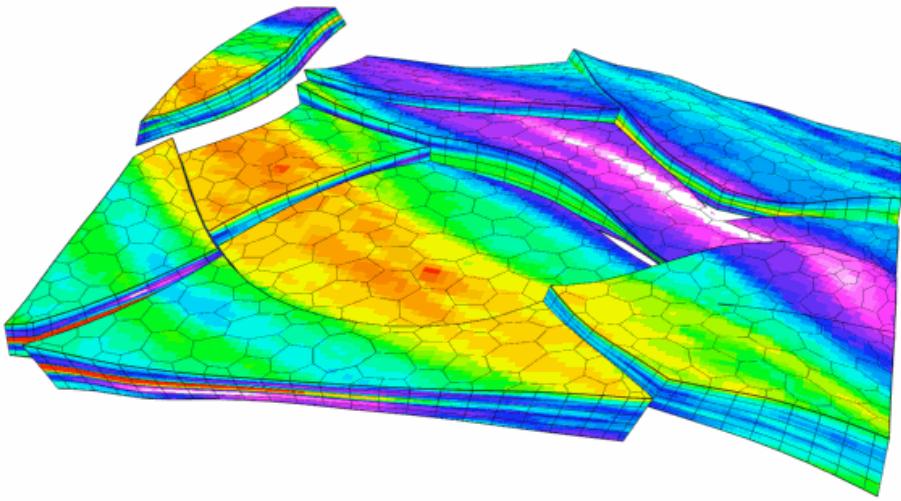


FIG. 5.3 – Exemple de sous-sol modélisé avec Gocad (image provenant du site officiel de Gocad).

Une autre approche pour l’intersection est proposée par Guiard [32]. Elle consiste à utiliser un plongement linéaire ; les surfaces paramétriques sont alors modélisées par subdivisions de surfaces composées de faces planes aux arêtes rectilignes. Les failles et les horizons peuvent s’intersecter sans forcément couper un bloc géologique, ce qui permet par exemple de représenter des failles pendantes (i.e. des failles dont au moins une extrémité s’arrête à l’intérieur d’un bloc). Cette méthode reprend le modèle topologique des n-g-cartes étendu [68]. Les approches de Schneider et Guiard sont statiques et ne prennent pas en compte l’historique de formation des couches géologiques et leur évolution.

Perrin[55] propose une description des causes et effets de l’évolution des couches géologiques, définie par une syntaxe qui permet de formaliser des phénomènes géologiques et leur succession au cours du temps. Ses règles assurent la consistance géologique d’un modèle 3D. Elles sont intégrées dans un *Schéma d’Évolution Géologique* (SEG), décrivant une interprétation du modèle à construire. Il se présente sous la forme d’un graphe acyclique orienté. Les nœuds de ce graphe représentent des surfaces délimitant les blocs géologiques ou des sous-SEG correspondant à un niveau de détail plus fin (par exemple, une faille au niveau n du SEG peut représenter un réseau de failles décrites au niveau $n+1$). Les arcs représentent soit une relation chronologique (par exemple, une faille est « antérieure à » une autre faille) ; soit une relation topologique basée sur les propriétés géologiques des surfaces qui s’intersectent (par exemple, une faille « coupe » un horizon). Les règles de syntaxe géologique permettent de préciser de quelle manière les nœuds du graphe interagissent. Les informations transportées par le SEG permettent donc de définir une chronologie des événements mais elles sont insuffisantes pour représenter une animation d’évolution de couches géologiques. En effet, les données temporelles précises ne sont pas présentes et la chronologie fournie par le SEG ne contient pas de durée. De plus, les liens de cause à effet entre les événements décrits sont inexistant : ce modèle ne permet pas, par exemple, de déterminer comment une couche a été créée ou comment sa forme a évolué.

En résumé, les représentations courantes se contentent principalement de modéliser et de structurer un sous-sol statique. Or, le géologue se base généralement sur des hypothèses d’évo-

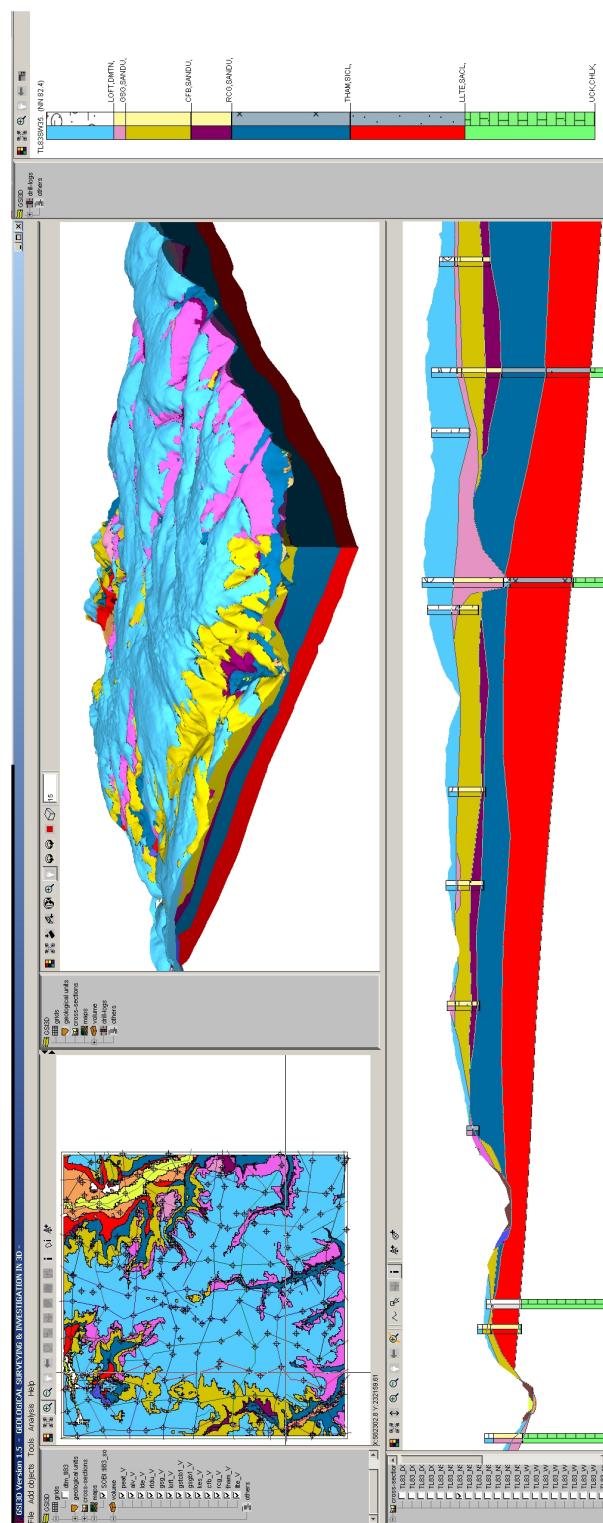


FIG. 5.4 – Interface de GSI3D (image créée par le « British Geological Survey »).

lutions du sous-sol pour en déduire sa structure et sa composition, il peut ainsi rechercher la présence de couches particulières, de réservoirs, etc.

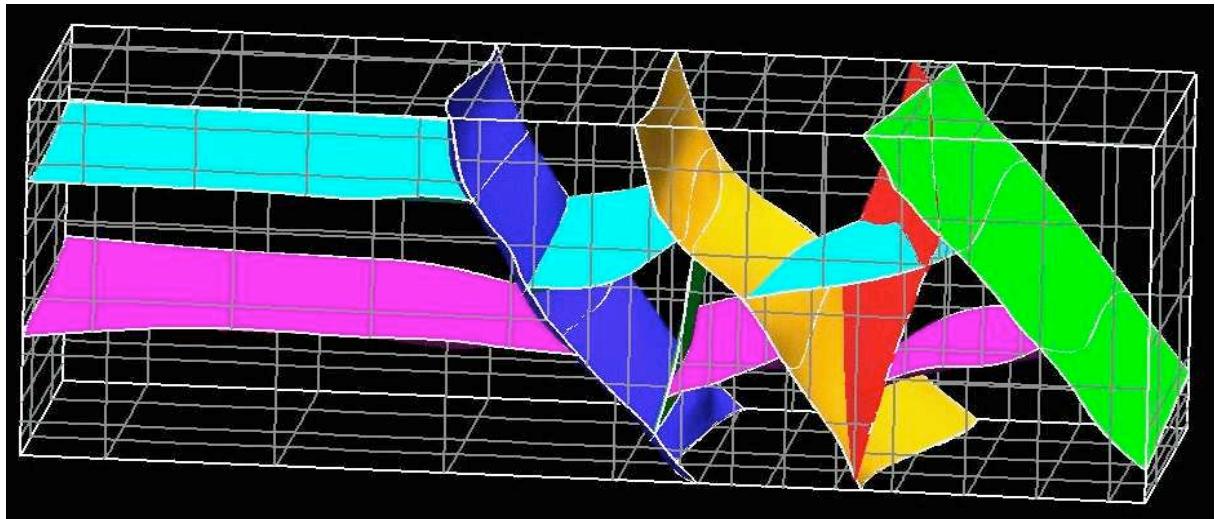


FIG. 5.5 – Modèle 3d du sous-sol construit avec des surfaces paramétriques par Schneider (Image extraite de [68]).

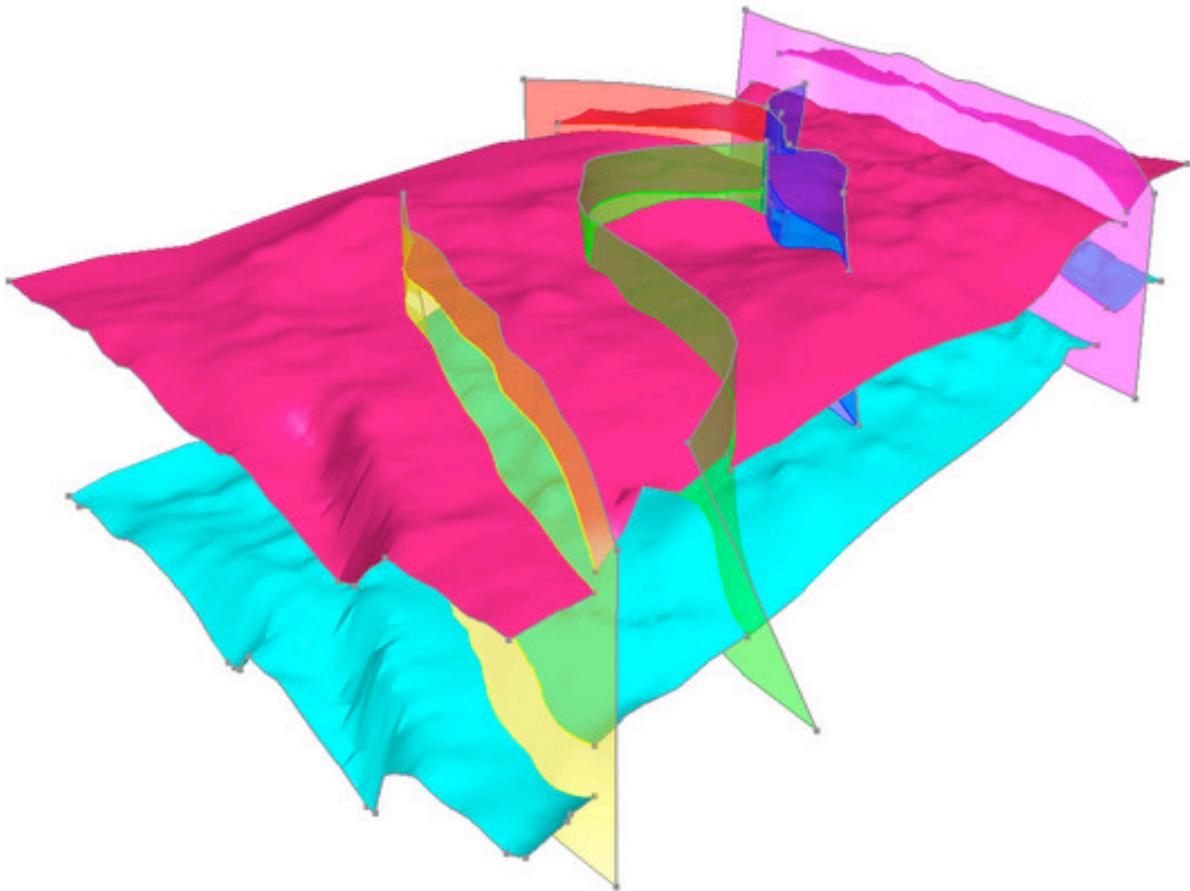


FIG. 5.6 – Modèle 3d du sous-sol construit avec des surfaces subdivisées par Guiard (Image extraite de [32]).

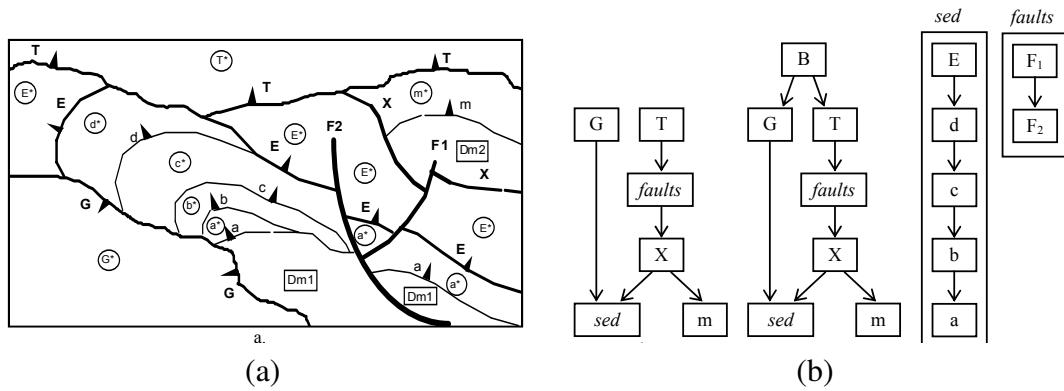


FIG. 5.7 – Exemple de SEG (extrait de[12]).

5.3 Modélisation 2D des phénomènes géologiques

Nous avons choisi quatre phénomènes, suffisamment représentatifs des évolutions géologiques et assez complexes pour montrer les possibilités de notre système : la sédimentation, l'érosion, la création de faille et le glissement de blocs géologiques. Ces quatre phénomènes et leurs combinaisons engendrent de nombreuses modifications géométriques et topologiques et permettent de représenter une large palette d'évolutions tout en restant simples à modéliser. Après avoir présenté ces phénomènes, nous décrivons comment ils sont intégrés dans le modèle d'animation. Pour chaque phénomène, la méthode générale consiste à définir les événements initiaux. Ces événements permettent au système de débuter l'animation du phénomène. D'autres événements apparaissent durant l'exécution du phénomène et doivent également être définis. Les événements entraînant des modifications topologiques et/ou de plongements doivent être définis grâce à des motifs et des algorithmes de transformation. Chaque événement exige au moins un motif et sa transformation associée.

5.3.1 Modèle de sédimentation

La sédimentation est l'ensemble des processus de surfaces par lesquels des particules en suspension se déposent. Selon les principes de stratigraphie énoncés par Stenon (1669), les sédiments se déposent en couches à peu près horizontales.

Nous considérons un modèle de sédimentation suivant une déposition strictement horizontale (nous pourrions cependant utiliser des formes non strictement horizontales en suréchantillonnant la surface de sédimentation comme nous le faisons pour l'érosion en section 5.3.2). La scène considérée représente une coupe transversale du sous-sol. La sédimentation débute par le remplissage des zones les plus basses. Ensuite, le niveau de sédimentation augmente et si deux couches issues d'une même sédimentation se rencontrent, elles fusionnent.

5.3.1.1 Modélisation topologique

La sédimentation peut être modélisée par une arête dont les extrémités glissent le long du bord du bloc en création. Pour réaliser une sédimentation, nous allons donc simplement créer une arête dont les extrémités vont être identifiées à deux nouveaux sommets insérés dans la zone de sédimentation. Pour simuler l'augmentation du volume de la couche, les sommets vont

glisser le long de la zone de sédimentation (cf. figure 5.8). Dans ce cadre, faire glisser un sommet revient à lui donner une position qui suit le profil de la zone de sédimentation dans une direction donnée. Le profil est constitué d'un certain nombre d'arêtes correspondant à la structure de la scène et/ou à la forme du profil (dans le cas du 0-plongement). Les deux nouveaux sommets ont donc découpé deux arêtes et se déplacent sur le profil de ces arêtes. Dans le cadre du 0-plongement, il suffit de calculer le plongement du sommet en mouvement. Dans le cadre du 1-plongement, il convient de calculer le plongement des n arêtes incidentes aux deux nouveaux sommets tout en assurant qu'ils respectent bien le profil de la scène et qu'ils sont en adéquation avec la structure topologique à tout instant. Le 1-plongement de l'arête de sédimentation peut facilement être déduit en le décrivant comme l'interpolation linéaire des coordonnées de ses extrémités, ce qui décrit le segment entre ses extrémités.

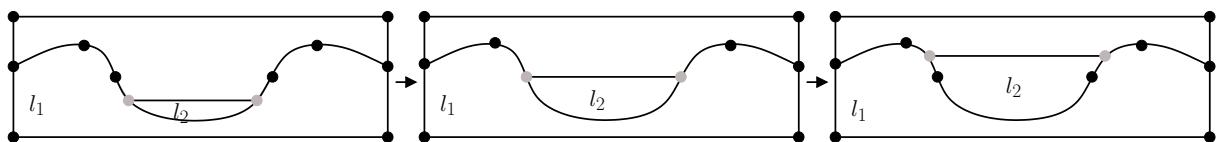


FIG. 5.8 – Sédimentation : création d'une arête permettant de créer la couche l_2 , augmentation du volume par glissement des sommets extrémités.

5.3.1.2 Modélisation à partir d'un scénario

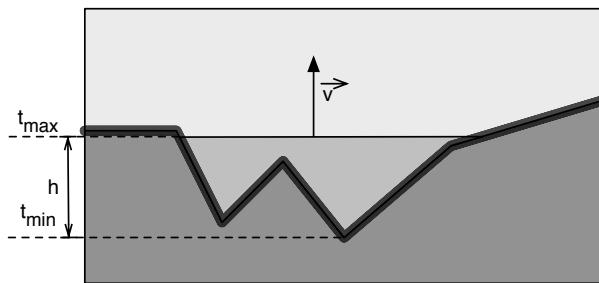


FIG. 5.9 – Paramètres du modèle de sédimentation (\vec{v} est le vecteur représentant la vitesse de sédimentation).

