Université de Montréal

Cadre de travail pour la spécification de systèmes avec des chaînes sur un complexe cellulaire

par

Richard Egli

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la faculté des études supérieures

en vue de l'obtention du grade de

Philosophiæ Doctor (Ph. D.) en informatique

Juin, 2000

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée

Cadre de travail pour la spécification de systèmes avec des chaînes sur un complexe cellulaire

présenté par

Richard Egli

a été évaluée par un jury composé des personnes suivantes :

Président :	Jean Meunier
Directeur de recherche :	Neil F. Stewart
Membre du jury :	Pierre Poulin
Examinateur externe :	Vadim Shapiro
Représentant du doyen :	Jean Meunier

Sommaire

Dans cette thèse, nous décrivons un cadre de travail pour la spécification et la manipulation de modèles de système. Notre but est de pouvoir modéliser facilement et de façon unifiée le comportement de ces systèmes.

Dans ce cadre, les systèmes sont modélisés en utilisant des chaînes qui permettent la spécification et la manipulation d'attributs associés aux éléments topologiques des systèmes. Le formalisme utilisé dans ce cadre est une modification et une extension d'un formalisme déjà existant.

Ce cadre de travail inclut une interface de programmation très facile à utiliser. Des exemples d'utilisation de cette interface sont ensuite décrits pour illustrer que des systèmes relativement complexes peuvent être modélisés de façon claire et concise avec les chaînes.

Mots clés :
Complexe cellulaire, cellule, chaîne, topologie, attribut, cadre de travail, modèle de système.

Table des matières

Table des figures

Remerciements

Je remercie sincèrement Neil Stewart, mon directeur de recherche. De cet homme d'une grande modestie, j'ai vraiment beaucoup appris. Il a su poser les questions qui m'ont fait avancer dans la bonne direction. Neil est un directeur très dévoué, qui ne compte pas ses heures.

Mes remerciement vont aussi à tous les membres du laboratoire d'infographie ainsi qu'au directeur Pierre Poulin pour leurs conseils et leur gentillesse. Je remercie aussi Normand Brière qui a bien voulu m'écouter et m'encourager dans mes recherches. Merci Normand !

Je remercie particulièrement ma conjointe Marie et mes enfants, Maude et Ariane, qui ont fait preuve de patience et de compréhension durant les recherches et la rédaction. Merci Marie pour ton appui inconditionnel dans la poursuite de mes études.

Je tiens aussi à souligner la contribution financière du CRSNG par l'intermédiaire de Neil Stewart ainsi que l'Université de Montréal pour l'attribution de la bourse FES-DIRO.

Remarques préliminaires

Certains mots se retrouvent dans le glossaire à la page 132. Il s'agit soit de mots qui ne sont pas expliqués dans la thèse, soit de mots dont la correspondance entre les termes anglais et français peut porter à confusion. À leur première apparition, ces mots seront mis en évidence par une *police de caractère oblique*.

La *police de caractère italique* a été choisie pour les mots anglais qui ont plusieurs sens, et pour lesquels nous avons préféré conserver une terminologie qui permet de faire le lien avec notre référence principale [1]. Cette police est aussi utilisée pour mettre en évidence une partie de texte et pour marquer les mots clés à leur première apparition.

Et, nous utilisons une `police de caractères avec des espacements constants` pour une ligne de code d'un programme (voir la section 2.2 pour plus de détails).

Chapitre 1

Introduction

Nous décrivons ici où se situe notre recherche puis nous décrivons des recherches en relation avec notre travail. Nous énumérons ensuite brièvement nos contributions et nous terminons par un aperçu du contenu des différents chapitres.

1.1 Mise en situation

Dans cette thèse, nous décrivons un *cadre de travail* pour la spécification et la manipulation de modèles de systèmes, tels que des modèles physiques ou pour d'autres applications. Plus précisément, le cadre de travail permet la spécification et la manipulation d'*attributs* associés avec les éléments topologiques des systèmes. Le concept de "cadre de travail" utilisé ici, inclut un formalisme et une *interface de programmation*. Dans le cadre de travail, les *modèles de chaînes* sont utilisés pour modéliser les systèmes.

Le modèle de chaînes fut introduit par Palmer et Shapiro [1]. Ces modèles introduisent un formalisme qui est utilisé dans un contexte de modélisation physique. Ils combinent la topologie, la géométrie et la représentation de données physiques

d'un objet[1]. Dans cette thèse, nous modifions et élargissons le modèle de chaînes. Une interface de programmation est proposée dans [2] appliquant le formalisme du modèle de chaînes. L'implantation de cette interface n'est pas disponible; la syntaxe est mal définie et difficile à utiliser. Nous avons donc développé une interface de programmation reflétant, entre autres les modifications et les extensions du formalisme. Cette interface de programmation est pour le langage de programmation C++ mais le choix du langage de programmation objet n'est pas important ; nous aurions pu choisir, par exemple Java. Des exemples illustrant le cadre de travail sont présentés : la plupart, des modèles physiques, comme un drapeau flottant au vent [3], et un exemple d'application en infographie, soit la subdivision de surface de Catmull-Clark [4].

Les objets définis par le formalisme du modèle de chaînes sont des *complexes cellulaires*. Cette façon de faire est souvent utilisée en *modélisation solide*. Un complexe cellulaire décrit un objet en terme de *cellules* de différentes dimensions et spécifie la topologie de celui-ci. Par exemple, un cube peut être décomposé en une 3-cellule (tout le cube), six 2-cellules (les faces, incluant les arêtes), douze 1-cellules (les arêtes, incluant les sommets) et huit 0-cellules (les sommets aux coins du cube). Les complexes et les cellules sont décrits en détail au chapitre 2.

Les *chaînes* sont utilisées dans le formalisme pour emmagasiner et manipuler les différents *coefficients* sur les différentes cellules d'un complexe. En fait, une p-chaîne est définie en associant un coefficient sur chacune des cellules de dimension p du complexe. Ces coefficients correspondent exactement aux attributs mentionnés au début de l'introduction. Prenons un exemple simple : lorsqu'un complexe cellulaire est formé de deux cubes partageant une face, il contient onze faces avec onze coefficients désignant l'aire de chacune des faces. Ces coefficients associés aux faces forment une 2-chaîne sur le complexe cellulaire. Des opérations sont ensuite définies sur les chaînes afin de pouvoir manipuler globalement (sur l'objet au com-

[1]Dans cette thèse, le mot "objet" est considéré comme étant un synonyme du mot "système".

plet) les coefficients. Nous expliquerons en détail les chaînes et nous donnerons des exemples pour montrer la facilité et la généralité avec lesquelles nous pourrons décrire différents systèmes.

L'utilisation de ce genre de formalisme est à l'origine motivée par une unification de la représentation du comportement d'un objet soumis à des lois. L'unification est un objectif scientifique fondamental car cela facilite la compréhension. Le fait d'avoir des outils de haut niveau permettant de décrire facilement, de façon unifiée et intuitive différents systèmes, engendre un plus haut degré d'abstraction. Avoir des outils de haut niveau, permet aussi de s'attaquer à des problèmes beaucoup plus complexes, car nous ne sommes plus préoccupés par des éléments de bas niveau.

En plus, l'unification a l'avantage pratique de fournir une représentation standard permettant d'échanger et de réutiliser les informations. Il faut, par contre, que ce standard soit simple et facile à utiliser. Le cadre de travail développé ici permet de décrire de façon très simple le comportement des systèmes représentés avec des chaînes sur un complexe. Il peut être utilisé pour des applications aussi variées que la simulation d'un environnement avec des lois physiques ou la description d'une méthode de subdivision de surface ou de volume.