La sédimentation est définie par cinq paramètres (Figure 5.9) : un préfixe pour désigner les différents morceaux créés par la sédimentation, une zone de sédimentation délimitée par deux sommets (*begin_interface*, *position_on_interface*) et (*end_interface*, *position_on_interface*), une date de début *t_min*, une date de fin *t_max* et une hauteur *h*. Ce modèle simplifié considère que la vitesse de sédimentation est constante (ie. : $v = \frac{h}{t_{max} - t_{min}}$).

Ce phénomène peut être décrit dans le scénario d'évolutions géologiques à l'aide de la fonction suivante :

Listing 5.1 – Description du phénomène de sédimentation

```

1  actionSedimentation(t_min, t_max - t_min,
2    vertical_velocity, "prefix_interfaces",
3    "begin_interface", position_on_interface, — sommet 1
4    coverage_direction,
5    "end_interface", position_on_interface); — sommet 2

```

Les paramètres $t_{max} - t_{min}$ et $vertical_velocity$ correspondent respectivement à la durée du phénomène et à la vitesse de montée verticale de la couche en création. Ces paramètres permettent de déduire facilement la hauteur de sédimentation. Le paramètre "coverage_direction" permet, avec l'aide des sommets 1 et 2, de définir une zone de sédimentation en discriminant les deux zones possibles entre ces sommets. Ces deux zones correspondent aux deux sens possibles de parcours d'une face. Si ce paramètre est positionné sur *forward*, le système parcourt la face de sédimentation dans le sens *début* vers *fin* de l'arête (dépendant de la désignation, voir section 3.2.3.1). Le paramètre *prefix_interfaces* permet au système de savoir comment désigner les arêtes provenant du phénomène de sédimentation.

Le traitement de ce phénomène par le système débute par la recherche des minima locaux dans la zone de sédimentation (Figure 5.10(a)). Ensuite, les dates de début de remplissage de ces minima sont calculées. Les événements de *création d'interfaces* associés à chaque minimum sont ajoutés à la file d'événements.

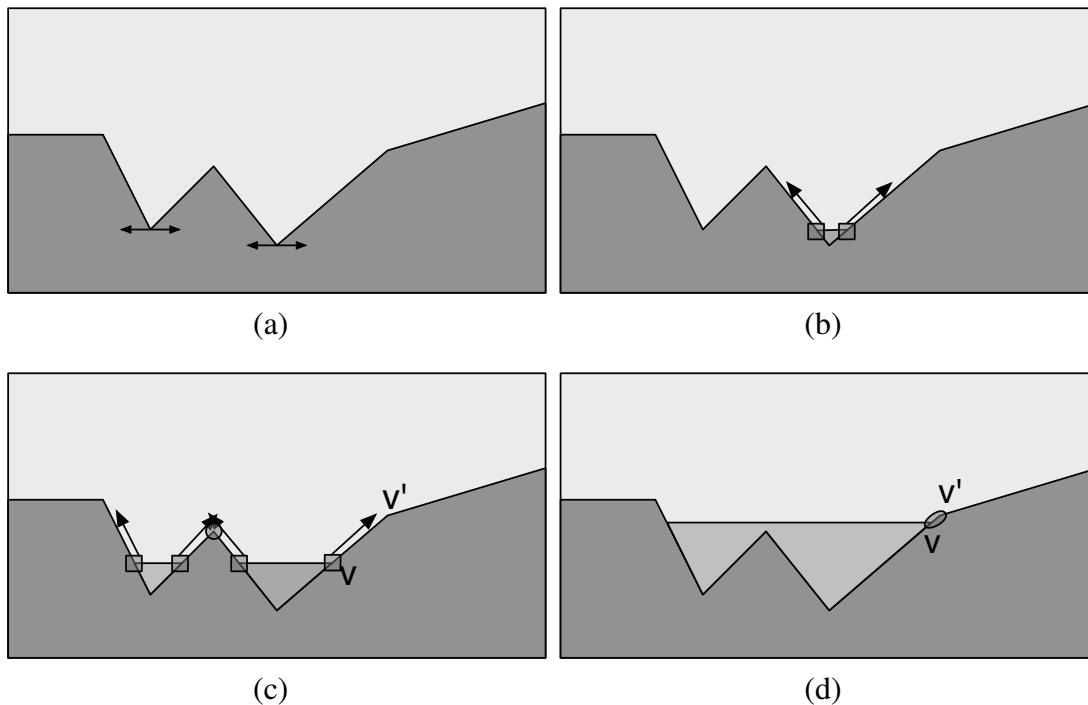


FIG. 5.10 – Étapes d'une sédimentation. (a) Recherche des minima locaux. (b) Création de la première face et glissement de ses extrémités le long des parois de la couche. (c) Création d'une nouvelle face et glissement de ses extrémités. Le sommet v se déplace vers le sommet v' . (d) Fusion des deux faces avec glissement des extrémités. Après cette étape, v et v' rentreront en collision, puis v glissera sur v' et continuera son déplacement le long de la couche.

Le traitement de la création d'interface consiste à insérer une arête dans la face de sédimentation au niveau de la zone du minimum courant (cf. figure 5.10(b)). Les extrémités d'arêtes sont plongées par une fonction d'interpolation entre les coordonnées du minimum local et le sommet suivant, pour glisser le long du bord de la couche tout en tenant compte de la vitesse de sédimentation.

Lors du glissement, les sommets peuvent rencontrer de nouveaux sommets. Ces événements sont considérés comme des collisions sommet / sommet liés par une arête, et leur date est calculée et ajoutée à la file. Ces événements peuvent être traités de deux manières suivant le contexte

local. Soit les deux sommets en collision sont issus du même phénomène de sédimentation (cf. figure 5.10(c)) et sont fusionnés, ce qui entraîne une fusion de faces (processus décrit à la figure 3.15(c)) ; soit un des sommets provient de la sédimentation et l'autre appartient à la couche du sous-sol (cf. figure 5.10(d)) et dans ce cas, le premier sommet v glisse sur le second v' et continue son chemin sur l'arête suivante (cf. figure 4.3). Le glissement de sommet peut entraîner d'autres événements de collisions sommet / sommet, qui sont donc insérés dans la file. Ce processus est répété jusqu'à la fin du phénomène de sédimentation.

Pour distinguer les deux cas de collisions, nous avons besoin de deux motifs de collision sommet / sommet : l'un pour la fusion de faces (cf. figure 5.11(a)) et l'autre pour le glissement de sommet (cf. figure 5.11(b)). Dans les deux cas, la taille s de l'arête reliant les sommets impliqués dans la collision est nulle. Le premier motif correspond à la détection de deux arêtes consécutives de taille nulle (qui n'appartiennent pas au processus de sédimentation et sont associées au prédictat " !Sed") incidentes à deux arêtes, qui sont issues de ce processus (et donc associées au prédictat "Sed"). L'algorithme de transformation consiste à contracter les deux arêtes de taille nulle (n_2, n_3) et de fusionner les deux interfaces ($sed01_0, sed01_1$), entraînant la fusion des faces de sédimentation. Le second motif correspond à la détection d'une arête de taille nulle (n_2). L'algorithme de transformation consiste à contracter cette arête et à insérer un nouveau sommet v sur l'arête suivante n_1 , coupant cette dernière en deux nouvelles arêtes respectivement désignées n_{1_0} et n_{1_1} . Ensuite, l'extrémité de l'interface de sédimentation est déconnectée puis identifiée à v .

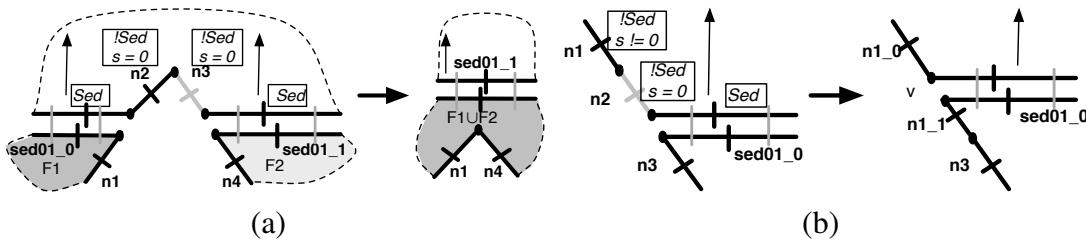


FIG. 5.11 – Motifs de collisions pour la sédimentation (les collisions sont représentées en gris, les prédictats sont encadrés et les flèches verticales représentent le sens du mouvement des faces de sédimentation). (a) Fusion des faces F_1 et F_2 issues du même processus de sédimentation. (b) Glissement d'un sommet sur un autre sommet. Les lignes en pointillés représentent un lien $(\alpha_1 \alpha_0)^* \alpha_1$ entre deux brins.

5.3.2 Modèle d'érosion

L'érosion est l'ensemble des processus de dégradation et de transformation du relief.

5.3.2.1 Modélisation topologique

L'érosion peut être modélisée par déformation du chemin d'arêtes représentant la zone érodée (cf. figure 5.12). Dans le cas du 0-plongement, on peut subdiviser ce chemin pour obtenir un mouvement plus réaliste. La déformation consiste ensuite à affecter des mouvements aux sommets du chemin d'arêtes. Dans le cas du 1-plongement, il faut s'assurer que pour chaque paire d'arêtes successives, l'extrémité de l'une soit confondue avec le début de l'autre, afin d'assurer la continuité géométrique et conserver la propriété de plongement.

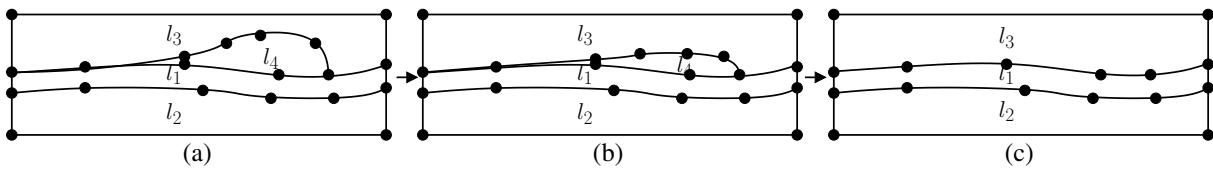


FIG. 5.12 – Érosion de la couche l_4 . (b) Diminution du volume de l_4 . (c) Suppression de l’interface entre l_3 et l_4

Durant l’érosion, des collisions entre le chemin d’arêtes et le reste du modèle peuvent se produire. Dans ce cas, il peut être nécessaire d’insérer un sommet au niveau de la collision et/ou supprimer des arêtes touchées par le chemin pour conserver la cohérence topologique. Comme pour la sédimentation, lorsque l’érosion touche une partie du modèle, les sommets de contact entre le chemin et le reste du modèle adoptent un comportement de glissement. Les arêtes de longueur nulle doivent être contractées.

5.3.2.2 Modélisation à partir d'un scénario

Notre approche permet de traiter toutes les formes que peut prendre le phénomène d’érosion. Nous choisissons ici de considérer un profil en forme de cuve pour simplifier le paramétrage de ce phénomène. Pour obtenir un profil en forme de cuve, les arêtes érodées sont ré-échantillonnées suivant un pas défini par l’utilisateur. L’érosion débute par la déformation de la surface délimitée par deux arêtes. Si la surface d’érosion touche une autre surface, celle-ci est modifiée en suivant le profil de la surface d’érosion (cf. figure 5.13).

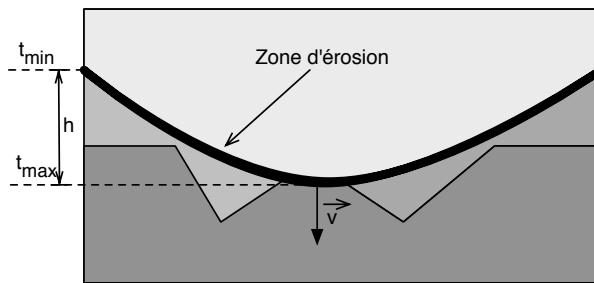


FIG. 5.13 – Paramètres du modèle d’érosion.

L’érosion est définie par quatre paramètres : la zone d’érosion délimitée par deux sommets (dans la figure 5.13, la zone est délimitée par les bords de la scène), une date de début t_min , une date de fin t_max et une hauteur h .

Ce phénomène peut être décrit dans le scénario d’évolutions géologiques à l’aide de la fonction suivante :

Listing 5.2 – Description du phénomène d’érosion

```

1 actionErosion(t_min , t_max - t_min ,
2   "begin_interface" , position_on_interface , — sommet 1
3   coverage_direction ,
4   "end_interface" , position_on_interface , — sommet 2
5   vertical_velocity ,
6   "prefix_interfaces" ,
7   sampling);
```

Les paramètres $t_{max}-t_{min}$, *vertical_velocity*, *coverage_direction* et *prefix_interfaces* ont le même sens que ceux de la fonction *actionSedimentation*. Le paramètre *sampling* correspond au pas de découpage de la surface érodée permettant d'approcher le profil d'érosion.

Considérant une implantation se basant sur un 0-plongement, le traitement de ce phénomène débute par le sur-échantillonage de l'interface d'érosion (cf. figure 5.14(a)). Pour cela, il suffit d'insérer de nouveaux sommets dont le nombre correspond au paramètre *sampling* de *actionErosion* sur l'arête considérée et de mettre à jour le plongement géométrique. Les trajectoires de ces sommets sont calculées pour donner à l'interface la forme d'une cuve. Les événements initiaux consistent donc à effectuer plusieurs insertions de sommets (si nécessaire) et à mettre à jour leur plongement.

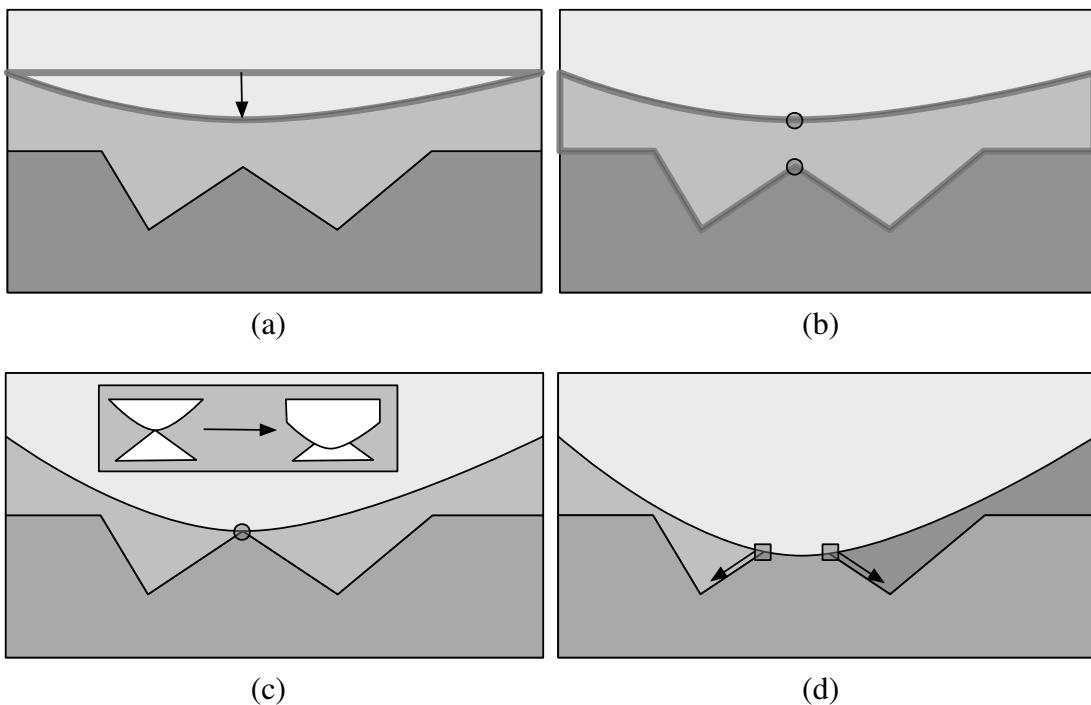


FIG. 5.14 – Étapes d'une érosion. (a) Sur-échantillonage de la surface d'érosion et mise à jour du plongement des sommets. (b) Recherche de la date de collision la plus proche : les sommets mis en jeu dans cette collision sont symbolisés par des disques. (c) À l'instant de la collision, traitement de l'événement suivant le contexte : ici, la surface d'érosion est prioritaire sur le reste de la scène. (d) La surface d'érosion a érodé le pic : une nouvelle face est créée et les points de contact glissent le long des couches géologiques.

Durant le mouvement des sommets, des collisions peuvent se produire entre la surface érodée et les couches du sous-sol. Après le calcul des trajectoires des sommets de l'interface érodée, les dates de ces collisions sont prédites et les événements associés sont ajoutés à la file. Le traitement de ces événements dépend du contexte local (cf. figure 5.14(c)). Soit l'interface d'érosion est prioritaire sur les autres interfaces du modèle et érode donc ces interfaces, soit l'interface d'érosion est éclatée en plusieurs morceaux après la collision. Cette notion de priorité est décrite explicitement dans [55] et adaptée à la modélisation de structures géologiques dans [11]. Durant la déformation de l'interface d'érosion, les sommets peuvent rentrer en collision soit avec d'autres sommets, soit avec des arêtes. Les Figures 5.15 et 5.16 décrivent les collisions sommet / arête possibles et leur traitement dépendant de la nature des entités mises en cause (les surfaces d'érosion ont comme prédicat "Ero", les autres " !Ero"). Les collisions sommet / som-

met, où les sommets ne sont pas liés par une arête, sont traités comme des cas particuliers de ceux montrés en figure 5.16. Le plongement géométrique de chaque nouveau sommet est calculé de telle manière que le mouvement de ce sommet corresponde à un glissement le long des arêtes initiales.

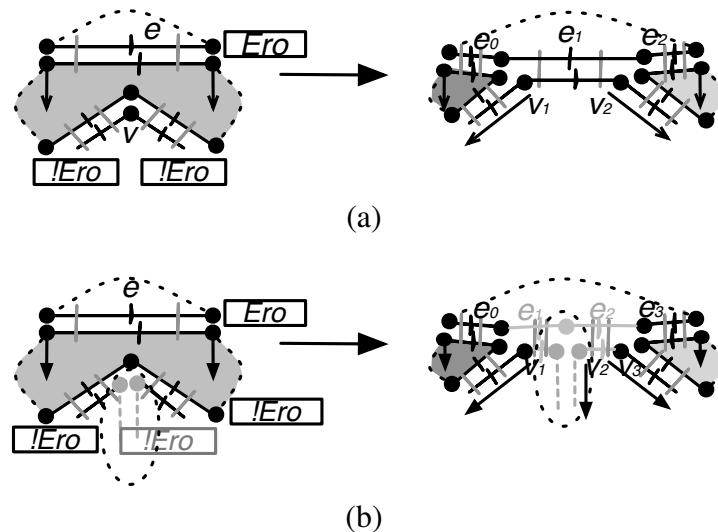


FIG. 5.15 – Traitement de la collision entre une arête d'érosion et un sommet du modèle au niveau topologique et géométrique. a) Intersection d'une arête avec un sommet de degré 2. b) Cas général d'intersection avec un sommet de degré > 2 .

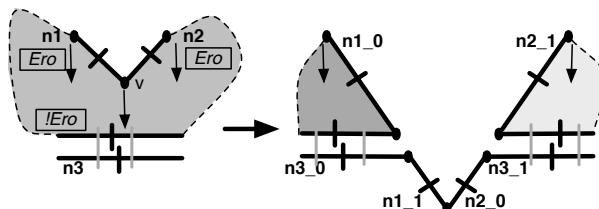


FIG. 5.16 – Traitement de la collision d'un sommet de la surface d'érosion avec une arête du sous-sol (le degré du sommet n'a ici aucune incidence).

La figure 5.15 illustre le traitement d'une collision entre une arête e appartenant à la surface d'érosion et un sommet v de la scène. Si le degré de v est égal à 2 (cf. figure 5.15(a)), deux nouveaux sommets v_1, v_2 sont insérés sur e , les arêtes incidentes à v sont déconnectées, puis liées aux sommets précédemment insérés. Le plongement de ces sommets est mis à jour pour correspondre à un glissement le long de l'arête e . Sinon, si le degré de v est $n > 2$, n nouveaux sommets sont insérés et toutes les arêtes incidentes à v d'abord déconnectées puis reliées aux nouveaux sommets en suivant l'ordre donné par l'orbite sommet.

La figure 5.16 illustre le traitement d'une collision entre un sommet v issu d'une interface d'érosion et une arête $n3$ du sous-sol. Le traitement débute par l'insertion d'un sommet sur l'arête $n3$, ce qui découpe cette dernière en $n3_0$ et $n3_1$ (dans le cas d'une collision sommet / sommet où les sommets ne font pas partie de la même arête, nous omettons simplement l'insertion d'un sommet, tout en poursuivant le reste du traitement). Ensuite, un sommet est inséré sur les deux arêtes $n1$ et $n2$ incidentes à v et qui partagent la face contenant $n3$ et v : $n1$ et $n2$ sont donc respectivement divisées en $n1_0$ et $n1_1$ d'une part, et en $n2_0$ et $n2_1$ d'autre part. Puis les relations topologiques entre ces arêtes sont mises à jour de manière à déconnecter $n1_1$

(resp. $n2_0$) de $n1_0$ (resp. $n2_1$) et à la relier à $n3_0$ (resp. $n3_1$). Enfin, nous déconnectons les deux nouvelles arêtes de $n3$ que nous identifions avec les sommets insérés sur $n1$ et $n2$.

Notons qu'après le traitement de la collision, les sommets de contact de la surface d'érosion peuvent glisser le long de la surface d'érosion (cf. figure 5.14(d)). Cette évolution est similaire au glissement de sommets de long d'arêtes décrit dans le cas de la sédimentation. L'événement de collision sommet / sommet peut donc être généré dans le cas de l'érosion. La figure 5.17 montre les motifs reconnaissant ces collisions et leur traitement. Le motif 5.17(a) est similaire au motif du glissement de sommet utilisé dans le processus de sédimentation (cf. figure 5.11(b)) pour la partie topologique mais diffère pour la partie plongement géométrique, qui correspond aux coordonnées temporelles de l'intersection de l'arête $n3$ avec l'arête $n1$. Le motif 5.17(b) représente la disparition de l'arête $n2$ par sa contraction en sommet. Un nouveau plongement géométrique est affecté à ce sommet, correspondant à l'intersection des lignes portant $n3$ et $n1 - n4$.

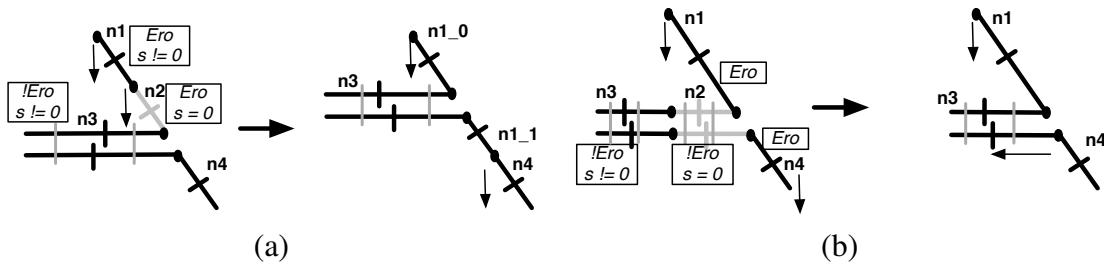


FIG. 5.17 – Motifs de collisions de deux sommets liés par une arête dans le cas de l'érosion. (a) Glissement de sommets. (b) Contraction d'arête.

5.3.3 Modèle de création de failles

Une faille est une zone de rupture le long de laquelle se produit généralement une déformation cisailante. Une faille peut débuter et s'arrêter n'importe où dans la scène. Elle peut donc diviser un bloc entièrement ou non. Les entités géologiques se trouvant de part et d'autre d'une faille peuvent alors glisser. La création de faille et le glissement sont dûs aux forces exercées par des contraintes tectoniques.

Dans [11, 68, 32], les modèles géologiques 3D représentent les failles par des surfaces délimitant des volumes 3D. Par analogie, nous avons choisi d'utiliser une représentation par ligne polygonale (cf. figure 5.19) en 2D. Une ligne polygonale est donc subdivisée en un ensemble de segments identifiés (seg_0 , seg_1 sur la figure 5.19). La croissance d'une ligne polygonale débute par la croissance de son premier segment. Une fois terminée, elle se poursuit par la croissance du deuxième segment, ainsi de suite. Les segments croissants peuvent rentrer en collision avec les entités de la scène, ce qui se traduit par une décomposition en sous-segments. La désignation de ces sous-segments est réalisée par l'utilisation d'un préfixe donné par l'utilisateur et d'un identifiant numérique ($prefixe_0$ et $prefixe_1$ issus de seg_0 , $prefixe_2$ issu de seg_1 sur la figure 5.19). Pour identifier la ligne polygonale, un parent commun à toutes les arêtes créées est inséré dans l'arbre de désignation.

5.3.3.1 Modélisation topologique

La création de failles est modélisée par la croissance d'un ensemble d'arêtes. Une arête peut débuter sa croissance soit sur le bord d'une autre arête, soit dans une face (cf. figure 5.18). Dans le premier cas, on insère un sommet sur l'arête du bord que l'on identifie avec une extrémité de l'arête en croissance. Si l'arête débute sa croissance dans une face, il faut la relier au modèle pour représenter l'information d'inclusion de l'arête dans la face. Pour cela, on peut utiliser plusieurs méthodes comme les arbres d'inclusion ou l'utilisation d'arêtes fictives. Dans le cadre de notre modèle, nous imposons que chaque carte comporte une seule composante connexe : l'arbre d'inclusion ne convient donc pas. Nous utilisons une arête fictive pour raccorder l'arête représentant une faille au bord de la face (cf. figure 5.18c). Cette arête fictive doit être supprimée dès qu'elle n'est plus nécessaire (i.e. l'arête a atteint le bord de la face) ou bien sa position doit être mise à jour au cas où sa face incidente est divisée.

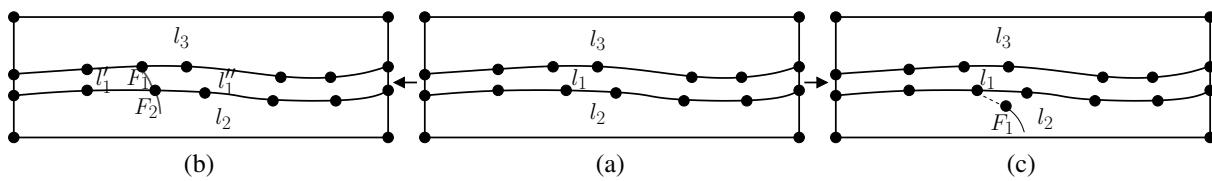


FIG. 5.18 – (a) La couche l_1 se trouve entre les couches l_2 et l_3 . (b) La faille F_1 , issue d'un sommet de l'interface ($l_1 - l_3$) coupe entièrement l_1 qui se scinde en l'_1 et l''_1 puis se prolonge en F_2 . (c) La faille F_1 débute à l'intérieur de l_2 , elle est donc reliée par une arête fictive (en pointillés) à un sommet de l'interface ($l_1 - l_2$).

5.3.3.2 Modélisation à partir d'un scénario

La création de failles est définie par quatre paramètres : la forme de la ligne polygonale, une date de début t_min , une date de fin t_max et un préfixe pour les arêtes créées.

Ce phénomène peut être décrit dans le scénario d'évolutions géologiques à l'aide de la fonction suivante :

Listing 5.3 – Description du phénomène de faille

```
1 actionFaille(t_min, t_max - t_min,
  "prefix_interfaces",
  shape);
```

Le paramètre $t_max - t_min$ correspond à la durée du phénomène. Le paramètre *prefix_interface* permet au système de savoir comment désigner les arêtes provenant du phénomène de création de failles. La figure 5.19 illustre ces paramètres.

Le traitement de ce phénomène débute par la recherche de la zone où débute la faille. Cette recherche consiste à localiser l'entité la plus proche (au sens géométrique) du premier sommet de la ligne polygonale. Ensuite, deux cas peuvent se présenter pour traiter la création de la faille (cf. figure 5.20) : la croissance débute soit sur un bord (sommet ou arête), soit à l'intérieur d'une face. Dans le premier cas, si la faille débute sur une arête, un sommet est inséré sur l'arête concernée. Ensuite, ce sommet est identifié avec l'extrémité de la première arête de la faille. Enfin, cette arête grossit en fonction des paramètres définis dans *action_faille* (cf. figure 5.21). Dans le second cas, pour maintenir la contrainte de connexité de chaque 2-g-carte de notre modèle, il faut insérer une arête dite fictive afin de représenter l'information d'inclusion

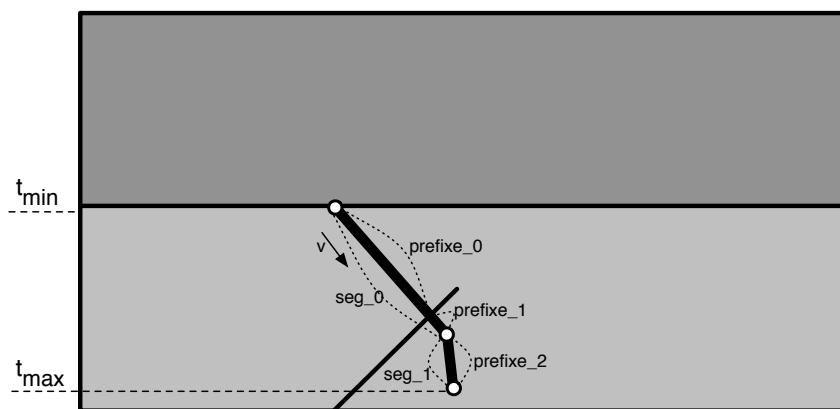


FIG. 5.19 – Paramètres du modèle de création de failles.

de l’arête dans la face, et obtenir ainsi une seule composante connexe par n -g-carte. Quel que soit le cas considéré, la croissance des arêtes est ensuite traitée de manière homogène.

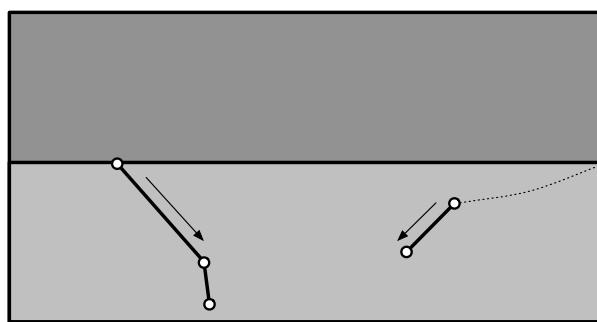


FIG. 5.20 – Différents cas de positions de débuts de failles (en pointillé : arête fictive).

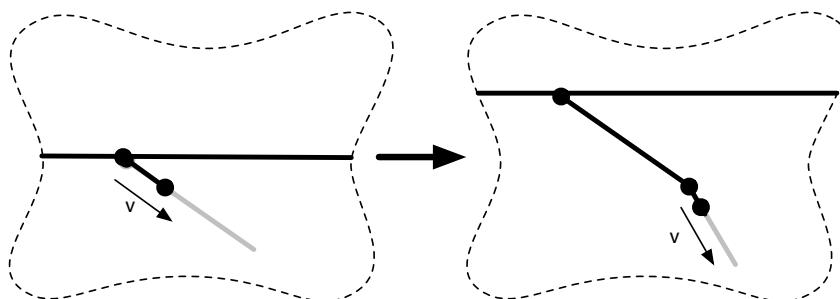


FIG. 5.21 – Croissance d’un ensemble de segments.

Si la ligne polygonale est constituée de plusieurs segments et que le segment en cours de croissance n’est pas le dernier constituant la faille, la date de fin de croissance du segment courant est ajoutée à la file d’événements. Cet événement permet d’indiquer au système qu’il doit vérifier si le phénomène est toujours en cours. Dans un tel cas, le système reprend le traitement de création de segment décrit plus haut.

Comme pour les autres phénomènes géologiques, les événements de collisions sont calculés. Nous considérons qu’il existe deux types de collisions, soit « bloquante », soit « traversante ». Dans le premier cas, la faille traverse l’interface rencontrée, ce qui se traduit par l’identification de l’extrémité croissante de la faille avec l’interface et l’insertion d’une nouvelle arête de

l'autre côté de l'interface pour continuer le processus de croissance (cf. figure 5.22). Dans le second cas (collision faille/faille par exemple), la croissance de la faille s'arrête au niveau de l'entité rencontrée (la figure 5.26(c) illustre ce cas, suivi d'un glissement), ce qui se traduit par l'identification de l'extrémité croissante de la faille avec le bord du bloc courant, avec si besoin l'insertion d'un sommet sur ce bord (cf. figures 4.4(b) et 4.5(b)).

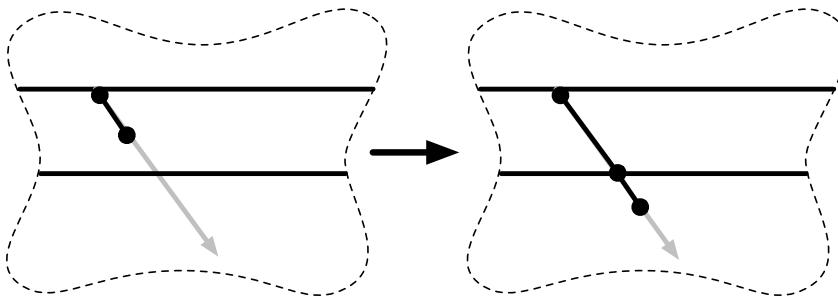


FIG. 5.22 – Intersection d'une faille avec une interface.

5.3.4 Modèle de glissement

Les failles étant causées par des tensions exercées sur les différentes couches, ces tensions entraînent le glissement de blocs le long de ces failles. On distingue plusieurs types de couples faille/glissement : les failles normales, les failles inverses et les failles décrochantes. Une faille normale est le résultat d'une extension (cf. figure 5.23(a)). Les blocs situés au-dessus de la faille descendent par rapport aux blocs du dessous. Une faille inverse est le résultat d'une compression (cf. figure 5.23(b)). Les blocs situés au-dessus de la faille montent par rapport aux blocs du dessous. Une faille décrochante est une faille d'inclinaison verticale le long de laquelle glissent horizontalement les blocs qu'elle délimite (cf. figure 5.23(c)).

5.3.4.1 Modélisation topologique

Le glissement est modélisé en changeant simplement le plongement des sommets composant les interfaces qui glissent. De la même façon que pour la sédimentation et l'érosion, les sommets extrémités des interfaces en mouvement doivent être éclatés si leur degré est supérieur à trois ou si les deux arêtes (ou l'arête représentant l'interface) incidentes au sommet ne sont pas colinéaires.

5.3.4.2 Modélisation à partir d'un scénario

Dans notre modèle, le glissement géologique est défini par sept paramètres : une date de début t_{min} , une date de fin t_{max} , la ligne polygonale composée d'arêtes à faire glisser, un pas de subdivision minimum pour représenter la déformation introduite par le glissement, une direction relative par rapport à la première arête de l'interface qui va glisser, une vitesse et une fonction de pondération de la vitesse *velocity_weighting_function* pour simuler une influence dépendant de la distance avec le point de glissement.

Ce phénomène peut être décrit dans le scénario d'évolutions géologiques à l'aide de la fonction suivante :

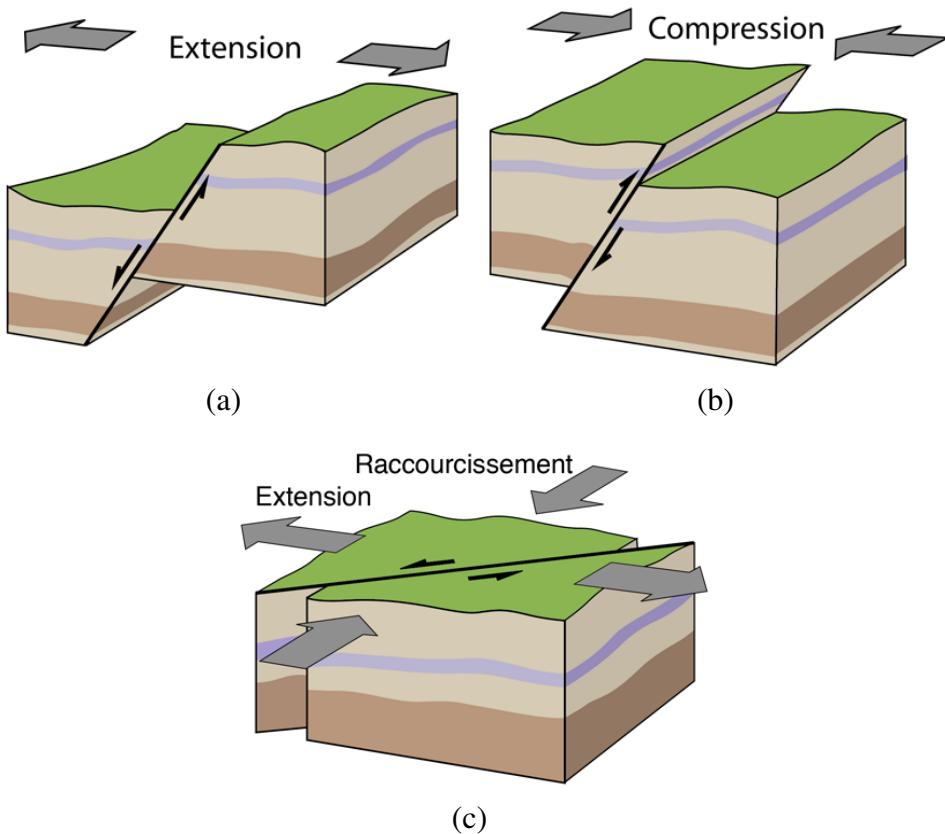


FIG. 5.23 – (a) Fissure normale. (b) Fissure inverse. (c) Fissure décrochante (Schéma de R. Lacassin 2005 - Wikipédia).