1.2 Travaux antérieurs

Requicha [5] a été le premier à montrer l'utilité des chaînes dans le domaine de la modélisation solide (par exemple, pour vérifier si un ensemble de faces compose un solide bien formé en R^3). Ses recherches étaient basées directement sur la topologie classique [6]. Mentionnons les travaux de Tonti [7], basés aussi sur cela, pour l'élaboration d'un schéma de classification de théories physiques dans un but d'unification en ingénierie et en physique [8]. Des problèmes pratiques en physique [9] ont aussi été représentés en utilisant la topologie classique. Par contre, ces applications limitent les coefficients des chaînes à des éléments d'un *groupe*.

Palmer et Shapiro ont amené l'idée que les coefficients peuvent être plus généraux, c'est-à-dire que les coefficients peuvent faire partie d'un ensemble plus ou moins arbitraire. Ils ont aussi proposé les opérations globales sur les chaînes. D'autres références historiques sont disponibles dans une publication de Chard et Shapiro [10].

Plusieurs chercheurs dans le domaine de la modélisation solide, par exemple [11], [12] et [13], ont utilisé un formalisme basé sur les complexes. Dans [11], le formalisme est basé seulement sur des *simplexes* [6] tandis que dans [13], le formalisme est basé sur des simplexes structurés dans des complexes cellulaires. Dans [12], une classe très générale de complexes cellulaires fut introduite. Weiler [14] travaille avec une représentation frontalière limitée à un espace à trois dimensions. Aucun de ceux-ci n'a considéré les chaînes. Par contre, Rossignac et O'Connor [12] de même que Weiler [14] ont la notion d'attributs associés avec des cellules mais les opérations globales sur les attributs n'ont pas été considérées. Différents systèmes auraient pu utiliser notre cadre de travail pour l'implantation de leur problème spécifique. La diversité des applications illustre la possibilité d'unification du modèle de chaînes. Même si nous nous concentrons dans le domaine de l'infographie, nous pouvons en énumérer[2] quelques-unes :

- système avec des réseaux de masses-ressorts [3] ;
- techniques de subdivision de surfaces [4] ou de volumes [15] ;
- simulation de l'air/fumée [16] ou eau [17] ;
- déformation d'objets en R^3 [15, 18] ;
- maillage progressif [19] ;
- figures articulées [21].

Certains de ces articles introduisent un niveau d'unification, par exemple, [18] présente une approche unifiée au niveau géométrique dans le contexte de la méthode d'éléments finis. D'autres [22, 23, 24], présentent des cadres de travail en

[2]Les trois premières applications seront décrites en détail dans la thèse.

animation. Le langage PLASM (*the Programming LAnguage for Solid Modeling*) [20] est un langage haut niveau inspiré du langage fonctionnel FL. Il est basé sur des complexes et il est utilisé pour définir des objets géométriques en modélisation solide. Tous ces systèmes sont par contre moins généraux pour définir des objets et leurs comportements que l'approche de modèle de chaînes présentée ici.

1.3 Contributions

Comme nous l'avons mentionné, notre cadre de travail est basé sur le formalisme de chaînes de Palmer et Shapiro [1, 2]. Par contre, nous avons ajouté à leur formalisme les extensions suivantes :

- les classes des coefficients sont plus générales (par exemple, les coefficients peuvent être des cellules) ;
- les opérateurs de chaînes *boundary* et *coboundary* sont généralisés par la méthode[3] nommée *dim_inc* ;
- la classe des cellules et des complexes est élargie, nous traitons aussi des cellules et des complexes non géométriques ;
- une méthode pratique est décrite pour le calcul de l'orientation relative des cellules ;
- la manipulation d'une partie d'une chaîne est possible grâce au mécanisme de sélection ;
- un nouveau type de chaîne dont les coefficients peuvent être associés à plus d'une cellule est développé (chaîne de sous-chaînes).

Ces extensions, et leurs implantations dans une interface de programmation pour un langage orienté objet, constituent les contributions les plus importantes de cette thèse.

[3] Le mot "méthode" ici est employé dans un contexte de programmation objet.

1.4 Aperçu des différents chapitres

Dans le chapitre 2, nous expliquons la base du formalisme et de l'interface de programmation du cadre de travail. Au chapitre 3, des exemples d'utilisation de l'interface de programmation sont détaillés. Des ajouts au cadre de travail sont ensuite expliqués au chapitre 4, suivis au chapitre 5 d'exemples d'utilisation de l'interface de programmation du cadre de travail étendu. Dans la conclusion, au chapitre 6, nous récapitulons les avantages à utiliser notre cadre de travail, pour ensuite donner les avenues de recherches ouvertes par ce travail.

Chapitre 2

Cadre de travail

Dans ce chapitre, nous introduisons la base du cadre de travail. Nous expliquons les concepts formels dans le modèle de chaînes ainsi que les modifications et certaines extensions que nous apportons. L'interface de programmation est aussi introduite dans un contexte de programmation objet.

2.1 Cellules et complexes cellulaires

Contrairement à notre référence principale [1], ainsi que dans d'autres références [11, 12], nous ne supposons pas que les cellules et les complexes cellulaires ont une réalisation *géométrique*. Dans cette thèse, les liens entre les cellules sont exclusivement topologiques[1]. Ce niveau de généralité nous permet d'utiliser notre cadre de travail pour des structures telles que des *graphes* ou des *hypergraphes*.

Par exemple, les complexes cellulaires utilisés dans [9, chapitre 12] pour la modélisation de circuits électriques sont topologiques ; aucune géométrie n'est nécessaire. Un hypergraphe [26] peut être représenté par un complexe cellulaire en regroupant les sommets (0-cellules) dans une 2-cellule par l'intermédiaire de 1-

[1] Voir des discussions apparentées dans [25].

cellules. Les hypergraphes sont utilisés en modélisation de solide, par exemple dans [27]. D'ailleurs notre modélisation d'un drapeau[2] utilisant un réseau de masses-ressorts à la section 3.1, n'est pas exclusivement basé sur des cellules ayant une réalisation géométrique.

Nous débutons nos définitions de la cellule et du complexe cellulaire en se basant sur [9]. Certaines conditions seront enlevées à la section 4.1, pour obtenir ce que nous appelons des cellules généralisées. Cela permettra un plus haut degré de généralité, autant pour des situations géométriques ou non géométriques.

Nous introduisons un complexe K pour représenter un objet afin d'identifier les éléments topologiques de base pour le système que nous modélisons. Les éléments topologiques, qui n'ont pour l'instant aucune géométrie, sont indiqués par des cellules. Une cellule est notée p-cellule lorsqu'elle est de dimension p. Nous avons des 0-cellules, des 1-cellules (qui sont composées d'un ensemble de 0-cellules), des 2-cellules (qui sont composées d'un ensemble de 1-cellules), etc. La définition est récursive : une p-cellule est composée d'un ensemble de $p+1$ $(p-1)$-cellules ou plus[3] pour $p \geq 1$ où les 0-cellules sont considérées comme les primitives.

Nous notons les ensembles de p-cellules par S_p et nous définissons qu'un n-complexe K est constitué de $n+1$ ensembles finis $S_0, S_1, \ldots, S_n$ avec un opérateur algébrique *boundary* ∂_p, $0 \leq p \leq n$. L'opérateur ∂_p associe à chaque élément de S_p une certaine combinaison d'éléments de S_{p-1} constituant sa frontière [9, p. 503]. Nous avons omis certains détails de l'opérateur *boundary* et nous reportons les détails concernant l'*orientation* à la section 2.3. Nous supposons aussi que l'intersection entre deux cellules d'un complexe K résulte en aucune ou en une seule cellule dans K. Nous définissons l'intersection en vérifiant les éléments communs qui composent deux cellules. Nous avons à la figure 2.1 (les cellules ne sont pas

[2]Notre modélisation de drapeau est basée sur [3].

[3]Le nombre de cellules de dimension inférieure qui compose une cellule est fini et une 1-cellule est normalement composée de deux 0-cellules distinctes.