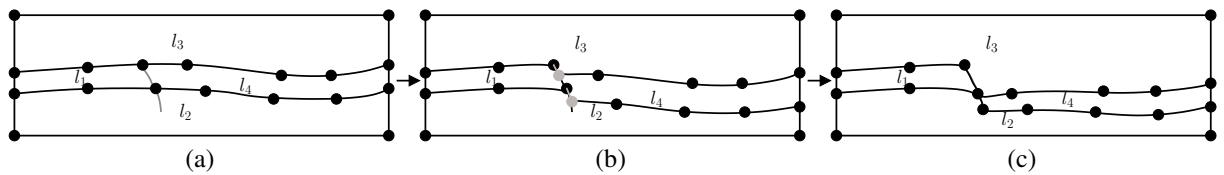


FIG. 5.24 – Glissement du bloc l_4 le long de la fissure centrale.

Listing 5.4 – Description du phénomène de glissement

```

1  actionGlissement(t_min , t_max - t_min ,
    interface_list , sampling ,
    relative_direction ,
    velocity ,
    velocity_weighting_function );
5

```

Les paramètres t_{max} , t_{min} , $sampling$ sont similaires à ceux vus précédemment. Le paramètre $interface_list$ correspond à la zone qui doit glisser. Le paramètre $relative_direction$ correspond à la direction relative à la première interface vers laquelle le système va effectuer le glissement.

Le traitement de ce phénomène débute par le sur-échantillonnage des arêtes composant la ligne polygonale qui va glisser. L'extrémité de la première arête est « éclatée » suivant une direction relative donnée par rapport à sa désignation (*left*, *right*), ce qui se traduit par l'insertion d'un sommet sur l'arête où s'effectue le glissement et l'identification de l'extrémité de la première arête sur ce sommet (cf. figure 5.26(d)). L'arête sur laquelle la ligne polygonale glisse

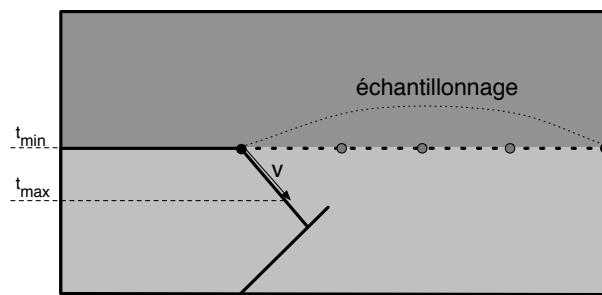


FIG. 5.25 – Paramètres du modèle de glissement.

donne la direction du mouvement du sommet courant. Cette direction et le paramètre vitesse permettent de déduire un vecteur vitesse et donc de calculer le mouvement des sommets de la ligne polygonale. Ces sommets sont plongés en fonction de leur position courante et du vecteur vitesse pondéré par la fonction fournie par l'utilisateur (cette fonction peut être quelconque). Dans notre exemple, nous utilisons une gaussienne pour obtenir une forme de glissement arrondie.

Le phénomène de glissement peut aboutir à tous les types de collisions : sommet / sommet non liés par une arête, sommet / sommet liés par une arête, et sommet / arête. Dans le cas d'une collision sommet / sommet non liés ou sommet / arête, la priorité est donnée au sommet / arête issue du glissement, ce qui se traduit par un comportement similaire à l'érosion et reprend donc ses mécanismes. En cas d'ambiguïté, c'est-à-dire quand les entités sont de même priorité et de même nature, le système affiche un message d'alerte à l'utilisateur (qui peut ainsi reconsidérer les priorités des interfaces). Dans le cas d'une collision sommet / sommet liés par une arête, i.e. pour une arête de longueur nulle, l'arête est contractée. Si cette arête correspondait à la zone de glissement, le nouveau sommet suivant est éclaté en suivant le schéma de la figure 4.3.

L'utilisation d'une date de début et de fin pour les phénomènes de création de faille et de glissement a un but de démonstration et de validation du système. En effet, à l'échelle géologique, ces phénomènes peuvent être considérés comme ponctuels.

5.4 Résultats

Cette section présente des résultats d'animation du sous-sol. Nous illustrons d'abord la cohérence topologique de la scène maintenue au cours de l'animation, en décrivant un scénario bas-niveau. Puis, nous proposons un exemple complet construit à partir d'un script haut-niveau appliqué à la géologie.

5.4.1 Modélisation topologique

Listing 5.5 – Script d'animation d'une scène géologique

```

1 require 'utils'
width = 50
height = 20
createEmbeddingFunctionFromFile = createEmbeddingFunctionFromFileCatmullRom
5 animation = CAnimation:new(gmv, width, height)

_____
8 — 0 -> 25 : déformation

```

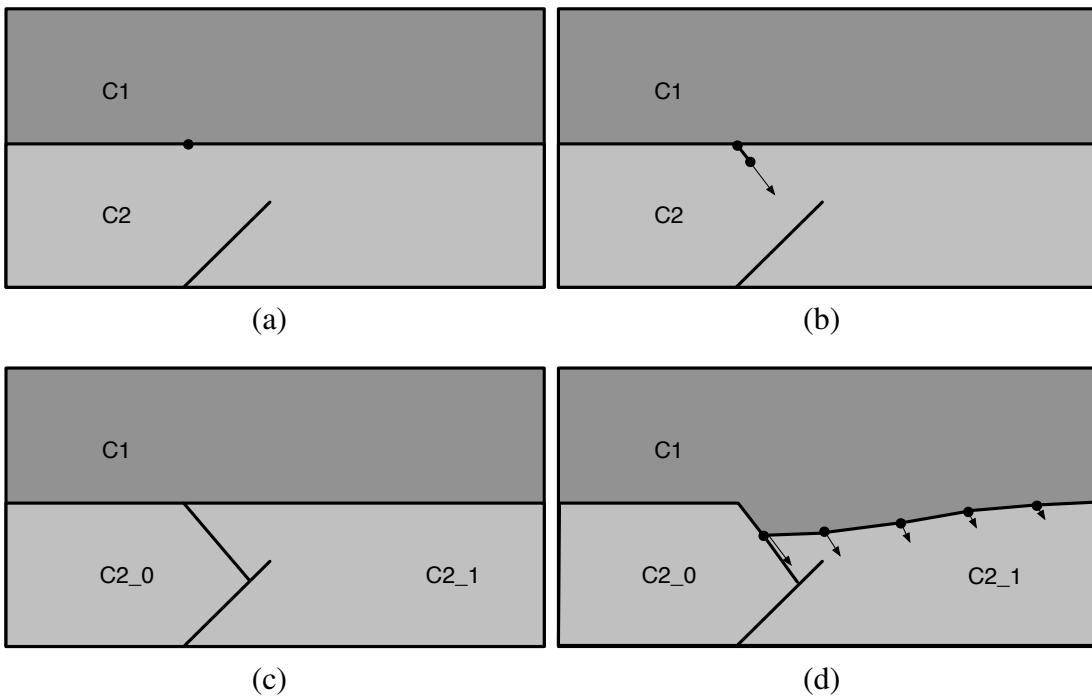


FIG. 5.26 – Étapes d'une création de faille suivie d'un glissement. (a) La couche C_2 contient une faille définie antérieurement. Début d'une nouvelle faille par l'insertion d'un sommet sur l'interface des couches C_1 et C_2 . (b) Création et croissance du premier segment de la faille. (c) Collision avec la faille initiale : la croissance s'arrête sur cette faille et C_2 est divisée en C_{2_0} et C_{2_1} . (d) Rééchantillonnage de l'interface entre C_1 et C_{2_1} et déformation de cette interface suivant la direction du glissement.

```

9 | dp = animation:addFirstDiscretePlane2d(0)
10 | — Création de la première interface géologique
11 | dp:createEdge("new_interface")
12 | dp:splitEdge({50}, "border_left" )
13 | dp:splitEdge({50}, "border_right" )
14 | dp:identification("border_left_1", begin, "new_interface", begin, left)
15 | dp:identification("border_right_1", begin, "new_interface", end, left)
16 | dp:setSymbolicEdgeEmbedding("new_interface",
17 |     morphoSimple1(createFixedInterpolatedLine(0, 15, 50, 15),
18 |         translate(scale(createEmbeddingFunctionFromFile("surf1_test.lua"), .1), 0, 5),
19 |         0, 25, false, false))

20 |

21 | — 25->40 : sédimentation
22 | dp2 = animation:copyLastDiscretePlane2d(25)
23 | dp2:splitEdge({33, 62}, "new_interface")
24 | dp2:splitEdgesPath({"new_interface_1"}, "sedimentation_a")
25 | dp2:setSymbolicEdgeEmbedding("sedimentation_a",
26 |     morphoSimple1(animation:findEdgeEmbedding("new_interface_1", 25),
27 |         translate(scale(createEmbeddingFunctionFromFile("surf2_test.lua"), .1), 0, 5),
28 |         25, 50, true, false))

29 |

30 | — 40->75 : sédimentation et création failles
31 | dp3 = animation:copyLastDiscretePlane2d(40)
32 | — arêtes fictives
33 | dp3:createEdge("dummy0");
34 | dp3:createEdge("dummy1");
35 | dp3:createEdge("dummy2");

36 | — arêtes représentant les bouts de failles
37 | dp3:createEdge("faille");
38 |
39 |
40 | — arêtes représentant les bouts de failles
41 | dp3:createEdge("faille");

```

```

dp3:createEdge("faille0");
dp3:createEdge("faille1");

45 — liaisons faille - arêtes fictives
46 dp3:identification("dummy0", end, "faille", begin, left)
dp3:identification("faille", end, "dummy1", begin, left)
dp3:identification("dummy1", end, "faille0", begin, left)
dp3:identification("faille0", end, "dummy2", begin, left)
50 dp3:identification("dummy2", end, "faille1", begin, left)

dp3:splitEdge({50}, "new_interface_1")
f = createInterpolatedLine("new_interface_1_0", 1, false, 30, 5)
dp3:setSymboleEdgeEmbedding("faille", dilatation(40, 75, changeRange(0, 1/3, f)));
dp3:setSymboleEdgeEmbedding("faille0", dilatation(40, 75, changeRange(1/3, 2/3, f)));
55 dp3:setSymboleEdgeEmbedding("faille1", dilatation(40, 75, changeRange(2/3, 1, f)));

dp3:identification("new_interface_1_0", end, "dummy0", begin, right)

60 —
61 — 75 -> 100
62 dp4 = animation:copyLastDiscretePlane2d(75)
dp4:contractEdge("dummy0");
dp4:contractEdge("dummy1");
65 dp4:contractEdge("dummy2");
dp4:contractEdge("faille0");
dp4:contractEdge("faille1");
dp4:setSymboleEdgeEmbedding("faille", f);
dp4:explodeVertex({ "new_interface_1_1", "faille" }, {"new_interface_1_0"}, "faille_0")
70 dp4:setSymboleEdgeEmbedding("faille", glissement(false, f, 75, 100))
dp4:setSymboleEdgeEmbedding("faille_0", glissement(true, f, 75, 100))

foldNewInterface_1_1 = animation:findEdgeEmbedding("new_interface_1_1", 75)
foldNewInterface_2 = animation:findEdgeEmbedding("new_interface_2", 75)
75 dp4:setSymboleEdgeEmbedding("new_interface_1_1", translateDynamic("new_interface_1_0",
    false, "faille", true, foldNewInterface_1_1))
dp4:setSymboleEdgeEmbedding("new_interface_2", translateDynamic("new_interface_1_0",
    false, "faille", true, foldNewInterface_2))

—
79 — 100 -> ...
80 dp5 = animation:copyLastDiscretePlane2d(100)
f = animation:findEdgeEmbedding("faille", 100)
local x1, y1 = f(0, 100)
local x2, y2 = foldNewInterface_1_1(0, 100)
local x, y = x1 - x2, y1 - y2
85 dp5:setSymboleEdgeEmbedding("new_interface_1_1", translate(foldNewInterface_1_1, x, y))
dp5:setSymboleEdgeEmbedding("new_interface_2", translate(foldNewInterface_2, x, y))
dp5:contractEdge("faille")

```

Après quelques expérimentations avec le script bas-niveau, nous avons décidé, dans le but de simplifier les calculs de collisions et leurs implantations, de nous restreindre au cas particulier du plongement sommet.

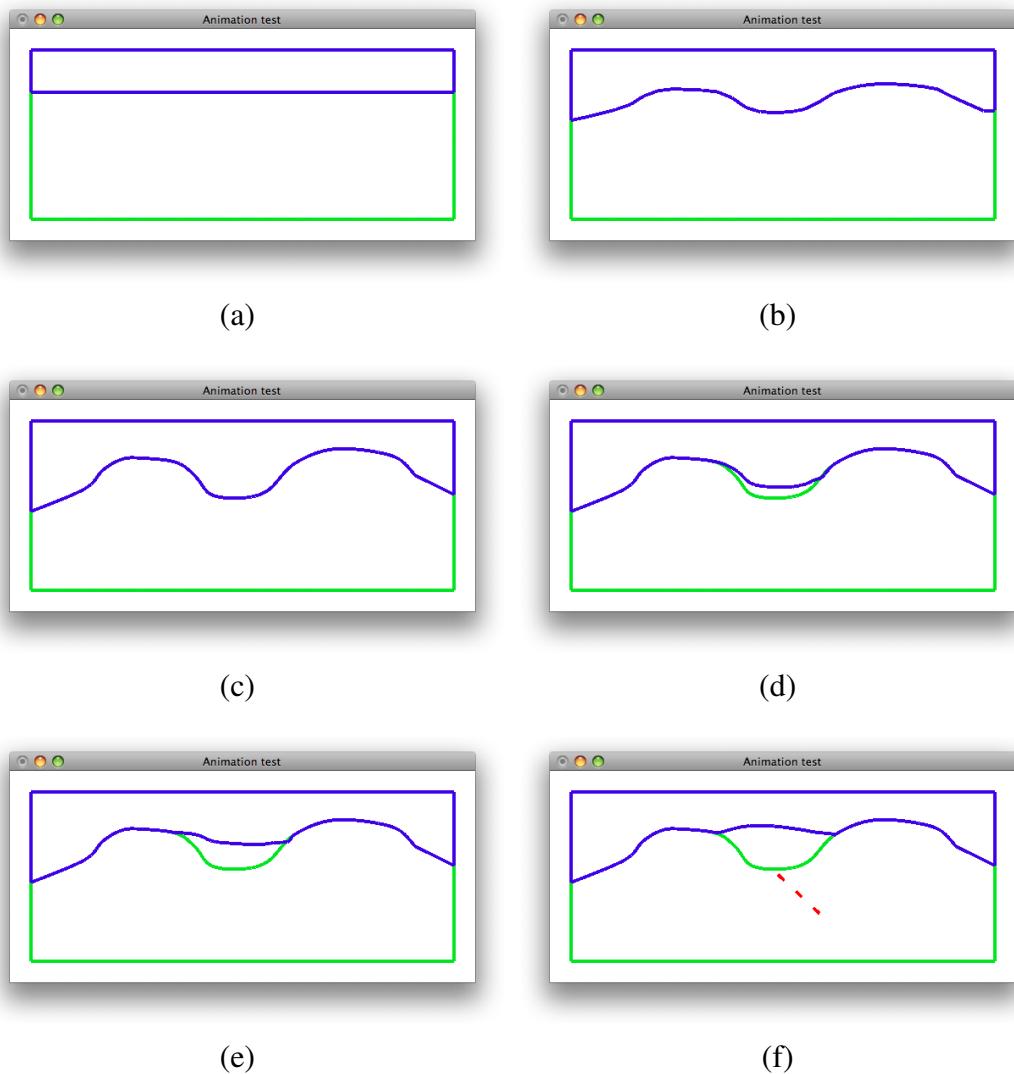


FIG. 5.27 – Images de l'animation d'une scène géologique.