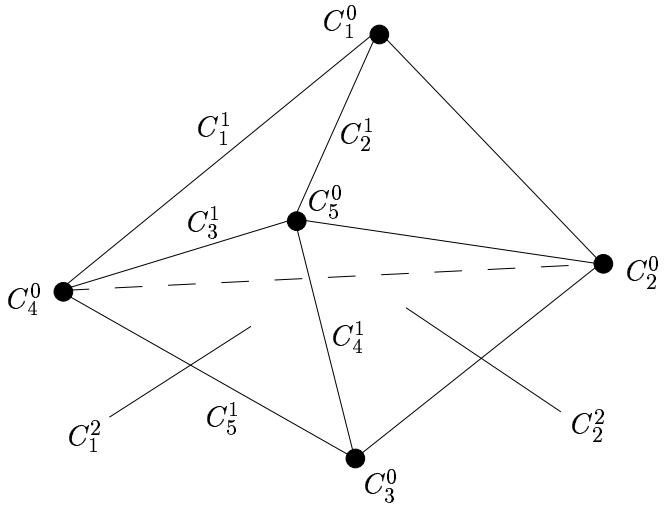

FIG. 2.1 – Exemple d'un 3-complexe valide.

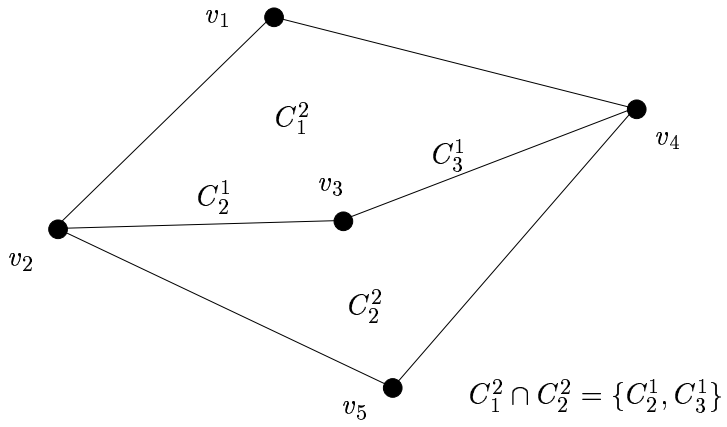

FIG. 2.2 – Exemple d'un 2-complexe invalide.

toutes étiquetées) un 3-complexe valide tandis qu'à la figure 2.2, nous avons un 2-complexe invalide[4].

Lorsque les ensembles $S_0, S_1, \ldots, S_n$ sont définis, nous pouvons référer aux *faces* d'une p-cellule; ce sont les $(p-1)$-cellules dans la frontière de la p-cellule. De façon similaire, lorsque nous référons aux *cofaces* d'une p-cellule; ce sont les $(p+1)$-cellules qui ont dans leur frontière la p-cellule. Avec les relations face-coface de toutes les cellules d'un complexe, nous avons assez d'informations pour savoir les relations d'*adjacence* qui existent entre deux cellules quelconques d'un complexe [1, section 3.1]. En fait, on dit que deux cellules sont adjacentes lorsqu'une des deux cellules fait partie de la frontière de l'autre cellule, ou lorsqu'elle fait partie d'une des frontières de la frontière, ou encore d'une des frontières, des frontières de la frontière, etc. Nous ajoutons à cela que deux p-cellules sont aussi adjacentes lorsqu'elles ont une $(p-1)$-cellule en commun dans leurs faces ou lorsqu'elles ont une $(p+1)$-cellule en commun dans leurs cofaces. À la prochaine section, ce concept sera plus détaillé.

La définition géométrique d'un complexe cellulaire donnée dans [1, section 3.1] est consistante avec notre définition topologique. Le fait d'envisager les complexes d'une façon topologique laisse ouvert beaucoup de questions mathématiques dont plusieurs sont de nature géométrique (par exemple l'utilisation du nombre de Betti [28], ou encore une bouteille de Klein [29] qui serait décrite avec notre complexe topologique, bien qu'il soit connu qu'il n'y a pas de complexe géométrique en R^3 qui corresponde à cet objet). Nous n'allons pas aborder dans cette thèse, ce genre de problème géométrique sur un complexe. Nous laissons à l'usager qui utilise l'interface de programmation, la responsabilité de décider si le complexe doit avoir une réalisation géométrique, et dans un tel cas, s'il y en a une, notons que notre cadre autorise l'usager à construire un n-complexe contenant des cellules

[4] À la figure 2.2, la définition formelle de S_0, S_1, et S_2 sont : $S_0 : C_1^0 = v_1, C_2^0 = v_2, C_3^0 = v_3, C_4^0 = v_4, C_5^0 = v_5$; $S_1 : C_1^1 = \{v_1, v_2\}, C_2^1 = \{v_2, v_3\}, C_3^1 = \{v_3, v_4\}, C_4^1 = \{v_1, v_4\}, C_5^1 = \{v_2, v_5\}, C_6^1 = \{v_4, v_5\}$; $S_2 : C_1^2 = \{C_1^1, C_2^1, C_3^1, C_4^1\}, C_2^2 = \{C_5^1, C_6^1, C_3^1, C_2^1\}$.

de dimension plus petite que n ne faisant pas partie d'une cellule de dimension supérieure (dans [12], on autorise cela aussi). L'usager a en fait, la responsabilité que le complexe topologique qu'il conçoit soit cohérent avec son application (par exemple, lorsqu'il a besoin que le complexe soit une *variété* [28]).

2.2 Méthodes sur les cellules et les complexes cellulaires

Comme nous l'avons mentionné ci-dessus, la construction et la manipulation d'entités telles que des cellules ou des complexes cellulaires, sont faites à partir d'une interface de programmation orientée objet[5]. En particulier, l'interface de programmation inclut une classe *Complex*, avec entre autres, les méthodes suivantes :

- *Complex()*. Constructeur pour la déclaration d'un complexe.
- *int dim()*. Retourne la dimension n (entier) de la cellule de la plus grande dimension du complexe.
- *int nb_cells(int p)*. Retourne le nombre (entier) de cellules de dimension p du complexe.
- *Cell cell(int i, int p)*. Retourne la $i^{\text{ième}}$ cellule de dimension p du complexe.
- *void add(Cell cellp)*. Ajoute la p-cellule identifiée par *cellp* au complexe, à la dernière position (c'est-à-dire, $K.cell(K.nb\_cells(p), p)$ retourne la cellule qui vient d'être ajoutée).

Nous notons $C_i^p = K.cell(i, p)$, soit la $i^{\text{ième}}$ cellule de dimension p du complexe K. Notez qu'ici, et à d'autres endroits dans la thèse, nous utilisons le nom des méthodes comme notation mathématique. Nous aurions pu ajouter de nouvelles

[5]Dans la présentation des méthodes, nous avons omis certains détails de la syntaxe C++ qui ne sont pas importants dans notre contexte.

notations mathématiques correspondantes (par exemple pour la méthode *nb_cells*) mais nous croyons que notre façon de faire, simplifie la notation et ne devrait causer aucune confusion.

L'ordre des p-cellules C_i^p, en fonction de i, dépend de l'ordre dans lequel les p-cellules ont été ajoutées au complexe. Par exemple pour C_4^0, à la figure 2.1, nous savons que c'est la quatrième 0-cellule à avoir été ajoutée au complexe (avec la méthode *add*). Plus loin dans cette section, nous verrons de quelle manière nous créons les cellules, dans quel ordre, et à quel moment nous les ajoutons au complexe.

Avec l'ordre des p-cellules C_i^p, en fonction de i, nous pouvons parler de *somme formelle* des p-cellules avec des coefficients sur K [1, 6], ou de façon équivalente, de vecteurs de coefficients [6, page 226]. Ce concept est développé à la section 2.4.