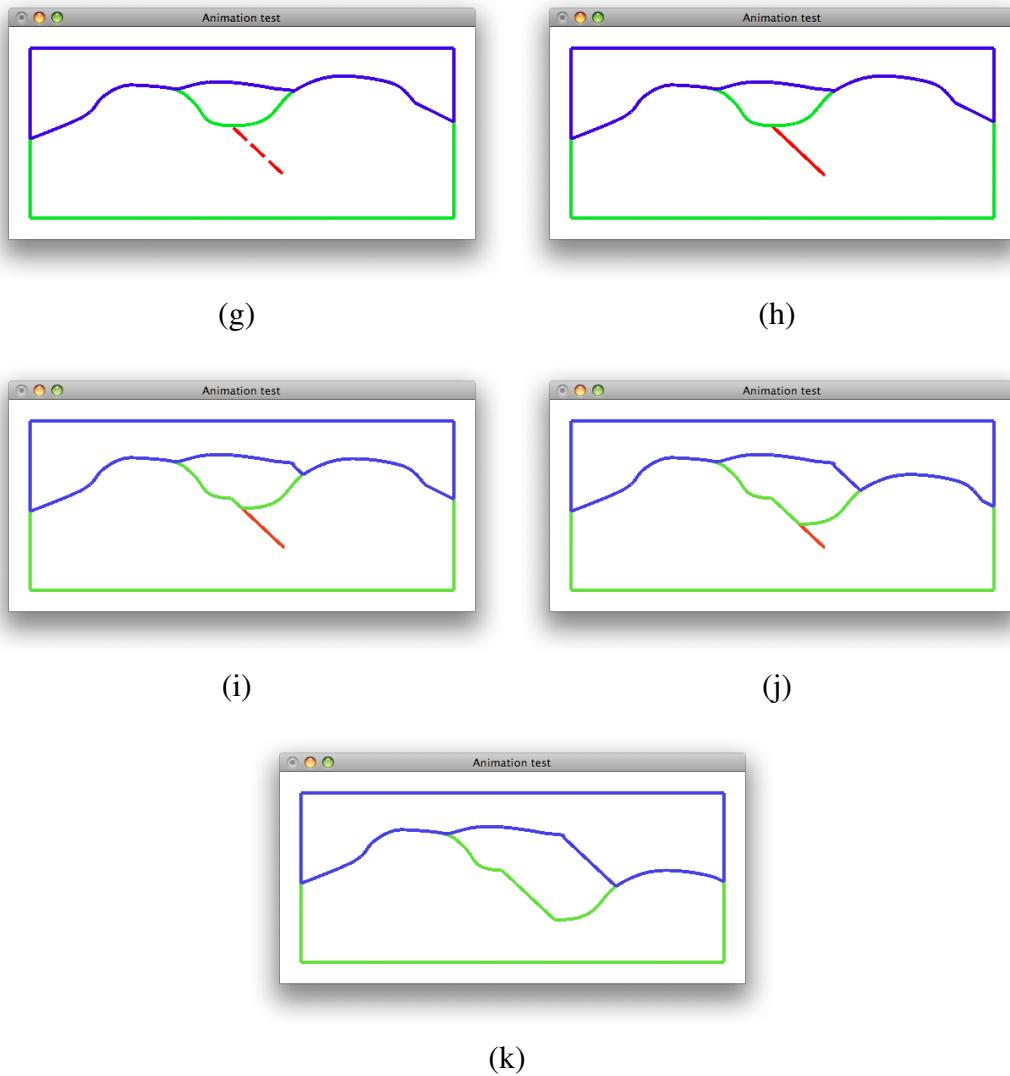


FIG. 5.27 – Images de l'animation d'une scène géologique.

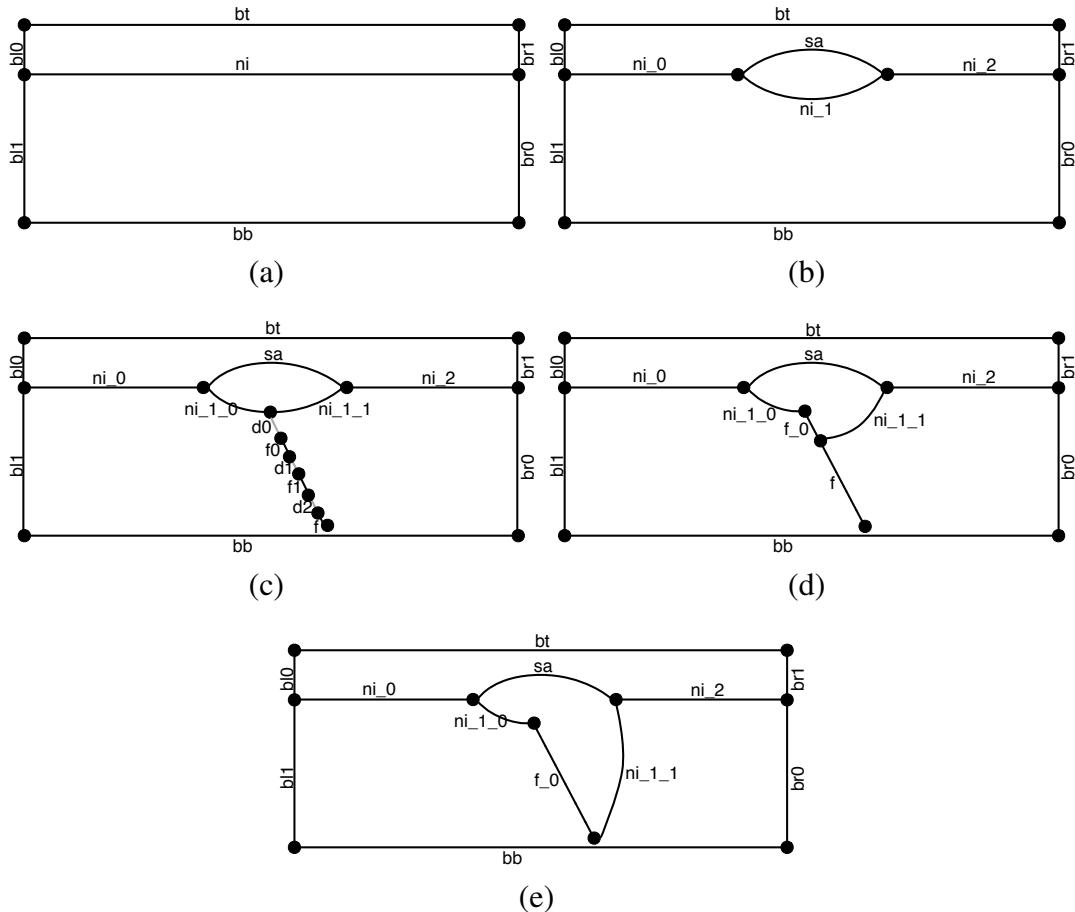


FIG. 5.28 – Images clefs créées par le script 5.5 vues de manière structurelle. (a) Scène de départ. (b) La sédimentation est réalisée par insertion d'une arête. (c) Création d'un réseau de failles qui se joignent par identification d'arêtes. (d) Les failles n'en font plus qu'une par contraction d'arêtes et un glissement se produit par éclatement d'un sommet. (e) Le sommet a terminé le glissement et il est identifié avec l'autre extrémité de la faille.

5.4.2 Modélisation à base de scénario générateur

Cette section est consacrée à la mise en pratique de notre modèle, allant du langage de scénario composé de fonctions géologiques (figure 4.1(a-b)) à l’animation graphique des entités, en passant par la génération du script bas-niveau (figure 4.1(c)) composé des opérations topologiques (figure 4.1(d)).

Le scénario 5.6 débute par la description de la construction d’un chenal via une succession de sédimentations et d’érosions. Il se termine par des créations simultanées de failles dont l’une conduit à un glissement.

Listing 5.6 – Ce scénario décrit l’état initial de la scène et son évolution au travers d’une succession de phénomènes géologiques : Sédimentation, Érosion, Création de failles et Glissement.

```

1 (1) scene = CSceneVertex :new(gmv, 8, 6);
2 scene :actionChenalCreation(0, "CI", "profil_7.v");
3 scene :actionSedimentation(10, 10,
4   3, "Sed01",
5     "border_left_0", left,
6       forward,
7       "border_right_1", left);
8 (4) scene :actionErosion(30, 10,
9   "Sed01_0", begin,
10    forward,
11    "Sed01_0", begin,
12    3,
13    "Ero01", 2);
14 (5) scene :actionSedimentation(40, 10,
15   .4, "Sed02",
16     "border_left_0_0", left,
17       forward,
18       "border_right_1_1", left);
19 (6) scene :actionFaille(60, 20,
20   "faille06",
21   {{1.843325, 1.92517}, {3.6, 2.5}},
22   0);
23 (7) scene :actionFaille(60, 20,
24   "faille01",
25   {{1,2}, {1.5,.3}},
26   0);
27 (8) scene :actionFaille(60, 20,
28   "faille05",
29   {{2.29413, 1.29558}, {2, 3}},
30   0);
31 (9) scene :actionFaille(60, 20,
32   "faille04",
33   {{2.55,1.55}, {2.5,3.5}, {1, 5}},
34   0);
35 (10) scene :actionFaille(60, 30,
36   "faille03",
37   {{2.55,1.54}, {2.5,.1}},
38   0);
39 (11) scene :actionFaille(60, 10,
40   "faille02",
41   {{.5,2}, {1.0,.5}},
42   0);
43 (12) scene :actionFaille(80, 20,
44   "faille07",
45   {{1,2},{7,3.5}},
46   0);
47 (13) scene :actionGlissement(110, 20,
48   {"Sed02_0_1"}, 1,
49   right, .1);

```

Ce scénario génère une animation illustrée par les images de la figure 5.29, extraites de notre lecteur d’animation. Ce logiciel est utilisé pour visualiser l’animation dans son intégralité et examiner la structure topologique à chaque instant. La partie gauche de la fenêtre présente

les entités regroupées dans une arborescence permettant de connaître leur origine et leurs descendants. Cette arborescence est mise à jour à chaque changement topologique. De plus, elle permet de sélectionner les entités par leur nom et de les surligner dans la fenêtre d'animation. La fenêtre d'animation permet de sélectionner des faces, des arêtes, des sommets et de leur donner des noms. Elle peut aussi afficher une vue éclatée de la structure topologique (cf. figure 5.30). La sélection, l'affichage et la navigation temporelle sont des outils d'analyse précieux pour comprendre l'évolution des entités.

Comme nous venons de le voir, tout scénario génère une suite d'événements. Ceux-ci sont analysés et traduits en un script d'évolutions de structures topologiques. Les paragraphes suivants détaillent le début du script du scénario 5.6 en se basant sur les figures 5.29 et 5.30 qui mettent en évidence les changements causés par le script haut-niveau sur la structure, la géométrie et l'arbre de désignation. Ensuite, ils décrivent brièvement la suite du script dont les étapes se trouvent sur la figure 5.30.

(1-2) Ce scénario crée une nouvelle animation pour laquelle chaque image est de dimension $8x6$, définit la première image-clef à la date 0 et positionne le « profil de » chenal $C1$ composé de dix arêtes désignées $C1_0$ à $C1_9$ (cf. figure 5.29(a)).

(3) Période de sédimentation entre les dates 10 et 20 avec une hauteur de 3. Cette sédimentation affecte toute la scène (du bord gauche au bord droit) et crée des interfaces disjointes dont la désignation est préfixée par $Sed01$. Ce phénomène a pour effet de créer deux nouvelles arêtes $Sed01_0$ et $Sed01_1$ dont les extrémités glissent le long du bord de $C1$ (cf. figure 5.29(b)). À l'étape de la figure 5.29(b), quatre sommets ont été insérés sur les quatre arêtes $C1_1$, $C1_2$, $C1_7$ et $C1_8$. Ces insertions aboutissent à la séparation de chaque arête en deux morceaux, (par exemple l'arête $C1_8$ est divisée en $C1_8_0$ et $C1_8_1$), ce qui se traduit par un sous-arbre dans l'arbre de désignation. Au niveau du pic central, $Sed01_0$ et $Sed01_1$ se rencontrent et fusionnent en $Sed01_0$ (cf. figure 5.29(c)). Le nom donné à l'arête résultante est dépendant de l'ordre de traitement des événements.

(4) Période d'érosion de la date 30 à 40 pour une hauteur de 3 appliquée à l'interface $Sed01_0$. Ce phénomène a pour effet d'échantillonner la surface d'érosion $Sed01_0$ (en $Sed01_0_0$, $Sed01_0_1$, $Sed01_0_2$ et $Sed01_0_3$) en affectant une trajectoire de déformation aux nouveaux sommets (cf. figure 5.29(d)). Le moteur de collision prédit une rencontre entre la surface d'érosion et le pic de $C1$. La collision est traitée en donnant la priorité à la surface d'érosion (cf. figure 5.29(e)). Ce traitement insère un sommet sur les arêtes $Sed01_0_1$ et $Sed01_0_2$, ce qui les scinde respectivement en $Sed01_0_1_0$ et $Sed01_0_1_1$, et en $Sed01_0_2_0$ et $Sed01_0_2_1$. L'érosion se poursuit et supprime les arêtes $C1_4$ et $C1_5$ (Figures 5.29(e) et 5.29(f)).

(5-12) Période de sédimentation (préfixe $Sed02$, aboutissant à l'arête $Sed02_0$) de la date 40 à 50 (cf. figure 5.31(b)) suivie d'une série de créations de failles ($faille01$ à $faille07$) (cf. figure 5.31(c)) qui se traduisent par une série de créations d'arêtes et de collisions (cf. figure 5.31(d)). À l'issue de ce processus, l'arête $Sed02_0$ a été découpée en $Sed02_0_0$ (partie gauche) et $Sed02_0_1$ (partie droite) par une faille. À la date 110, l'arête $Sed02_0_1$ glisse vers sa droite (cf. figures 5.31(e) à 5.31(e)).

Le listing 5.7 est un extrait du script bas-niveau généré par l'application et il correspond au début de la première sédimentation décrite dans le scénario 5.6, figure 5.29(b). Ligne 2, le script crée une nouvelle 2-g-carte, copie de la précédente, à la date 10. Il déclare ensuite deux nouvelles interfaces, $Sed01_0$ et $Sed01_1$ (Lig.3, 6), et les identifie aux sommets $C1_1/end$ (rappel : un sommet est désigné par un nom d'arête et sa position par rapport à elle), $C1_2/begin$

et $C1_7/end$, $C1_8/begin$ (Lig.4,5 et Lig.7,8). $C1_1/end$ et $C1_2/begin$ (respectivement $C1_7/end$ et $C1_8/begin$) désignent un même sommet. Les deux nouvelles faces issues de ces identifications sont donc constituées d'une arête seulement et sont donc dégénérées. Quatre événements de glissements de sommets sont créés, ce qui se traduit par quatre insertions de sommets (Lig.9,12,15,18) sur les arêtes $C1_1$, $C1_2$, $C1_3$ et $C1_4$, quatre déconnexions (Lig.10,13,16,19) et quatre identifications des extrémités des arêtes $Sed01_0$ et $Sed01_1$ aux nouveaux sommets (Lig.11, 14, 17, 20).

Listing 5.7 – Cet extrait de script bas-niveau représente les opérations topologiques du début de la première sédimentation du Listing 5.6.

```

1  [...]
dp1 = animation:copyLastDiscretePlane2d(10)
dp1:createEdge("Sed01_0", 0)
dp1:identification("C1_1", end, "Sed01_0", begin, left)
5 dp1:identification("C1_2", begin, "Sed01_0", end, left)
dp1:createEdge("Sed01_1", 0)
dp1:identification("C1_7", end, "Sed01_1", begin, left)
dp1:identification("C1_8", begin, "Sed01_1", end, left)
dp1:splitEdge({50}, "C1_1")
10 dp1:unlink("Sed01_0", begin)
dp1:identification("C1_1_1", begin, "Sed01_0", begin, left)
dp1:splitEdge({50}, "C1_2")
dp1:unlink("Sed01_0", end)
dp1:identification("C1_2_0", end, "Sed01_0", end, left)
15 dp1:splitEdge({50}, "C1_7")
dp1:unlink("Sed01_1", begin)
dp1:identification("C1_7_1", begin, "Sed01_1", begin, left)
dp1:splitEdge({50}, "C1_8")
dp1:unlink("Sed01_1", end)
20 dp1:identification("C1_8_0", end, "Sed01_1", end, left)
[...]
```

L’animation résultant du scénario 5.6 est composée de 2908 brins répartis sur 24 images-clefs. Actuellement, nous utilisons Maxima pour résoudre sans optimisation les équations non linéaires provenant du moteur de collisions. Le temps total de calcul est donc conséquent, ici de 14 mins 26s (sur un portable de type MacBook avec un processeur à 2.16 GHz). Cette durée descend à 25s si les solutions des équations non linéaires sont mises en cache et pourrait encore être améliorée en utilisant des bibliothèques de calculs et des techniques de détections de collisions optimisées.

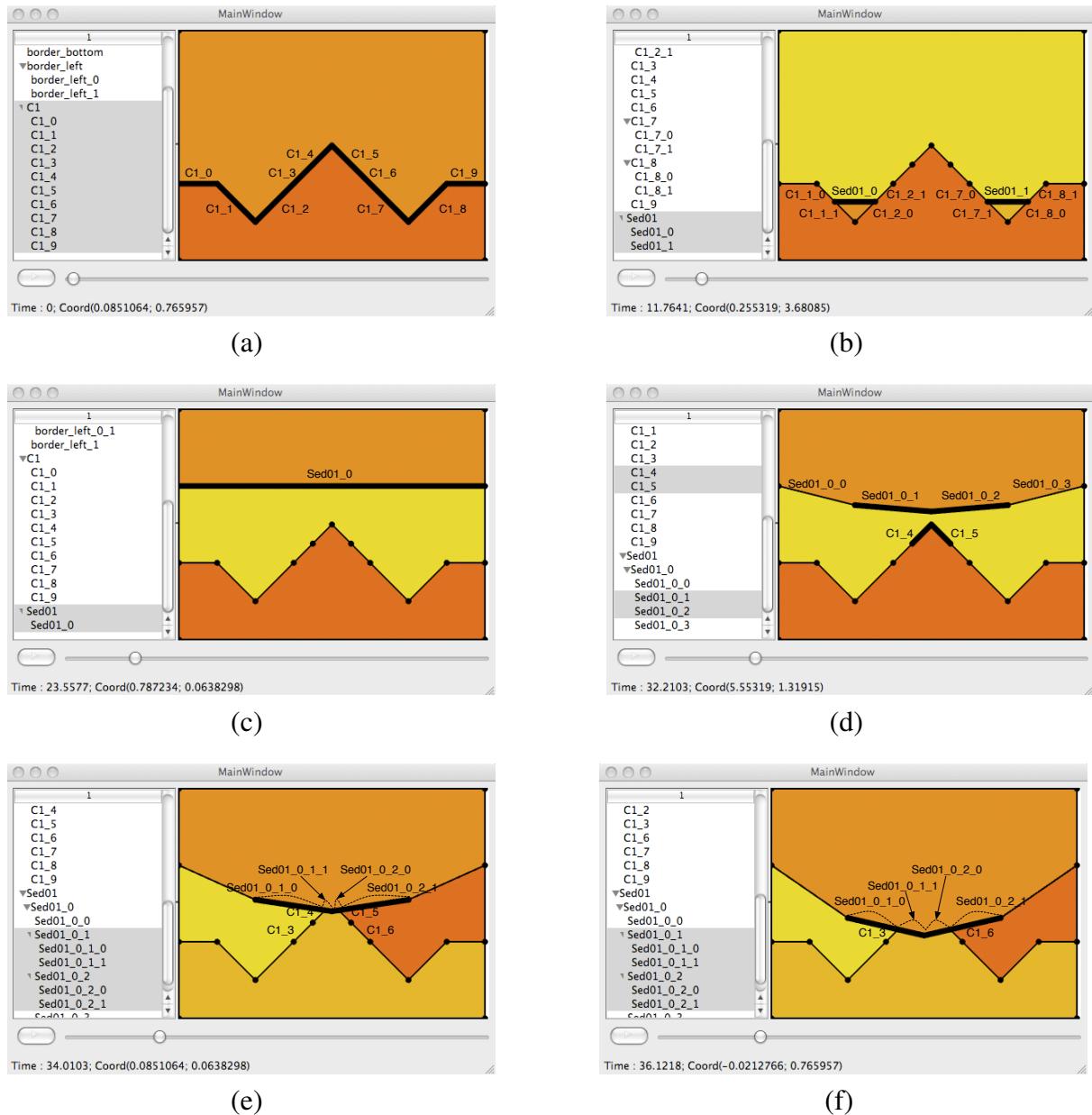


FIG. 5.29 – Images de l’animation générée à partir du Listing 5.6 illustrant l’évolution des noms des entités : (a) Création du sous-sol à partir d’un profil (2-g-carte 1). (b) Début de sédimentation créant deux blocs disjoints (2-g-carte 2). (c) Fusion des deux blocs (2-g-carte 5). (d) Début de l’érosion et détection de la prochaine collision causée par le phénomène (2-g-carte 6). (e) Résultat après le traitement de la collision entre la surface d’érosion et le pic (2-g-carte 7). (f) L’érosion continue et supprime les arêtes $C1_4$ et $C1_5$ (2-g-carte 8).

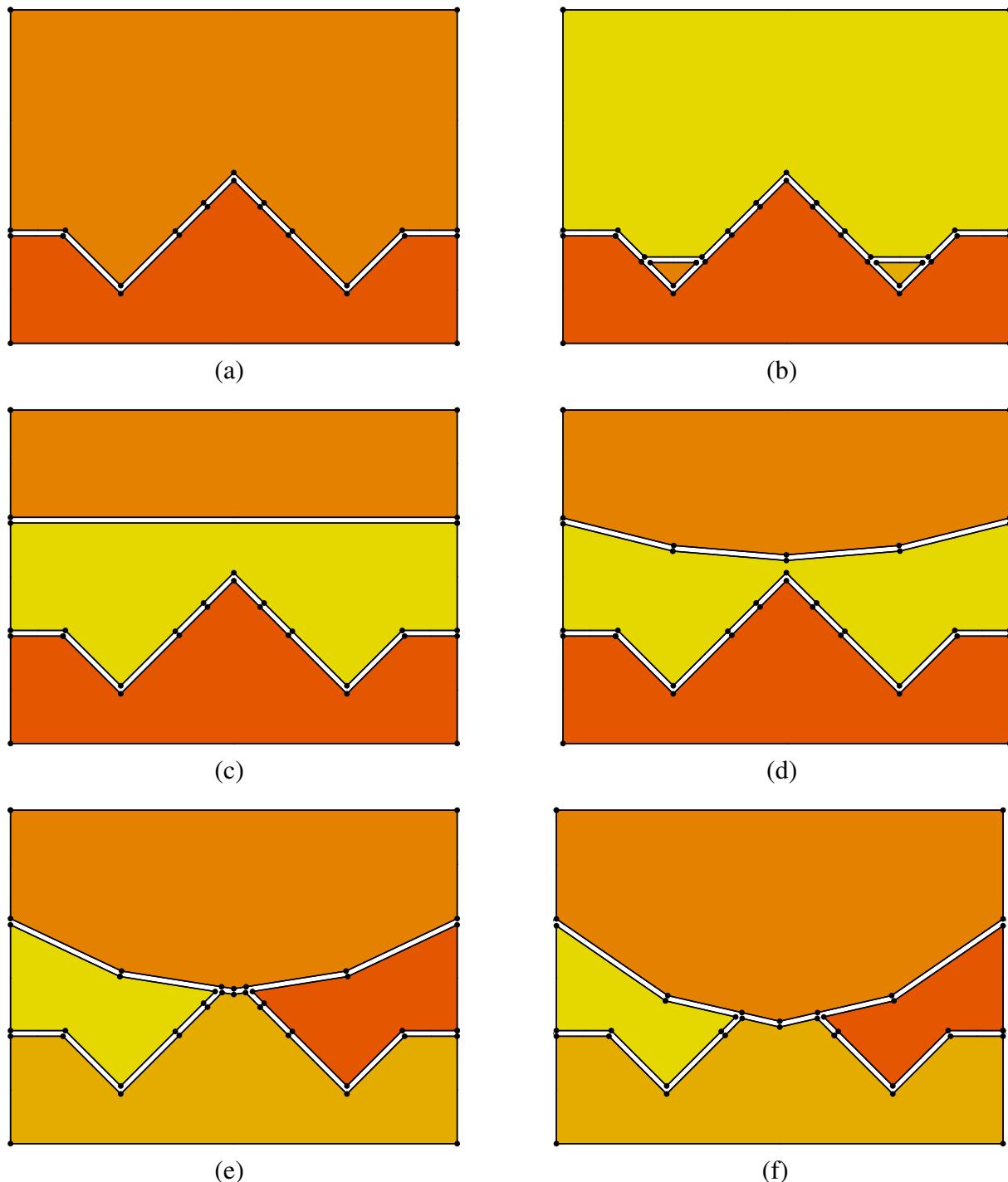


FIG. 5.30 – Première partie de l’animation générée à partir du Listing 5.6 - Vue topologique

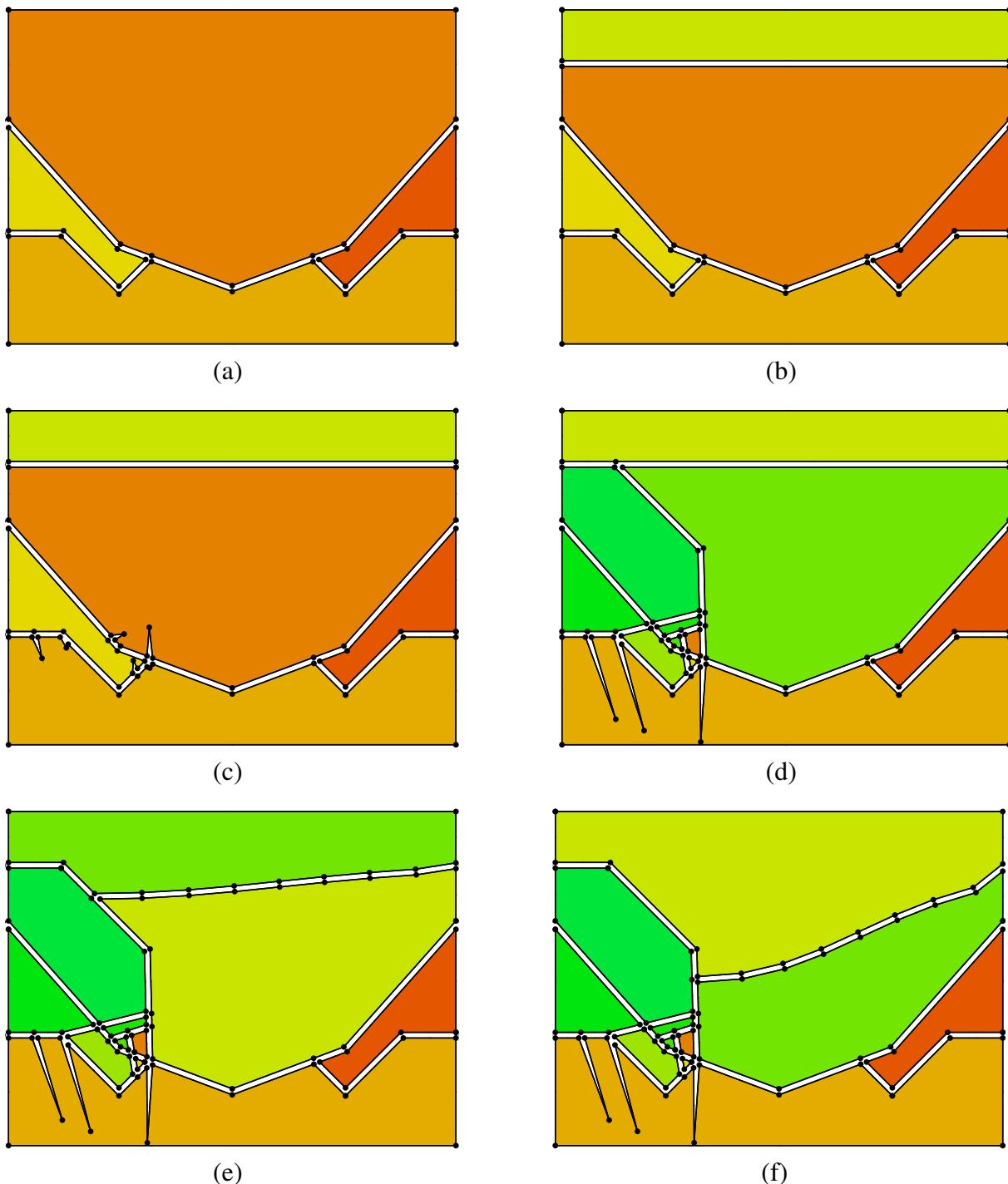


FIG. 5.31 – Seconde partie de l'animation générée à partir du Listing 5.6 - Vue topologique

5.5 Conclusion

Dans ce chapitre, nous avons illustré comment adapter notre système d'animation de structures à une application donnée, la géologie, et plus précisément fourni une méthodologie d'adaptation. Il s'agit d'une part de déterminer les phénomènes que l'on souhaite reproduire. Ensuite, pour chaque phénomène, nous définissons ses paramètres, qu'il faut fournir dans le scénario (ou calculer par simulation). Nous définissons ensuite les événements initiaux permettant de débuter le phénomène. Enfin, pour chaque événement possible (en 2D actuellement), nous recensons les divers cas que l'on peut rencontrer puis indiquons pour chacun d'eux, les transformations topologiques nécessaires. Actuellement, ces transformations sont décrites par un algorithme, mais des pistes de génération de code automatique sont envisagées pour améliorer le traitement de cette partie. La mise en œuvre de notre modèle en géologie pour quatre phénomènes basiques (sédimentation, érosion, apparition de faille et glissement) nous a permis de créer des animations tout à fait satisfaisantes, en particulier pour la problématique de la création de chenaux.

CONCLUSION ET PERSPECTIVES

Ce chapitre présente le bilan de cette thèse ainsi que de futures directions de travail initiées par nos travaux.

6.1 Bilan

Nous sommes d'abord partis du constat qu'il est important, en sciences expérimentales, d'étudier la structure et l'évolution des objets pour en comprendre le fonctionnement. Dans cette optique, nous avons proposé une nouvelle méthode d'animation de structures topologiques. Son objectif est de représenter l'évolution d'une structure à travers le temps à partir d'une description pouvant être donnée par l'utilisateur en des termes qui lui sont familiers, tout en garantissant la cohérence topologique du système au cours du temps. Ce modèle est décomposé en trois composantes. La première est structurelle et se base sur le formalisme des cartes généralisées. Afin de représenter les évolutions d'une structure, elle est organisée autour d'une suite ordonnée dans le temps de n -g-cartes. Ceci implique qu'entre deux n -g-cartes, aucun changement topologique n'intervient. La deuxième est événementielle et fournit à l'utilisateur les outils capables d'intervenir sur la scène grâce à des événements initiaux et un ensemble de comportements à adopter suivant les situations géométriques / topologiques (cas de collisions) détectées. La troisième composante est sémantique et permet de représenter de façon explicite les modifications subies par la structure ainsi que l'historique des entités. Pour cela, les entités sont désignées et manipulées par l'intermédiaire d'un script composé d'opérations topologiques.

Assurer la validité de ce modèle exigeait de maîtriser chacune des composantes citées ci-dessus et leurs interactions. Nous avons donc choisi de considérer l'évolution de structures 2D de manière exhaustive, plutôt que de répondre à la problématique en 3D de manière partielle seulement.

Concrètement, la première étape a été d'écrire un prototype d'animation topologique. Guidé par des discussions avec des géologues, nous avons décidé d'appliquer notre modèle dans le cadre de l'animation des couches du sous-sol. Le sous-sol est particulièrement bien adapté car il est par nature décomposable en couches, blocs, etc. que l'on peut décrire à l'aide d'une partition de l'espace. L'étape suivante a été de faire évoluer le sous-sol par l'application de différents

phénomènes naturels. Afin de pouvoir fournir une application dans des temps raisonnables, nous avons décidé de choisir quatre phénomènes classiques, à savoir la sédimentation, l'érosion, la création de failles et le glissement. Pour que le tout soit cohérent avec notre modèle, nous avons décrit ces quatre phénomènes en termes d'événements initiaux et en termes d'événements de collisions se produisant au cours de l'animation. Le géologue se contente ainsi de fournir sous forme textuelle un scénario d'évolutions composé de ces quatre phénomènes pour obtenir l'animation correspondante.

En résumé, les apports de ce travail sont :

- 1) un modèle d'animation de subdivision nD de l'espace permettant, grâce à la coopération de trois composantes événementielle, structurelle et sémantique, de créer un objet spatio-temporel représentant l'évolution de la structure au cours du temps ainsi qu'un script indiquant la liste des transformations successives que la partition a subi au cours du temps. Cette animation est contrôlée via un scénario spécifique à l'application visée, c'est-à-dire basé sur un vocabulaire « métier » ;
- 2) un système qui garantit la cohérence entre le modèle topologique et géométrique lors d'évolution de la structure, via un mécanisme de prédiction des événements topologiques basé : sur la détection des auto-collisions de la partition d'une part ; et sur un mécanisme de règles d'autre part, permettant, pour un événement détecté, un phénomène simulé et un contexte local observé, de définir précisément la transformation topologique à appliquer ;
- 3) une technique de détection des auto-collisions d'une partition en se basant sur la topologie, plus précisément en restreignant la recherche des collisions potentielles aux entités topologiques incidentes à une même cellule ;
- 4) une implantation de ce système dans le cas $2D$, définissant les types d'événements de collisions potentielles, les algorithmes de détection de ces collisions, les opérations de base nécessaire à la définition des traitements de collision, la désignation basée sur les arêtes et la syntaxe du script bas-niveau ;
- 5) une application de ce système en géologie, considérant quatre phénomènes fondamentaux, la sédimentation, l'érosion, la création de faille et le glissement. Le scénario permet d'indiquer l'ensemble des phénomènes à reproduire, leur durée de vie et leurs paramètres. Des applications concrètes, comme la construction de chenaux, ont été étudiées.

6.2 Perspectives

Ce paragraphe présente quelques perspectives de recherche donnant directement suite à ce travail. Nous constatons d'abord que les deux principales limites de ce travail sont, d'une part, l'implantation cantonnée au cas $2D$ et, d'autre part, le fait que les descriptions données sous forme de scénario ne sont pas totalement intuitives à cause du mécanisme de désignation. Nous constatons en outre que la partie événementielle se base sur des techniques de détections des collisions continues, techniques bien connues pour leur manque de robustesse sur le plan numérique. De ce fait, trois directions principales sont envisagées pour poursuivre le travail initié dans cette thèse. Elles concernent respectivement l'extension à la $3D$, les approches visant à faciliter la manipulation de ce système pour l'utilisateur final et les techniques d'amélioration de la robustesse du système.

6.2.1 Instanciation du modèle en 3D

Le but de cette thèse était de proposer un nouveau modèle d'animation topologique proposant des mécanismes haut-niveau décrivant l'évolution entièrement contrôlée des entités d'une scène, tout en assurant la cohérence topologique et géométrique à chaque instant. Ce modèle décrit, il fallait choisir une dimension d'instanciation. Pour limiter la complexité des opérations liées à la topologie, au plongement (géométrique, désignation), à la détection de collisions, nous avons limité notre étude au cas 2D, ce qui n'enlève rien à sa généralité. Pour étendre à la 3D, nous devons reprendre toute la méthodologie de conception en vue de cette nouvelle instantiation. Cette adaptation concerne les trois composantes de notre système (structurelle, événementielle et sémantique) que nous détaillons dans la suite.

6.2.1.1 Partie structurelle

En 3D, le principe de construction du modèle est inchangé. La partie structurelle reste composée d'une succession de n -g-cartes fermées connexes, ici, plus spécifiquement, des 3-g-cartes fermées connexes.

En ce qui concerne les opérations de manipulation de la structure topologiques, les opérations « élémentaires » (contractions, insertions et suppressions de cellules) sont d'ores et déjà définies en dimension n ; il en est de même pour des opérations de modélisation plus complexes (extrusion, produit cartésien, chanfreinage, etc.). Si des opérations plus sophistiquées spécifiques à des applications particulières apparaissent, il faudra alors les répertorier et les définir au moins en dimension 3, sinon en dimension n .

6.2.1.2 Partie événementielle

Les événements sont insérés dans une file à priorité sans a priori sur la dimension. On peut donc reprendre ce concept en dimension 3. Le changement se situe au niveau de la détection de collisions qui dépend directement du plongement géométrique considéré et des traitements des différents cas de collision : dans le cas du plongement linéaire, il faut considérer, en plus des collisions sommet/sommet et sommet/arête, les collisions sommet/face, arête/arête et arête/face. Certaines de ces collisions pourraient entraîner des modifications de la structure topologique que nous n'avons pas considéré jusqu'ici (par exemple, un sommet intersectant une face pourrait scinder cette dernière en plusieurs morceaux) : de nouveaux motifs de transformations topologiques seraient alors à définir.

6.2.1.3 Partie sémantique

La partie sémantique repose essentiellement sur les paramètres des opérations topologiques. Ces paramètres dépendent en partie du mécanisme de désignation des entités. Or, ce mécanisme est actuellement défini sur des arêtes, à partir desquelles on désigne les sommets extrémités et les faces incidentes. Mais rien ne permet actuellement d'affirmer que baser la désignation sur les arêtes permettra de désigner les volumes incidents. Une évolution intuitive vers la 3D laisse penser que baser la désignation sur les faces de la scène est plus pertinent. Seule une étude approfondie permettra de répondre à cette question.

6.2.2 Améliorations pour une utilisation plus facile du système

Pour le moment, le système est assez complexe à utiliser. Nous avons déjà indiqué que l'adaptation à une application donnée exige l'écriture d'algorithmes de reconnaissance de motif et transformations locales de la structure. L'utilisation de règles de transformation est incontournable, et devrait bénéficier de travaux actuellement effectués dans l'équipe IG du laboratoire XLIM-SIC [57]. Mais au delà de cette amélioration visant les concepteurs des applications, des améliorations sont également à proposer pour que l'utilisateur final puisse véritablement exploiter les applications développées. Pour cela, il faut garantir à la fois l'écriture aisée de scénario mais également permettre un meilleur contrôle du résultat, en particulier lorsque plusieurs phénomènes sont simulés en même temps. Nous pensons ainsi qu'un système de modification partielle de scénario et de recalcul adaptatif de l'animation serait un outil particulièrement utile.