Nous avons aussi inclus dans notre interface de programmation une classe *Cell*, qui comprend, entre autres, les méthodes suivantes :

- *Cell(int p).* Constructeur[6] pour la déclaration d'une p-cellule.
- *int dim().* Retourne la dimension p (entier) de la cellule.
- *int nb_adj(int delta_p).* Retourne le nombre (entier) de cellules adjacentes de dimension $p+\delta p$, où p est la dimension de l'instance de cellule qui invoque la méthode et où δp est un entier qui est spécifié par le paramètre *delta_p*. Notez que l'on distingue $\delta p = 0^+$ et $\delta p = 0^-$. Cette distinction sera expliquée plus loin dans cette section.
- *Cell adj(int i, int delta_p).* Retourne la $i^{\text{ième}}$ cellule adjacente de dimension $p+\delta p$, où p est la dimension de l'instance de cellule qui invoque la méthode et où δp est un entier qui est spécifié par le paramètre *delta_p*. Encore une fois on distingue $\delta p = 0^+$ et $\delta p = 0^-$. Cette distinction sera expliquée plus loin dans cette section.

[6] Une instance de la classe *Cell* est en fait une *référence* à une cellule de façon à ne conserver qu'une seule copie d'une cellule ayant un compteur de référence.

- *void*[7] *add(Cell cellpm1, int r_o)*. Ajoute[8] la $(p-1)$-cellule identifiée par *cellpm1* à l'instance de la classe *Cell* qui invoque la méthode (une p-cellule). La cellule *cellpm1* doit être déjà ajoutée au même complexe que la cellule qui invoque la méthode. Le paramètre *r_o*, qui est considéré seulement pour la première $(p-1)$-cellule ajoutée, est utilisé pour spécifier l'orientation relative (la valeur par défaut est de $+1$). Ce paramètre est expliqué en détail à la section 2.3.

Voyons comment nous procédons pour la *création* d'une p-cellule. Ceci est fait en trois étapes, qui doivent être effectuées dans l'ordre, et ce, pour une p-cellule en particulier :

- déclaration de la p-cellule à l'aide du constructeur,
- ajout de la p-cellule à un complexe en utilisant la méthode *add* de la classe *Complex*,
- assemblage de la p-cellule en ajoutant des $(p-1)$-cellules à celle-ci avec la méthode *add* de la classe *Cell*. Les $(p-1)$-cellules doivent avoir déjà été ajoutées au même complexe que la p-cellule.

Voici comment nous effectuons la création d'un complexe. On procède en créant les 0-cellules[9], les 1-cellules, pour terminer par les cellules de la plus grande dimension du complexe. Aussitôt qu'une cellule est déclarée, elle doit être ajoutée au complexe avant qu'elle ne soit ajoutée à une autre cellule du même complexe. Par exemple, pour construire le complexe à la figure 2.1, on commence par déclarer les 0-cellules

[7] Nous avons utilisé "void" ici pour indiquer que la méthode ne retourne aucune valeur, contrairement aux autres méthodes que nous avons énumérées ci-dessus.

[8] Lorsque nous ajoutons une $(p-1)$-cellule à une p-cellule, cela doit être considéré dans le sens de concaténer, d'attacher ou d'adjoindre.

[9] En fait, nous commençons par les cellules de la plus petite dimension, qui devraient en général, être des 0-cellules.

en fournissant au constructeur la dimension de la cellule entre parenthèses, soit[10] :

```
Cell C⁰₁(0), C⁰₂(0), C⁰₃(0), C⁰₄(0), C⁰₅(0);
```

Après avoir déclaré les 0-cellules, on les ajoute au complexe K :

```
K.add(C⁰₁);
K.add(C⁰₂);
   ⋮
K.add(C⁰₅);
```

Après avoir ajouté les 0-cellules au complexe K, on voudra peut-être associer des coefficients à celles-ci, comme la position en R^3 par exemple (voir la section 2.6). Pour l'instant, nous allons seulement nous intéresser à la topologie, alors pour les 0-cellules, une simple déclaration et l'ajout de celles-ci au complexe K, termine la création de celles-ci.

Ensuite, on crée les 1-cellules, les 2-cellules, etc. Lorsque l'on crée par exemple la cellule C_1^2 de la figure 2.1, nous aurons les cinq lignes de code suivantes :

```
Cell C²₁(2);
K.add(C²₁);
C²₁.add(C¹₃);
C²₁.add(C¹₄);
C²₁.add(C¹₅);
```

[10]Nous utilisons une `police de caractères avec des espacements constants` lorsque nous écrivons une ligne de code d'un programme. Notez que nous utilisons quand même la notation mathématique lorsque nous référons à une méthode qui n'est pas dans une ligne de code. Il est évident dans l'exemple présenté, qu'on ne peut pas utiliser C_4^0 comme étiquette pour une cellule. Pour être exacte, il aurait fallu utiliser une étiquette comme `C_0_4`, ce qui est moins évident. Alors pour plus de clarté, nous avons choisi des étiquettes pour les cellules plus proches de notre notation mathématique, même si celles-ci sont invalides dans une ligne de code.

Notez qu'avant d'assembler une p-cellule (troisième, quatrième et cinquième lignes de code), il faut l'ajouter au complexe (deuxième ligne de code).

Soulignons que pour l'usager, la déclaration des cellules et l'utilisation de la méthode *add*, peut devenir laborieuse, surtout avec des complexes qui contiennent beaucoup de cellules. Dans ce cas, il serait probablement plus pratique de créer un fichier *texte* décrivant les cellules d'un complexe et un analyseur syntaxique qui ferait la déclaration des cellules et les appels à la méthode *add*. Il serait encore plus intéressant de faire un programme graphique interactif pour créer les cellules d'un complexe. Mais nous ne l'avons pas fait.

Suite à la construction du complexe cellulaire, nous avons besoin d'obtenir des informations de celui-ci ; plus particulièrement, les méthodes adj et nb_adj méritent notre attention. En fait, elles sont une généralisation de face et coface. Dans [1], ils définissent que les faces d'une p-cellule[11] $c \in K$ sont les $(p-1)$-cellules de K dans sa frontière et ils définissent que les cofaces d'une p-cellule $c' \in K$ sont les $(p+1)$-cellules de K dont c' est une face. Lorsque δp est égal à -1, nous avons l'ensemble suivant :

$$\{c.adj(i,-1) : i \in \{1,\ldots,c.nb\_adj(-1)\}\}$$

qui correspond exactement aux faces de c. Lorsque δp est égal à $+1$, nous avons l'ensemble :

$$\{c.adj(i,+1) : i \in \{1,\ldots,c.nb\_adj(+1)\}\}$$

qui correspond exactement aux cofaces de c. Par exemple, pour δp égal à -1 avec le complexe de la figure 2.1, nous avons que pour tout i dans l'intervalle

[11] Dans la thèse, nous notons une cellule de dimension p par : C_i^p (avec un "C" en majuscule) lorsque nous considérons l'ordre de la cellule dans le complexe K. C'est-à-dire que nous avons que $C_i^p = K.cell(i,p)$. Lorsque nous avons une cellule c, avec un "c" minuscule, nous avons une cellule quelconque. Une cellule c^p indique une cellule quelconque de dimension p. Une cellule c_i^p indique une cellule de dimension p avec un ordre autre que l'ordre de la cellule dans le complexe. Nous verrons plus loin en fait que cela indique un ordre d'adjacence.