6.2.2.1 Améliorer la description des cas de collision

Lorsqu'un événement de collision est détecté, il faut le traiter. Pour cela, nous devons définir le cas de collision et le résoudre en choisissant une méthode de résolution dépendant de l'application visée et du contexte. Pour déterminer la solution à appliquer, on pourrait utiliser un algorithme listant et testant tous les cas. Cependant, cette architecture est rigide (l'insertion de nouveaux cas peut être délicate) et elle n'est pas intuitive (l'utilisateur final doit connaître la programmation et la topologie pour fournir les méthodes de résolution adaptées à son application). Il serait plus simple de définir des motifs [29][57], correspondant à une partie de la structure à rechercher et des conditions à vérifier, et leur traitement. Cette approche est en partie utilisée dans ce mémoire mais n'est pas totalement exploitée car les méthodes proposées dans le domaine ne permettent pas encore de décrire tous les cas que nous avons rencontrés.

6.2.2.2 Construction de scénarii

L'écriture de scénarii textuels demande une certaine maîtrise de l'outil et nécessite une visualisation quasiment étape par étape de l'animation résultante. En outre, l'écriture du scénario nécessite une bonne connaissance du processus de désignation des entités et l'historique auquel il fait référence. Souvent, un phénomène demandé dans le scénario peut se révéler impossible, car il fait référence à des entités qui n'existent plus au moment où le phénomène s'enclenche. Une gestion plus souple des désignations (tenant en compte l'historique) pourrait améliorer la conception, en proposant des solutions de compromis (signalées tout de même à l'utilisation, au travers d'avertissements). Il est cependant clair qu'une interface dédiée à la conception de scénarii serait un outil précieux. Une autre approche serait de coupler le système d'animation à un simulateur qui permettrait de gérer les déplacements et déformations, déciderait des phénomènes à enclencher, etc. De cette manière, on se passerait de scénario ou bien le scénario s'adresserait au simulateur et non au système d'animation.

6.2.2.3 Contrôle de l'animation

Il faut noter que nous gérons les événements localement, c'est-à-dire que les transformations topologiques sont effectuées sur la zone de l'événement et non globalement sur les entités associées au phénomène. De ce fait, l'utilisateur a un contrôle assez faible (une fois les règles définies), et ne peut qu'ajuster les priorités des entités, si ces dernières sont prises en compte.

Ce faible contrôle peut très bien déboucher sur des transformations non désirées par l'utilisateur, mais correctes sur le plan topologique. De même, si plusieurs phénomènes sont simultanés dans le scénario, il n'est pas garanti que les transformations combinées fournissent le résultat attendu, car les transformations sont définies sur les phénomènes isolés. Ce type de problème se pose peu en géologie (pas de sédimentation et d'érosion simultanée ; ou événements de type faille considérés comme instantané dans l'échelle géologique) mais nécessiterait une attention particulière dans d'autres domaines.

6.2.2.4 Rejeu de scénarii

Lors de la création d'une animation, l'utilisateur peut désirer modifier une partie du scénario si le résultat attendu n'est pas celui escompté. Dans une approche classique, l'utilisateur doit vérifier et/ou redéfinir toutes les actions qui suivent ses modifications car les entités de la scène peuvent ne plus exister, ne plus avoir été créées, etc. Le concept de *nomination persistante*, essentiellement utilisé dans le domaine de la CAO actuellement [50][2], consisterait à compléter le modèle de construction de scénario : des informations enregistrées lors de la création du scénario initial seraient utilisées, après la modification d'une certaine étape de ce scénario, pour mettre automatiquement à jour les étapes suivantes en tenant compte des modifications éventuelles appliquées aux entités.

6.2.3 Amélioration de la robustesse et des performances

Au coeur de la robustesse et des performances de notre système se trouve le système de détection des collision. Pour le moment, la détection est assez fiable mais cette robustesse se fait au détriment du temps de calcul. Nous étudions donc ici quelques pistes pour améliorer les performances tout en ne dégradant pas la robustesse.

6.2.3.1 Améliorer les performances

Dans ce travail, nous avons tenu à traiter les résolutions d'équations de mouvements et d'intersections de la manière la plus souple et précise possible, en ayant recours à des bibliothèques externes de résolution formelle d'équations, impactant fortement le temps de création d'une animation. Lorsque les améliorations citées ci-dessus auront été mises au point (notamment l'extension à la 3D), il conviendra de réfléchir au moyen d'accélérer les temps de traitement pour obtenir un outil aussi interactif que possible. Ceci passe par une analyse des types de trajectoires à prendre en compte et établir des algorithmes de résolution analytique ou approchée de ces collisions. En outre, quelques optimisations classiques, à base de volumes englobants ou partitions de l'espace compléteraient avantageusement toute la chaîne de détection des collisions.

6.2.3.2 Améliorer la robustesse

La détection de collision utilisée repose sur un modèle de géométrie classique, mais est fortement dépendant de divers paramètres de précision ϵ . Afin d'améliorer la robustesse de la détection de collision, nous pourrions utiliser les algèbres géométriques. Les algèbres géométriques sont un outil mathématique qui consiste en l'application à la géométrie des algèbres

de Grassmann et des algèbres de Clifford, introduites au 19ème siècle par Hermann Günther Grassmann et William Kingdon Clifford. L'intérêt de ces algèbres est de représenter des relations géométriques (intersections, incidences, isométries, etc.) par des termes algébriques et des opérations sur ces termes. Leur avantage, à la différence des méthodes classiques, est de s'affranchir de la définition d'un système de coordonnées et à s'abstraire de la dimension pour formuler des opérations ou des démonstrations géométriques. Les solutions proposées sont, par conséquent, souvent plus robustes et plus génériques que les solutions classiques [22, 17, 7].

SOURCES

A.1 Fonctions utilitaires pour le script bas-niveau

Listing A.1 – Fonctions utilitaires pour l’animation de structures topologiques

```

1   function clamp(val, min, max)
2     if val < min then return min end
3     if val > max then return max end
4     return val
5   end

6   function changeRange(cmin, cmax, fct)
7     return function (p, t)
8       local np = (cmax - cmin) * p + cmin
9       return fct(np, t)
10      end
11    end

12   function interpole(t, tmin, tmax)
13     t = clamp(t, tmin, tmax)
14     return (t - tmin) / (tmax-tmin)
15   end

16   function mix(a, b, p)
17     return a * (1 - p) + b * p
18   end

19   function linearInterpolation(x1, y1, x2, y2, p)
20     return mix(x1, x2, p), mix(y1, y2, p)
21   end

22   function createFixedInterpolatedLine(x1, y1, x2, y2)
23     return function (p, _)
24       return linearInterpolation(x1, y1, x2, y2, p)
25     end
26   end

27   function b_BSpline(i, t)
28     if i == -2 then
29       return (((-t + 3) * t - 3) * t + 1) / 6;
30     elseif i == -1 then
31       return (((3 * t - 6) * t) * t + 4) / 6;
32     elseif i == 0 then
33       return (((-3 * t + 3) * t + 3) * t + 1) / 6;
34     elseif i == 1 then
35
36
37
38
39
40

```

```

        return (t * t * t) / 6;
    end

    return 0;
end

function b_CatmullRom(i, t)
    if i == -2 then
        return ((-t + 2) * t - 1) * t / 2;
    elseif i == -1 then
        return (((3 * t - 5) * t) * t + 2) / 2;
    elseif i == 0 then
        return ((-3 * t + 4) * t + 1) * t / 2;
    elseif i == 1 then
        return ((t - 1) * t * t) / 2;
    end

    return 0;
end

function p_BSpline(i, t, tbl, b)
    local px = 0;
    local py = 0;

    if (i == table.getn(tbl)) then
        i = i + 1
    end

    for j = -2, 1 do
        local v;
        if (i + j < 1) then
            v = tbl[1];
        elseif (i + j <= table.getn(tbl)) then
            v = tbl[i + j];
        else
            v = tbl[table.getn(tbl)];
        end
        px = px + b(j, t) * v[1];
        py = py + b(j, t) * v[2];
    end

    return px, py;
end

function createEmbeddingFunctionFromFileBSpline(filename)
    return createEmbeddingFunctionFromFileControlCurve(filename, b_BSpline, 2)
end

function createEmbeddingFunctionFromFileCatmullRom(filename)
    return createEmbeddingFunctionFromFileControlCurve(filename, b_CatmullRom, 1)
end

function g_XSpline(u, q, p)
    return q * u + 2 * q * math.pow(u, 2) + (10 - 12 * q - p) * math.pow(u, 3)
        + (2 * p + 14 * q - 15) * math.pow(u, 4) + (6 - 5 * q - p) * math.pow(u, 5);
end

function h_XSpline(u, q)
    return q * u + 2 * q * math.pow(u, 2) - 2 * q * math.pow(u, 4) - q * math.pow(u, 5);
end

function eval_XSpline(t, lv, ls)
    local DELTA = 1.0

    if t >= table.getn(lv) then
        t = table.getn(lv) - 1
    end

    if t > 0 and t < table.getn(lv) then
        local kf = math.floor(t) - 1
        local k = kf

        local T0p = kf + 1

```

```

115      if 1s[k + 1] > 0 then
T0p = T0p + 1s[k + 1] * DELTA
end

120      local T1p = kf + 2
      if 1s[k + 2] > 0 then
T1p = T1p + 1s[k + 2] * DELTA
end

125      local T2m = kf + 1
      if 1s[k + 1] > 0 then
T2m = T2m - 1s[k + 1] * DELTA
end

130      local T3m = kf + 2
      if 1s[k + 2] > 0 then
T3m = T3m - 1s[k + 2] * DELTA;
end

135      local tmp_factor = 2 / (DELTA * DELTA);
      local pm1 = math.pow(kf - T0p, 2) * tmp_factor;
      local pm0 = math.pow(kf + 1 - T1p, 2) * tmp_factor;
      local pp1 = math.pow(kf + 2 - T2m, 2) * tmp_factor;
      local pp2 = math.pow(kf + 3 - T3m, 2) * tmp_factor;

140      local qp0
      if 1s[k + 1] < 0 then
qp0 = - 1s[k + 1] / 2.0
else
qp0 = 0;
end

145      local qp1
      if 1s[k + 2] < 0 then
qp1 = - 1s[k + 2] / 2.0
else
qp1 = 0;
end

150      local qp2
      if 1s[k + 1] < 0 then
qp2 = - 1s[k + 1] / 2.0
else
qp2 = 0;
end

155      local qp3
      if 1s[k + 2] < 0 then
qp3 = - 1s[k + 2] / 2.0
else
qp3 = 0;
end

160      local A0, A1, A2, A3
      if t <= T0p then
A0 = g_XSpline((t - T0p) / (kf - T0p), qp0, pm1);
else
170      if qp0 > 0 then
A0 = h_XSpline((t - T0p) / (kf - T0p), qp0)
else
A0 = 0
end
end

175      A1 = g_XSpline((t - T1p) / (kf + 1 - T1p), qp1, pm0);
      A2 = g_XSpline((t - T2m) / (kf + 2 - T2m), qp2, pp1);

      if t >= T3m then
A3 = g_XSpline((t - T3m) / (kf + 3 - T3m), qp3, pp2);
else
180      if qp3 > 0 then
A3 = h_XSpline((t - T3m) / (kf + 3 - T3m), qp3)

```

```

    else
        A3 = 0
    end
    end

    local SUM = A0 + A1 + A2 + A3;
    tv = {}

    if k > 0 then
        tv[1] = lv[k]
    else
        tv[1] = lv[1]
    end

    tv[2] = lv[k + 1]
    if k <= table.getn(lv) - 3 then
        tv[3] = lv[k + 2]
    else
        tv[3] = lv[table.getn(lv)]
    end
    if k < table.getn(lv) - 3 then
        tv[4] = lv[k + 3]
    else
        tv[4] = lv[table.getn(lv)]
    end

    local x, y
    x = (tv[1][1] * A0 + tv[2][1] * A1 + tv[3][1] * A2 + tv[4][1] * A3) / SUM
    y = (tv[1][2] * A0 + tv[2][2] * A1 + tv[3][2] * A2 + tv[4][2] * A3) / SUM

    return x, y
end

return nil, nil
end

function createEmbeddingFunctionFromFileXSpline(filename)
dofile(filename)
local surfpt = surf
local surfsk = surf_sk

return function(p, t)
    if p == 1 then
        return surfpt[table.getn(surfpt)][1], surfpt[table.getn(surfpt)][2]
    end
    local p1 = (table.getn(surfpt) - 1) * p + 1

    return eval_XSpline(p1, surfpt, surfsk)
end
end

function createEmbeddingFunctionFromFileControlCurve(filename, b_fct, shift)
dofile(filename)
local surfpt = surf
local b = b_fct
return function (p, t)
    local n = table.getn(surfpt)
    local xmin = surfpt[1][1]
    local xmax = surfpt[n][1]
    local xcurrent = xmin * (1 - p) + xmax * p
    local last = surfpt[1]
    local vend = nil

    local lasti = 0;
    for i=1, table.getn(surfpt) do
        local vertex = surfpt[i]
        if (last[1] <= xcurrent) and (vertex[1] >= xcurrent) then
            vend = vertex
            lasti = i
            break
        end
    end
    last = vertex
end

```

```

260
261     local p1
262
263     if vend[1] ~= last[1] then
264         p1 = (xcurrent - last[1]) / (vend[1] - last[1])
265     else
266         p1 = 0
267     end
268
269     return p_BSpline(lasti, p1, surfpt, b)
270 end
271
272 function morphesimple1(fctbegin, fctend, tmin, tmax, battachBegin, battachEnd)
273     return function(p, t)
274         if (t > tmax) then t = tmax end
275         if (t < tmin) then t = tmin end
276
277         local p11, p12 = fctend(p, t)
278         local p21, p22 = fctbegin(p, t)
279
280         if (battachBegin) then
281             if (p == 0) or (p == 1) then return p21, p22 end
282             elseif (battachEnd) then
283                 if (p == 0) or (p == 1) then return p11, p12 end
284             end
285
286             local ii = interpolate(t, tmin, tmax)
287
288             return mix(p21, p11, ii), mix(p22, p12, ii)
289         end
290     end
291
292 function dilatation(tmin, tmax, fct)
293     return function(p, t)
294         t = clamp(t, tmin, tmax)
295
296         local pt = interpolate(t, tmin, tmax)
297
298         local xb, yb = fct(0, t)
299         local xe, ye = fct(1, t)
300
301         local p1 = .5 - pt / 2
302         local p2 = .5 + pt / 2
303
304         local nxb, nyb = mix(xb, xe, p1), mix(yb, ye, p1)
305         local nxe, nye = mix(xb, xe, p2), mix(yb, ye, p2)
306
307         return mix(nxb, nxe, p), mix(nyb, nye, p)
308     end
309 end
310
311 function dilatationPtFixe(tmin, tmax, fct)
312     return function(p, t)
313         t = clamp(t, tmin, tmax)
314
315         local pt = interpolate(t, tmin, tmax)
316         local xb, yb = fct(0, t)
317         local xe, ye = fct(1, t)
318
319         local nxb, nyb = xb, yb
320         local nxe, nye = mix(xb, xe, pt), mix(yb, ye, pt)
321
322         return mix(nxb, nxe, p), mix(nyb, nye, p)
323     end
324 end
325
326 function glissement(bbegin, fct, tmin, tmax)
327     return function(p, t)
328         t = clamp(t, tmin, tmax)
329
330         local p1 = interpolate(t, tmin, tmax)

```

```

      local np
      if bbegin then
        np = p1 * p
      else
        np = (1 - p1) * p + p1
      end

      return fct(np, t)
    end
  end

  function scale(fct, scalex, scaley)
    if scaley == nil then scaley = scalex end

    return function(p, t)
      local x, y, a = fct(p, t)
      return x * scalex, y * scaley, a
    end
  end

  function translate(fct, tx, ty)
    if ty == nil then ty = tx end

    return function(p, t)
      local x, y, a = fct(p, t)
      return x + tx, y + ty, a
    end
  end

— Attention, angle en radian ! math.rad pour convertir
  function rotate(fct, a)
    local cosa = math.cos(a)
    local sina = math.sin(a)

    return function(p, t)
      local x, y, a = fct(p, t)
      return x * cosa - y * sina, x * sina + y * cosa, a
    end
  end

  function fctClamp(f, xmin, ymin, xmax, ymax)
    return function (p, t)
      x, y, a = f(p, t)
      return clamp(x, xmin, xmax), clamp(y, ymin, ymax), a
    end
  end

  function translateDynamic(interface1, bbegin1, interface2, bbegin2, fct)
    local p1
    local p2
    if bbegin1 then p1 = 0 else p1 = 1 end
    if bbegin2 then p2 = 0 else p2 = 1 end

    return function(p, t)
      local x1, y1 = animation:findEdgeEmbedding(interface1, t)(p1, t)
      local x2, y2 = animation:findEdgeEmbedding(interface2, t)(p2, t)

      local x, y = fct(p, t)

      return x - x1 + x2, y - y1 + y2
    end
  end

```

A.2 Reconnaissance de motifs

Pour reconnaître un motif, on peut essayer un calculer un isomorphisme de graphe entre le brin en cours et le motif à accorder. Si le calcul échoue, le motif n'est pas trouvé, sinon, on

teste les conditions (sémantique : désignation, dernier événement impliqué, etc. ; géométrique : position, taille d'arête, volume, etc.) que doivent vérifier sur chaque brin.

```

Entrées :  $b_1 \in G_2, b_2 G'_2$ 
Sorties : bool ;  $f, f^{-1} : \text{brin} \longrightarrow \text{brin}$ 
 $pbi_1, pbi_2 : \text{tableau\_associatif}(\text{brin}, \text{int})$ 
 $bf_1, bf_2 : \text{file}(\text{brin}) ; bc_1, bc_2 : \text{brin}$ 
 $bf_1.\text{enfiler}(b_1) ; bf_2.\text{enfiler}(b_2)$ 
 $pbi_1[b_1] \leftarrow 0 ; pbi_2[b_2] \leftarrow 0$ 
 $f_1, f_2 : \text{file}(\text{brin})$ 
 $n : \text{int} \leftarrow 1$ 
tant que non  $bf_1.\text{vide}()$  et non  $bf_2.\text{vide}()$  faire
     $bc_1 = bf_1.\text{defiler}() ; bc_2 = bf_2.\text{defiler}()$ 
    pour chaque  $\text{involution } \alpha_i$  faire
        si  $pbi_1[bc_1\alpha_i] \neq pbi_2[bc_2\alpha_i]$  alors retourner ( $false, nil$ )
        si  $pbi_1[bc_1\alpha_i] = nil$  et
             $pbi_2[bc_2\alpha_i] = nil$  alors
                 $pbi_1[bc_1\alpha_i] \leftarrow n ; pbi_2[bc_2\alpha_i] \leftarrow n$ 
                 $bf_1.\text{enfiler}(bc_1\alpha_i) ; bf_2.\text{enfiler}(bc_2\alpha_i)$ 
                 $++n$ 
             $f_1.\text{enfiler}(bc_1\alpha_i) ; f_2.\text{enfiler}(bc_2\alpha_i)$ 
        si  $f_1 \neq f_2$  alors retourner ( $false, nil$ )
         $f_1.\text{vider}() ; f_2.\text{vider}()$ 
retourner ( $true, IsoFunc(pbi1,pbi2)$ )

```

Algorithme 6 : Calcul l'isomorphisme entre deux n-g-cartes s'il existe (nD)

PUBLICATIONS PERSONNELLES

Conférences internationales avec comité de lecture

- P.-F. Léon, X. Skapin, P. Meseure, *A topology-based animation model for the description of 2D models with a dynamic structure*, Virtual Reality Interactions and Physical Simulation (VRIPHYS), Grenoble, France, November 2008.
- P.-F. Léon, X. Skapin, P. Meseure, *Topologically-based animation for describing geological evolution*, International Conference on Computer Vision and Graphics (ICCVG), Warsaw, Poland, September 2006.