$1, \ldots, C_1^2.nb\_adj(-1)$, la méthode $C_1^2.adj(i,-1)$ retourne une cellule C_j^1, où $j \in \{3,4,5\}$. L'ordre dans lequel la méthode adj avec δp égal à -1 retourne les $(p-1)$-cellules selon i, correspond exactement à l'ordre dans lequel les $(p-1)$-cellules ont été ajoutées pour assembler la p-cellule. Par exemple, si nous ajoutons les 1-cellules dans le même ordre que nous avions lors de l'exemple de la création de la cellule C_2^1 ci-dessus, nous avons que $C_1^2.adj(1,-1)$ retourne la cellule C_3^1, $C_1^2.adj(2,-1)$ retourne la cellule C_4^1 et $C_1^2.adj(3,-1)$ retourne la cellule C_5^1.

Nous allons maintenant détailler le fonctionnement de l'algorithme utilisé pour les méthodes $adj(i,\delta p)$ et $nb\_adj(\delta p)$ lorsque δp est différent de $+1$ et -1. Nous introduisons d'abord une notation mathématique pour simplifier la description de l'algorithme. Cette notation représente un vecteur de cellules et nous définissons $adj(\delta p)$ par :

$$c^p.adj(\delta p) = (c^p.adj(1,\delta p), \ldots, c^p.adj(c^p.nb\_adj(\delta p), \delta p))$$

Nous allons calculer $c^p.adj(\delta p)$ et par le fait même, montrer le fonctionnement des méthodes $adj(i,\delta p)$ et $nb\_adj(\delta p)$. Pour comprendre cet algorithme, il faut savoir comment sont emmagasinées les informations lorsqu'on ajoute une $(p-1)$-cellule à une p-cellule.

Pour chacune des p-cellules C_i^p d'un n-complexe, pour $1 \leq p \leq n$, il y a d'emmagasiné un vecteur $C_i^p.v_f$ de $(p-1)$-cellules et pour $0 \leq p \leq n-1$, un vecteur $C_i^p.v_c$ de $(p+1)$-cellules. Ce sont respectivement les faces et les cofaces de la p-cellule. Ces vecteurs sont mis à jour par la méthode *add* de la classe *Cell*.

Alors pour une p-cellule c^p d'un complexe, nous avons respectivement que $c^p.adj(-1) = c^p.v_f$ et que $c^p.adj(+1) = c^p.v_c$. Examinons maintenant ce qui survient lorsqu'une $(p-1)$-cellule c^{p-1} est ajoutée à une p-cellule c^p. D'abord, on ajoute à la fin du vecteur $c^p.v_f$ la cellule c^{p-1}, c'est-à-dire que[12]

[12] On place entre "[ ]" l'indice du $i^{\text{ième}}$ élément du vecteur et on a que *length* est égal au nombre d'éléments du vecteur.

$c^p.v_f[(c^p.v_f).length + 1] \leftarrow c^{p-1}$. Ensuite, on ajoute à la fin du vecteur $c^{p-1}.v_c$ la cellule c^p, c'est-à-dire que $c^{p-1}.v_c[(c^{p-1}.v_c).length + 1] \leftarrow c^p$.

Maintenant, nous sommes en position pour trouver $c^p.adj(\delta p)$ lorsque δp est différent de $+1$ et -1. Nous avons un algorithme récursif pour cela, il est montré à la figure 2.3. Il utilise l'union $\sqcup$ entre deux vecteurs v_1 et v_2 définie comme suit :

$v_1 \sqcup v_2$. Les cellules de v_2 sont ajoutées à la fin de v_1, par contre, lorsqu'une cellule de v_2 est dans v_1, elle n'est pas ajoutée.

Remarquez que l'opération $\sqcup$ n'est pas commutative.

En examinant la définition de cette opération et l'algorithme de la figure 2.3, nous remarquons que l'ordre des cellules adjacentes qui sont retournées dans un vecteur de cellules, dépend de l'ordre dans lequel la cellule qui invoque la méthode, et ses cellules adjacentes, ont été assemblées. Par exemple, une 2-cellule avec laquelle nous invoquons la méthode *adj(1,-2)* retourne une cellule qui est la première 0-cellule qui a été ajoutée à la première 1-cellule qui a été ajoutée à la 2-cellule.

Examinons de plus près le traitement qui est effectué lorsque δp est égal à 0^+ (qui est spécifié avec $delta\_p = 100$) et lorsque δp est égal à 0^- (qui est spécifié avec $delta\_p = -100$). Ces valeurs de δp ont été ajoutées pour des raisons de complétude. Nous voulons en fait obtenir les p-cellules adjacentes à la p-cellule c^p qui invoque la méthode. Lorsque δp vaut 0^-, les p-cellules qui partagent une $(p-1)$-cellule avec c^p sont retournées et lorsque δp vaut 0^+, les p-cellules qui partagent une $(p+1)$-cellule avec c^p sont retournées.

Nous avons que $adj(\delta p)$ retourne un vecteur de cellules. Ainsi, la méthode $adj(i, \delta p)$ retourne $adj(\delta p)[i]$ et la méthode $nb\_adj(\delta p)$ retourne $adj(\delta p).length$.

Nous nous sommes inspirés de [12, page 156] pour trouver des propriétés à $adj(i, \delta p)$ et $adj(\delta p)$: pour toutes cellules b et c d'un complexe K, nous avons que :

$$b = c.adj(i, \delta p) \implies \text{il existe un } j \text{ unique, tel que } c = b.adj(j, -\delta p)$$

$V \leftarrow \emptyset$ (V est un vecteur de cellules);
si $\delta p = -1$ **alors**
  $c^p.adj(\delta p) \leftarrow c^p.v_f$;
si $\delta p = +1$ **alors**
  $c^p.adj(\delta p) \leftarrow c^p.v_c$;
si $\delta p < -1$ **alors**
  pour $i = 1$ *à* $(c^p.v_f).length$ **faire**
    $V \leftarrow V \sqcup (c^p.v_f[i]).adj(\delta p + 1)$;
  $c^p.adj(\delta p) \leftarrow V$;
si $\delta p > +1$ **alors**
  pour $i = 1$ *à* $(c^p.v_c).length$ **faire**
    $V \leftarrow V \sqcup (c^p.v_c[i]).adj(\delta p - 1)$;
  $c^p.adj(\delta p) \leftarrow V$;
si $\delta p = 0^-$ **alors**
  pour $i = 1$ *à* $(c^p.v_f).length$ **faire**
    $V \leftarrow V \sqcup (c^p.v_f[i]).adj(+1)$;
  $c^p.adj(\delta p) \leftarrow V$ sur lequel on enlève la cellule c^p;
si $\delta p = 0^+$ **alors**
  pour $i = 1$ *à* $(c^p.v_c).length$ **faire**
    $V \leftarrow V \sqcup (c^p.v_f[i]).adj(-1)$;
  $c^p.adj(\delta p) \leftarrow V$ sur lequel on enlève la cellule c^p;

FIG. 2.3 – Algorithme récursif afin de calculer $c^p.adj(\delta p)$.

et aussi que :

c est une composante de $b.adj(\delta p) \Longleftrightarrow b$ est une composante de $c.adj(-\delta p)$.

Notez que nous fixons comme convention que $-0^+ = 0^+$ et que $-0^- = 0^-$ pour que ces propriétés soit valides. Cette convention est aussi utilisée plus loin dans la thèse.

Nous pouvons maintenant préciser la relation d'adjacence entre deux cellules : lorsque nous disons que deux cellules b et c sont adjacentes, c'est qu'il existe un δp, tel que :

c est une composante de $b.adj(\delta p)$.

Nous croyons que l'usager utilisera rarement la méthode $adj(i, \delta p)$, à part peut-être pour vérifier des propriétés topologiques du complexe cellulaire. Le fait par contre de généraliser les opérateurs *face* et *coface* permettra de généraliser, par le fait même, les opérateurs de chaînes *boundary* et *coboundary* (voir la section 2.4).