Revue nationale avec comité de lecture

- P.-F. Léon, X. Skapin, P. Meseure, *Modèle générateur d'évolutions géologiques par animation basée sur la topologie*, REFIG2009, vol. 3 (1), 2009.

Conférences nationales sans comité de lecture

- P.-F. Léon, X. Skapin, P. Meseure, *Animation événementielle de structures topologiques : application à la construction de chenaux*, AFIG2007, Marne la vallée, France - Novembre 2007.
- P.-F. Léon, X. Skapin, P. Meseure, *Modélisation d'évolution de couches géologiques*, GTMG2006, Cachan, France - Mars 2006.

BIBLIOGRAPHIE

- [1] Amaury AUBEL et Daniel THALMANN : Realistic deformation of human body shapes. *In Proc. Computer Animation and Simulation 2000*, pages 125–135, 2000.
- [2] M. BABA-ALI, D. MARCHEIX, X. SKAPIN et Y. BERTRAND : Intégration des opérations de nomination dans un modèle géométrique 3d. *In Journées de l'Association Française d'Informatique Graphique*, November 2005.
- [3] A. BAC, V. Tran NAM, M. DANIEL et M. PERRIN : Traitement de surfaces géologiques pour la construction de modèles 3d. *In GTMG 2005*, 2005.
- [4] BAUMGART : A polyhedron representation for computer vision. *SIGGRAPH Comput. Graph.*, pages 589–596, 1975.
- [5] D. BECHMANN, Y. BERTRAND et S. THERY : Continuous free form deformation. *In COMPUGRAPHICS '96 : Proceedings of the fifth international conference on computational graphics and visualization techniques on Visualization and graphics on the World Wide Web*, pages 1715–1725, New York, NY, USA, 1997. Elsevier Science Inc.
- [6] D. BECHMANN et N. DUBREUIL : Animation through space and time based on a space deformation model. *The Journal of Visualization and Computer Animation*, 4(3):165–184, juillet–septembre 1993.
- [7] T. BELLET, A. ARNOULD, S. CHARNEAU et L. FUCHS : Modélisation nd à base d’algèbres géométriques. *In AFIG 2008*, Toulouse, November 2008.
- [8] Paul BORREL et Dominique BECHMANN : Deformation of n-dimensional objects. *In SMA '91 : Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 351–369, New York, NY, USA, 1991. ACM.
- [9] S. BRANDEL, D. BECHMANN et Y. BERTRAND : Stigma : a 4-dimensional modeller for animation, 1998.
- [10] S. BRANDEL, D. BECHMANN et Y. BERTRAND : The thickening : an operation for animation. *The Journal of Visualization and Computer Animation*, 11(5):261–277, 2000.
- [11] S. BRANDEL, M. PERRIN, J.-F. RAINAUD et S. SCHNEIDER : Geological interpretation makes earth models easier to build. *In EAGE 63rd Conference, Extended Abstracts*, June 2001.
- [12] S. BRANDEL, S. SCHEIDER, M. PERRIN, N. GUIARD, J.-F. RAINAUD, P. LIENHARDT et Y. BERTRAND : Automatic building of structured geological models. *Journal of Computing and Information Science in Engineering*, june 2005.
- [13] E. BRISSON : Representing geometric structures in d dimensions : topology and order. *In SCG '89 : Proceedings of the fifth annual symposium on Computational geometry*, pages 218–227, New York, NY, USA, 1989. ACM.

- [14] Richard L. BURDEN et Douglas J. FAIRES : *Numerical Analysis*. Brooks Cole, December 2004.
- [15] N. BURNTYK et M. WEIN : Computer generated key frame animation. *Journal of the Society of Motion Picture and Television Engineers*, pages 149–153, 1971.
- [16] Christos G. CASSANDRAS et Stephane LAFORTUNE : *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [17] S. CHARNEAU : *Étude et application des algèbres géométriques pour le calcul de la visibilité globale dans un espace projectif de dimension $n \geq 2$* . Thèse de doctorat, Université de Poitiers – Laboratoire SIC, Décembre 2007.
- [18] Robert D. COOK, David S. MALKUS, Michael E. PLESHA et Robert J. WITT : *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, 2007.
- [19] Sabine COQUILLART et Pierre JANCÉNE : Animated free-form deformation : an interactive animation technique. *SIGGRAPH Comput. Graph.*, 25(4):23–26, 1991.
- [20] G. DAMIAND, M. DEXET-GUIARD, P. LIENHARDT et É. ANDRES : Removal and contraction operations to define combinatorial pyramids : application to the design of a spatial modeler. *Image and Vision Computing*, 23(2):259–269, February 2005.
- [21] Mathieu DESBRUN et Marie-Paule CANI : Smoothed particles : A new paradigm for animating highly deformable bodies. In Ronan BOULIC et Gérard HÉGRON, éditeurs : *Eurographics Workshop on Computer Animation and Simulation, EGCAS '96, August, 1996*, pages 61–76, Poitiers, France, August 1996. Springer-Verlag. Published under the name Marie-Paule Gascuel.
- [22] Leo DORST, Daniel FONTIJNE et Stephen MANN : *Geometric Algebra for Computer Science : An Object-Oriented Approach to Geometry (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
- [23] P. DWORKIN et D. ZELTER : A new model for efficient dynamic simulation. In *EUROGRAPHICS Workshop on Computer Animation and Simulation (EGCAS)*, pages 135–147, Barcelone, septembre 1993.
- [24] J. EDMONDS : A combinatorial representation for polyhedral surfaces. In *Notices*, volume 7. Amer. Math. Soc., 1960.
- [25] R. EGLI et N. F. STEWART : Chain models in computer simulation. *Math. Comput. Simul.*, 66(6):449–468, 2004.
- [26] H. ELTER et P. LIENHARDT : Extension of the notion of map for the representation of the topology of cellular complexes. In *4th Canadian Conference on Computational Geometry*, St John's, Canada, 1992.
- [27] M. FLOATER, Y. HALLBWACHS, O. HJELLE et M. REIMERS : A cad-based approach to geological modelling. In *Modeling 98 Conference*, Nancy, France, Juin 1998.
- [28] Fabien GARAT et Daniel THALMANN : Guiding and interacting with virtual crowds in real-time. In *In Proceedings of Eurographics Workshop on Animation and Simulation*, pages 23–34. Eurographics, SpringerVerlag, 1999.
- [29] J.-L. GIAVITTO et O. MICHEL : Mgs : a rule-based programming language for complex objects and collections. *Electr. Notes Theor. Comput. Sci.*, 59(4), 2001.
- [30] Site web gocad. <http://www.gocad.com/>.
- [31] S. GOTTSCHALK, M. C. LIN et D. MANOCHA : Obbtree : A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

- [32] N. GUIARD, M. PERRIN, J.-F. RAINAUD et Y. BERTRAND : Outils pour la modélisation de scènes géologiques 3d. *GTMG*, 2005.
- [33] Nicolas GUIARD : *Construction de modèles géologiques 3D par co-rangement de surfaces*. Thèse de doctorat, Université de Poitiers, 2006.
- [34] G. GUIMBERTEAU, O. TERRAZ, S. MÉRILLOU, D. GHAZANFARPOUR et D. PLEMENOS : Internal wood growth simulation based on subdivised 3d objects. Rapport technique, Laboratoire MSI, 2005.
- [35] Thomas C. HALES : The honeycomb conjecture. *DISCR.COMPUT.GEOM.*, 25:1, 2001.
- [36] T. JUND, D. CAZIER et J.-F. DUFOURD : Système de prédiction pour la détection de collisions dans un environnement déformable. In *Journées de l'Association Française d'Informatique Graphique*, 2008.
- [37] John LASSETER : Principles of Traditional Animation Applied to 3D Computer Animation. *SIGGRAPH87*, 21(4):35–44, juillet 1987.
- [38] Francis LAZARUS et Anne VERRAUST : Three-dimensional metamorphosis : a survey. *The Visual Computer*, 14(8/9):373–389, décembre 1998. ISSN 0178-2789.
- [39] Tong-Yee LEE et Po-Hua HUANG : Fast and intuitive metamorphosis of 3d polyhedral models using smcc mesh merging scheme. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):85–98, 2003.
- [40] P.-F. LÉON : Simulation d'évolution de couches géologiques. Mémoire de D.E.A., Université de Poitiers, 2005. Master 2 Recherche spécialité Fondements et Ingénierie de l'Informatique et de l'Image - Laboratoire Signal Image Communications.
- [41] P.-F. LÉON, X. SKAPIN et P. MESEURE : Topologically-based animation for describing geological evolution. In *International Conference on Computer Vision and Graphics*, September 2006.
- [42] P. LIENHARDT : Subdivisions of n-dimensional spaces and n-dimensional generalized maps. *SCG '89 : Proceedings of the fifth annual symposium on Computational geometry*, pages 228–236, 1989.
- [43] P. LIENHARDT : Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1), 1991. 59-82.
- [44] P. LIENHARDT : n-dimensional generalised combinatorial maps and cellular quasimani-folds. *International Journal of Computational Geometry and Applications*, 1994.
- [45] P. LIENHARDT, X. SKAPIN et A. BERGEY : Cartesian product of simplicial and cellular structures. *International Journal on Computational Geometry and Applications*, 14(3): 115–159, Juin 2004.
- [46] A. LINDENMAYER : Mathematical models for cellular interactions in development. *Journal of Theoretical Biology*, 18:280–315, 1968.
- [47] A. LINDENMAYER et G. ROZENBERG : Parallel generation of maps : Developmental systems for cell layers. In *Proceedings of the International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*, pages 301–316, London, UK, 1979. Springer-Verlag.
- [48] A. LUCIANI, S. JIMENEZ, J.L. FLORENS, C. CADOUX et O. RAOULT : Computational physics : a modeler simulator for animated physical objects. In *EUROGRAPHICS Workshop on Computer Animation and Simulation*, pages 425–437, Vienne, 1991.
- [49] J.-L. MALLET : Geomodeling. *Oxford University Press*, 2002.

- [50] D. MARCHEIX et G. PIERRA : A survey of the persistent naming problem. In *SMA '02 : Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 13–22, New York, NY, USA, 2002. ACM.
- [51] J.-P. MAY : *Simplicial Objects in Algebraic Topology*, volume 11 de *Van Nostrand Mathematical Studies*. Van Nostrand, première édition, 1967.
- [52] P. MESEURE : Animation basée sur la physique pour les environnements interactifs temps-réel, 2002. Mémoire de HDR.
- [53] P. MESEURE, A. KHEDDAR et F. FAURE : Détection des collisions. *Rapport de l'Action Spécifique CNRS Num. 90, RTP 7*, december 2003.
- [54] Soraia Raupp MUSSE, Branislav ULICNY, Amaury AUBEL et Daniel THALMANN : Groups and crowd simulation. In *SIGGRAPH '05 : ACM SIGGRAPH 2005 Courses*, page 2, New York, NY, USA, 2005. ACM.
- [55] M. PERRIN : Geological consistency : an opportunity for safe surface assembly and quick model exploration. *3D Modeling of Natural Objects, A Challenge for the 2000's*, 3:4–5, june 1998.
- [56] Jean-Pierre PETIT : *Le topologicon*. Belin, 1985.
- [57] M. POUDRET, J.-P. COMET, P. LE GALL, A. ARNOULD et P. MESEURE : Topology-based geometric modelling for biological cellular processes. In *Langage and Automata Theory and Applications*, Tarragona, Spain, Mars 2007.
- [58] X. PROVOT : Collision and self-collision handling in cloth model dedicated to design garments. *Graphics Interface*, pages 177–189, 1997.
- [59] Xavier PROVOT : Deformation constraints in a mass-spring model to describe rigid cloth behavior. In Wayne A. DAVIS et Przemyslaw PRUSINKIEWICZ, éditeurs : *Graphics Interface '95*, pages 147–154. Canadian Human-Computer Communications Society, 1995.
- [60] P. PRUSINKIEWICZ et A. LINDENMAYER : *The Algorithmic Beauty of Plants (The Virtual Laboratory)*. Springer, October 1990.
- [61] Fasheng QIU et Xiaolin HU : Behaviorsim : A learning environment for behavior-based agent. In *SAB '08 : Proceedings of the 10th international conference on Simulation of Adaptive Behavior*, pages 499–508, Berlin, Heidelberg, 2008. Springer-Verlag.
- [62] W. T. REEVES : Particle systems—a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, 1983.
- [63] Aristides G. REQUICHA : Representations for rigid solids : Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464, 1980.
- [64] Craig W. REYNOLDS : Flocks, herds, and schools : A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
- [65] Site web rml. <http://www.beicip.com/>.
- [66] Jarek ROSSIGNAC et Jay J. KIM : Computing and visualizing pose-interpolating 3d motions. *Computer-Aided Design*, 33(4):279–291, 2001.
- [67] George Thomas SALLEE : *Incidence Graphs of Convex Polytopes*. Thèse de doctorat, University of Washington, 1966.
- [68] Sébastien SCHNEIDER : *Pilotage automatique de la construction de modèles géologiques surfaciques*. Thèse de doctorat, École des Mines de Saint-Étienne, 2002.

- [69] Thomas W. SEDERBERG et Eugene GREENWOOD : A physically based approach to 2-d shape blending. In *SIGGRAPH '92 : Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1992. ACM.
- [70] Jean-Pierre SERRE : Homologie singuliere des espaces fibres. *The Annals of Mathematics*, 54(3):425–505, 1951.
- [71] Michal SHAPIRA et Ari RAPPOPORT : Shape blending using the star-skeleton representation. *IEEE Comput. Graph. Appl.*, 15(2):44–50, 1995.
- [72] C. SMITH : *On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling*. Thèse de doctorat, University of Calgary, April 2006.
- [73] Yi SONG, Li BAI et Yangsheng WANG : 3d object modelling for entertainment applications. In *ACE '06 : Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 62, New York, NY, USA, 2006. ACM.
- [74] O. TERRAZ, G. GUIMBERTEAU, S. MÉRILLOU et D. GHAZANFARPOUR : 3gmap l-systems : An application to the modeling of wood. *The Visual Computer*, 25(2):165–180, 2009.
- [75] Gino van den BERGEN : Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.
- [76] K. WEILER : Edge-based data structures for solid modeling in a curved surface environment. In *IEEE Comput. Graph. Appl.*, pages 5(1) :21–40, 1985.
- [77] K. WEILER : The radial edge structure : A topological representation for non-manifold geometric boundary modeling. In M.-J. WOZNY, H.-W. MC LAUGHLIN et J.-L. ENCARNACÃO, éditeurs : *Geometric Modeling for CAD Applications*, pages 3–36. Elsevier Science, 1988.

Animation et Contrôle de structures topologiques : application à la simulation d'évolution de couches géologiques 2D.

Résumé Cette thèse présente une nouvelle méthode d'animation basée sur la topologie. Elle consiste à animer une partition de l'espace et en assurer sa cohérence. Le système d'animation s'appuie sur un mécanisme événementiel qui détecte les incohérences topologiques. L'animation est générée à partir du traitement séquentiel de tous les événements. Lors du mouvement des entités, l'instant des collisions est calculé et des événements sont générés. Pour prendre en charge ces collisions, ces événements sont traités par rapport à leurs contextes locaux (géométrique, sémantique). Un traitement engendre des changements géométriques et topologiques et assure la cohérence entre le modèle géométrique et le modèle topologique. Cette thèse présente également une application en géologie permettant de générer l'animation de l'évolution du sous-sol à partir de phénomènes naturels décrits dans un scénario. Dans ce cadre, un scénario est composé d'une suite de phénomènes géologiques (sédimentation, érosion, création de failles et glissement) analysée par le système d'animation 2D. Un phénomène est traduit en un ensemble d'événements initiaux. Une fois l'animation générée, le géologue peut l'analyser et la valider grâce aux informations sémantiques et historiques fournies par le modèle.

Animation and Control of topological structures : application to the simulation of geological layer evolution in 2D.

Abstract This thesis presents a new animation method based on topology. It consists in animating a space partition while ensuring its consistency. The scenes are described from a high-level language depending on the application. The animation system relies on an event approach which detects topological events. The animation is generated from the sequential processing of all events. During the motion of entities, the date of collisions is computed and events are generated. For processing those collisions, events are handled with respect to their local contexts (both geometrical and semantical). This thesis also presents an application in geology which allows the user to generate a subsoil evolution from some natural phenomena described by a scenario. A scenario is made of a sequence of geological phenomena (sedimentation, erosion, fault creation and sliding) analyzed by the animation system. A phenomenon is therefore translated into a set of initial events. After the animation has been generated, the geologist can analyze and validate it with the semantical and historical information given by the model.

Discipline : Informatique graphique.

Mots clés : Modélisation procédurale - nD - 2D+1 - Animation - Topologie - G-Carte - Géologie

Pierre-François Léon- Laboratoire CNRS XLIM-SIC
Bât. SP2MI - Téléport 2, Bd Marie et Pierre Curie - BP 30179 - 86962 Futuroscope Cedex