2.3 Cellules orientées

Dans beaucoup d'applications, il est important de pouvoir spécifier l'*orientation* des cellules ; en particulier, ce concept s'avère nécessaire pour définir l'opérateur *boundary* ∂_p, introduit à la section 2.1. Pour une 1-cellule, l'orientation correspond au concept d'une flèche, et pour une 2-cellule, au concept de sens horaire et de sens anti-horaire (voir la figure 2.4). Le concept peut être donné formellement avec des p-simplexes, avec un p arbitraire [6, page 222]. Pour un complexe, lorsque celui-ci peut être décomposé en p-simplexes [13], le concept peut aussi être donné formellement, mais il est difficile et coûteux d'utiliser cette approche pour des cellules de grande dimension. Une méthode simple, basée sur l'orientation relative, sera décrite ci-dessous.

Une motivation (certainement pas la seule) pour introduire l'opérateur *boundary* est la possibilité de donner des conditions algébriques qui correspondent à notre idée intuitive d'une condition nécessaire sur les cellules d'un complexe, à savoir que chacune des frontières forme un cycle [9, page 504]. De manière informelle, cela veut dire que la frontière d'une 2-cellule (des 1-cellules) forme une boucle composée de segments dont les extrémités se rejoignent. Et la frontière d'une 3-cellule (des 2-cellules) devrait se joindre pour former des éléments de surface contiguë et sans trou (comme la surface d'un ballon de soccer). Veuillez noter, par contre, que notre système n'impose pas cette condition.

L'opérateur *boundary* peut être vu comme un opérateur linéaire sur un espace vectoriel avec des vecteurs dont les composantes sont indexées par les éléments de S_p [9]. Un élément de S_p est identifié avec un vecteur ayant le nombre un à la position correspondant à cet élément et avec un zéro aux autres positions. L'opérateur *boundary* ∂_p assigne à chacun des éléments de S_p, la somme des vecteurs dans S_{p-1} correspondant à la frontière des éléments de S_p, où chaque terme est positif ou négatif selon l'orientation relative de S_p avec l'élément particulier de la frontière [9, page 503].

L'orientation relative est calculée à partir d'une p-cellule et d'une $(p-1)$-cellule. Lorsque la $(p-1)$-cellule n'a pas déjà été ajoutée à la p-cellule, l'orientation relative est de 0. Lorsque l'orientation de la $(p-1)$-cellule est dans le "même sens" que la p-cellule, l'orientation relative est de $+1$, et est de -1 sinon. Dans notre cadre de travail, nous utilisons la méthode *orientation* définie sur la classe *Cell*. Voici sa déclaration :

- *int orientation(Cell cellpm1).* Retourne l'orientation relative (0, $+1$ ou -1) entre la p-cellule qui invoque la méthode et la $(p-1)$-cellule identifiée par *cellpm1*.

Ainsi à la figure 2.4, nous avons que :

$C_1^2.orientation(C_1^1) = +1.$

$C_1^2.orientation(C_2^1) = -1.$

$C_1^2.orientation(C_5^1) = 0.$

Sur la figure, nous avons les orientations des cellules qui sont dessinées. Voyons maintenant comment le calcul de l'orientation relative est fait avec la méthode *orientation*.

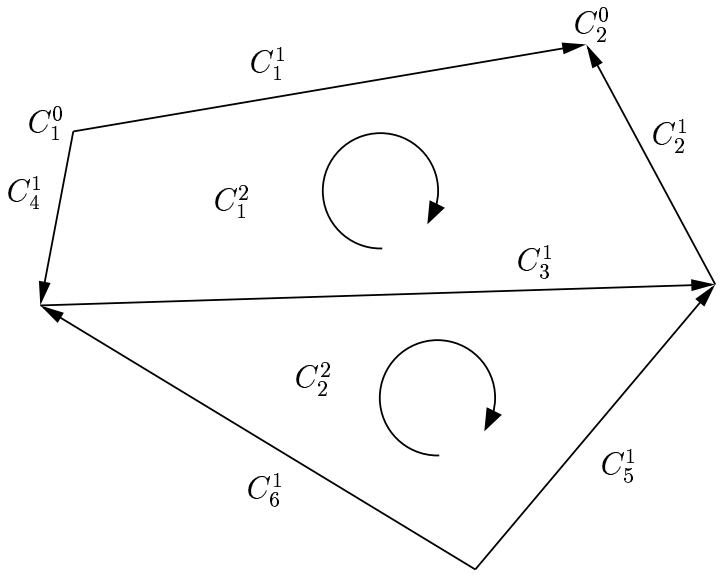

FIG. 2.4 – 2-complexe avec des cellules orientées.

D'abord, nous fixons que l'orientation relative de la première $(p-1)$-cellule ajoutée à une p-cellule est de $+1$. Ainsi lorsque nous faisons :

```
Cpi.add(Cp-1m);
```

nous avons alors que $C_i^p.orientation(C_m^{p-1})$ vaut $+1$ lorsque C_m^{p-1} est la première $(p-1)$-cellule ajoutée à C_i^p.

Nous pouvons, par contre, décider que l'orientation relative de la première $(p-1)$-cellule ajoutée à une p-cellule est de -1. Il suffit de faire :

$\texttt{C}_\texttt{i}^\texttt{p}\texttt{.add}(\texttt{C}_\texttt{m}^{\texttt{p}-\texttt{1}}, -\texttt{1});$

notez que le deuxième paramètre de la méthode *add* qui indique l'orientation relative entre les deux cellules est optionnel et qu'il vaut $+1$ par défaut. Notez aussi que ce paramètre est ignoré lorsqu'une $(p-1)$-cellule n'est pas la première ajoutée à une p-cellule.

La $j^{\text{ième}}$ cellule ajoutée à la cellule C_i^p correspond à la cellule $C_i^p.adj(j,-1)$, pour tout j dans l'intervalle $1,\ldots,C_i^p.nb\_adj(-1)$. Notons ici une cellule $C_i^p.adj(j,-1)$ par c_j^{p-1}. Nous venons de voir comment fonctionne l'orientation relative entre C_i^p et c_j^{p-1} lorsque j est égal à un, examinons maintenant comment calculer celle-ci lorsque j est plus grand que un. Pour pouvoir calculer l'orientation relative, nous supposons qu'entre les faces d'une cellule c_j^{p-1} et les faces de la cellule c_{j+1}^{p-1}, il y a une $(p-2)$-cellule unique[13]. Nous notons cette cellule partagée c_k^{p-2}. Remarquez que si l'on ne satisfait pas à cette condition, le cadre de travail peut quand même être utile pour une application n'utilisant pas des cellules orientées. Remarquez aussi que nous supposons que les cellules c_j^{p-1}, c_{j+1}^{p-1}, et c_k^{p-2} ont déjà été créées.

Lorsque ce n'est pas la première $(p-1)$-cellule ajoutée, nous avons une façon récursive de calculer l'orientation relative de C_i^p avec une $(p-1)$-cellule : l'orientation relative avec c_{j+1}^{p-1} est la même qu'avec c_j^{p-1} lorsque leur orientation relative avec c_k^{p-2} est opposée, et l'orientation relative avec c_{j+1}^{p-1} est l'inverse d'avec c_j^{p-1} lorsque leur orientation relative avec c_k^{p-2} est la même. Autrement dit, l'orientation relative entre C_i^p et c_{j+1}^{p-1} avec $j \geq 1$ est définie comme suit :

[13]Il n'y a pas de détection d'erreurs.

si $c_{j+1}^{p-1}.orientation(c_k^{p-2}) = c_j^{p-1}.orientation(c_k^{p-2})$ **alors**
  $C_i^p.orientation(c_{j+1}^{p-1}) \leftarrow -C_i^p.orientation(c_j^{p-1})$;
sinon
  $C_i^p.orientation(c_{j+1}^{p-1}) \leftarrow C_i^p.orientation(c_j^{p-1})$;

Cette façon de calculer l'orientation relative est illustrée à la figure 2.5. Les carrés du haut de la figure représentent c_j^{p-1}, et les carrés du bas représentent c_{j+1}^{p-1}.

Lorsque l'on calcule l'orientation relative entre une 1-cellule et une 0-cellule, nous n'avons évidemment pas de cellule c_k^{p-2} (de dimension -1) qui existe. Afin que l'orientation relative puisse être calculée dans ce cas, nous créons une cellule de dimension -1, soit $\Lambda = C_1^{-1}$. Cette (-1)-cellule est formellement ajoutée à toutes les 0-cellules et a une orientation relative de $+1$ avec celles-ci[14].

En fixant comme convention que les 0-cellules sont orientées positivement, nous n'avons qu'à fixer les orientations relatives, comme à la figure 2.6, pour avoir l'équivalent des orientations de la figure 2.4.

Maintenant que nous savons comment le système calcule les orientations relatives, il est facile de construire le complexe en choisissant l'ordre dans lequel nous ajoutons les $(p-1)$-cellules à une p-cellule. Voici comment nous procédons pour construire le complexe des figures 2.4 et 2.6. Nous déclarons et ajoutons au complexe les 0-cellules, les 1-cellules et les 2-cellules. Ensuite, nous assemblons les 1-cellules, par exemple la cellule C_1^1 en ajoutant en premier à celle-ci la 0-cellule C_2^0, et en deuxième la 0-cellule C_1^0. L'assemblage des 2-cellules est aussi très simple. Par exemple, pour assembler la cellule C_1^2, nous ajoutons une première 1-cellule avec laquelle nous voulons une orientation relative de $+1$, soit C_1^1 (lorsque nous désirons

[14]Dans notre système, la (-1)-cellule n'est pas réellement créée et ajoutée à toutes les 0-cellules, mais notre algorithme d'orientation relative effectue les calculs comme si elle l'avait été.

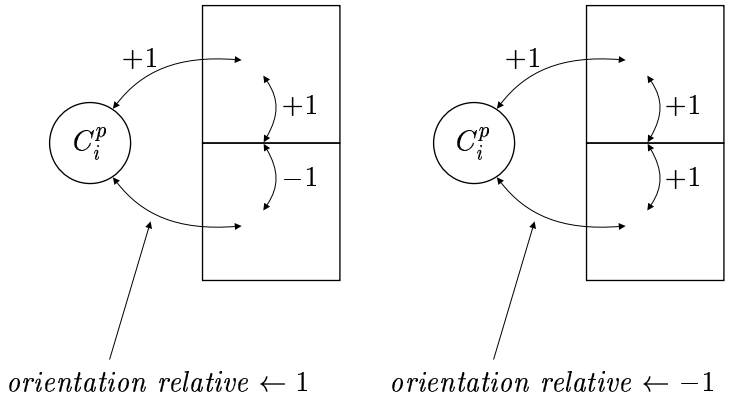

FIG. 2.5 – Calcul de l'orientation relative.

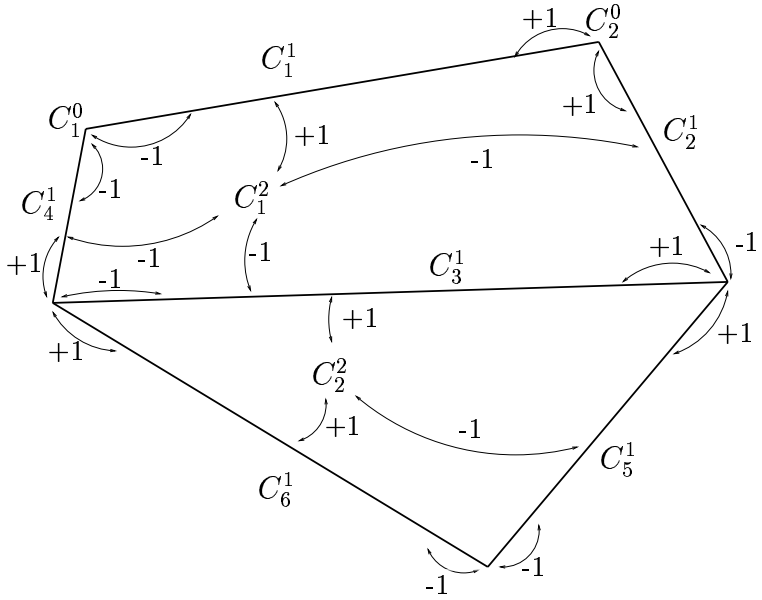

FIG. 2.6 – 2-complexe dont les orientations sont relatives.

que toutes les 1-cellules aient une orientation relative de -1 avec la 2-cellule, il faut, lors de l'ajout de la première 1-cellule, spécifier que l'orientation relative est de -1 : voir plus haut dans cette section). Ensuite nous ajoutons les autres 1-cellules à la 2-cellule C_1^2 en prenant soin d'avoir une 0-cellule qui est partagée par deux 1-cellules ajoutées successivement. Nous avons donc ici deux possibilités, soit d'ajouter dans l'ordre C_2^1, C_3^1, C_4^1, ou bien d'ajouter dans l'ordre C_4^1, C_3^1, C_2^1. Avec un choix ou l'autre, nous obtenons les mêmes orientations relatives (remarquez par contre, que l'ordre des cellules adjacentes est modifié).

Nous aurons la même logique à appliquer lors de la création d'une p-cellule en général. Voici un autre exemple, cette fois avec une 3-cellule C_i^3 dont l'orientation relative avec des 2-cellules est illustrée à la figure 2.7. Les carrés du haut représentent la première 2-cellule c_1^2 ajoutée, et ceux du bas, représentent la deuxième 2-cellule c_2^2 ajoutée. Lorsque les orientations relatives des deux 2-cellules avec la 1-cellule partagée sont opposées (premier cas de la figure 2.7), nous avons que l'orientation relative entre C_i^3 et c_2^2 est la même que l'orientation relative entre C_i^3 et c_1^2 ($+1$ dans la figure). Lorsque les orientations relatives des deux 2-cellules avec la 1-cellule partagée sont identiques (deuxième cas de la figure 2.7), nous avons que l'orientation relative entre C_i^3 et c_2^2 est l'inverse (-1) que l'orientation relative entre C_i^3 et c_1^2 ($+1$).

La façon qu'a notre système de gérer les orientations respecte la manière habituelle de concevoir des entités topologiques orientées, sauf qu'il utilise seulement des orientations relatives. En travaillant seulement avec celles-ci, nous n'avons pas à donner une caractérisation algébrique et algorithmique d'un sens horaire ou antihoraire dans un plan, ou ce qui est encore plus difficile, une caractérisation d'une orientation dans une cellule de plus grande dimension. En fait, l'orientation absolue n'est pas intrinsèque à une cellule, car elle est essentiellement arbitraire, et en changeant sa définition, nous ne faisons que changer son interprétation. Pour donner un exemple simple et plus spécifique, dans un contexte d'un coefficient (attribut) associé avec une p-cellule d'un complexe, supposons que $p = 1$, que $cell1$ est un

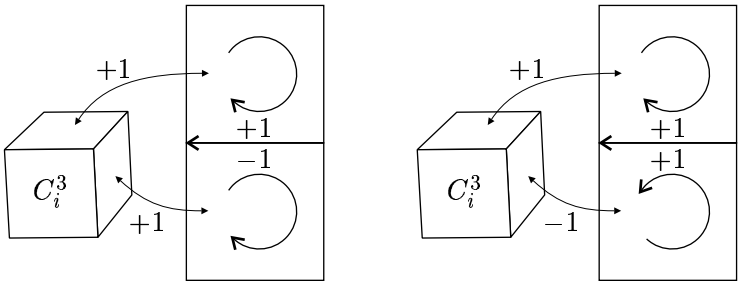

FIG. 2.7 – Orientation relative dans une 3-cellule.

identificateur d'un segment de ligne (1-cellule), et que le coefficient est un vecteur de force $\mathbf{f} \in R^3$ dirigé en direction de la 0-cellule qui a une orientation relative de $+1$ avec la 1-cellule : si nous voulons spécifier une expression qui donne une force $\mathbf{f}$ dans la direction d'une 0-cellule *cell*0, qui est un identificateur d'une des deux 0-cellules de la 1-cellule, il est suffisant de multiplier $\mathbf{f}$ par $cell1.orientation(cell0)$ (rappelons que la méthode *orientation* retourne une orientation relative).

Pour des complexes de dimension 2, le calcul de l'orientation relative est équivalent à calculer les nombres d'incidences (α_{ij} et β_{ij}) [28, section 3] qui sont nécessaires à l'utilisation du nombre de Betti dans les théorèmes mathématiques classiques.

Comme nous l'avons vu, un avantage de laisser notre système calculer l'orientation relative est qu'il est très facile de construire un complexe orienté et de choisir les orientations relatives entre les cellules. Dans un système utilisant des orientations relatives qui sont fixées par l'usager [1], il est laborieux pour celui-ci de fixer chacune des orientations relatives. Dans un tel système, l'usager fixe

de façon arbitraire l'orientation relative lors de chacun des ajouts d'une $(p-1)$-cellule à une p-cellule. Par exemple, à la figure 2.6, il devra spécifier tous les -1 et les $+1$. Par contre, s'il choisit de fixer l'orientation relative entre la cellule C_2^2 et C_1^3 à $+1$ (comme dans la figure) mais qu'il fixe entre la cellule C_2^2 et C_5^1 aussi à $+1$ tout en conservant toutes les autres orientations relatives illustrées dans la figure, nous aurions alors que notre définition d'orientation relative ne sera pas respectée pour les 1-cellules C_3^1 et C_5^1. Car ayant les mêmes orientations relatives avec C_2^2 et partageant une 0-cellule, elles devraient avoir des orientations opposées avec cette 0-cellule. Comme nous l'avons vu, l'usager peut quand même fixer dans notre système les orientations relatives de son choix, mais la variété de ses choix est limitée par notre définition de l'orientation relative. Il ne pourra pas créer une situation qui ne respecte pas notre définition d'orientation relative comme dans l'exemple précédent.

Remarquez que dans beaucoup d'applications, l'usager ne veut pas vraiment spécifier une orientation particulière aux cellules (par exemple dans le réseau de masses-ressorts à la section 3.1), il désire seulement que le complexe soit orienté, ce qu'il obtient avec notre système en respectant qu'entre deux $(p-1)$-cellules ajoutées successivement à une p-cellule, une seule $(p-2)$-cellule soit partagée entre les deux $(p-1)$-cellules. Par contre, comme avant, nous laissons la responsabilité à l'usager de s'assurer que les cellules qu'il construit dans le complexe soient appropriées. Par exemple, il est facile de construire une bande de Möbius avec trois 2-cellules orientées mais cela ne peut pas correspondre à une surface orientée.

Un autre avantage d'avoir des orientations relatives, comme dans notre système, est qu'une application voulant économiser de l'espace mémoire n'est pas obligée d'emmagasiner les orientations relatives. Comme nous l'avons vu, elles sont calculées. Par contre, cela peut demander beaucoup de temps de calcul pour une application utilisant beaucoup cette information, mais il est alors évidemment possible de les emmagasiner. Nous n'allons d'ailleurs pas les emmagasiner directement dans la structure de données des cellules, mais plutôt dans une chaîne de

sous-chaînes d'entiers (voir la sous-section 4.3.5).

2.4 Chaînes

Une p-chaîne est définie sur un complexe K et un ensemble G. C'est une somme formelle

$$\sum_i g_i C_i^p$$

sur les p-cellules C_i^p du complexe K, avec les coefficients $g_i \in G$. Une p-chaîne assigne un élément de G à chacune des p-cellules $C_i^p = K.cell(i,p)$ dans le complexe K [1, section 3.2]. Un exemple d'une 2-chaîne est illustré à la figure 2.8. Dans notre cadre de travail, l'ensemble G peut être par exemple un ensemble de réels, un ensemble de vecteurs, un ensemble de couleurs dans l'espace RGB, un ensemble de polynômes, et même, comme nous allons le voir à la section 3.2, un ensemble de p-cellules.

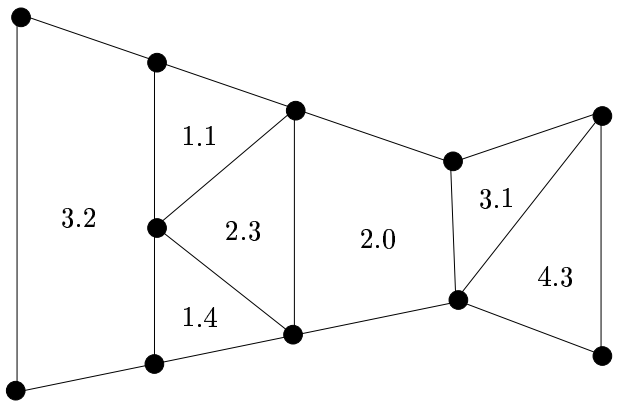

FIG. 2.8 – Exemple d'une 2-chaîne.

Nous pouvons par exemple dans un réseau de masses-ressorts (que nous al-

lons décrire plus en détail à la sous-section 3.1.1), avoir une 0-chaîne de réels représentant la masse aux 0-cellules et une 1-chaîne sur l'ensemble des vecteurs dans R^3 représentant les forces élastiques des ressorts associées aux 1-cellules.

On peut faire une correspondance entre la somme formelle et un vecteur de coefficients [6, page 226], soit :

$$\sum_i g_i C_i^p \quad \longleftrightarrow \quad (g_1, g_2, \ldots, g_i, \ldots, g_n), \quad n = K.nb\_cells(p),$$

où $nb\_cells(p)$ retourne le nombre de cellules de dimension p du complexe K.

2.4.1 Opérations et méthodes sur les chaînes

Nous avons inclus dans notre interface de programmation la classe *Chain*, qui comprend, entre autres les méthodes suivantes :

- *int dim()*. Retourne la dimension p (un entier) de la chaîne. C'est-à-dire, la dimension p des cellules sur laquelle la chaîne qui invoque la méthode est définie.
- *Complex complex()*. Retourne le complexe K (de la classe *Complex*) sur lequel la chaîne qui invoque la méthode est basée.

La classe de base *Chain* a des classes dérivées *Chain_Real*, *Chain_Real2* et *Chain_Real3*, qui désignent respectivement des chaînes sur R, R^2 et R^3. Pour plus de détails, voir l'annexe B. Le programmeur pourra facilement créer d'autres classes dérivées pour avoir des chaînes sur d'autres types de coefficient. Une classe dérivée typique *Chain_Type* a le constructeur suivant :

- *Chain_Type(int p, Complex complex_K)*. Constructeur pour une chaîne ayant des coefficients du type *Type*. Requiert les paramètres d'entrées p, la dimension de la chaîne et *complex_K* (de la classe *Complex*), le complexe sur lequel la chaîne est définie.

En plus du constructeur, une classe *Chain_Type* inclut, entre autres, les méthodes :

- *Type& coefficient(int i).* Retourne la référence au coefficient g_i (de la classe *Type*) associé à la cellule $K.cell(i,p)$, où K est le complexe sur lequel la chaîne qui invoque la méthode est basée, et avec p étant la dimension de la chaîne.
- *Type& coefficient(Cell cell).* Retourne la référence au coefficient g_i (de la classe *Type*) associé à la cellule *cell* (de dimension p), où i est tel que $cell = K.cell(i,p)$, K est le complexe sur lequel la chaîne qui invoque la méthode est basée, et avec p étant la dimension de la chaîne.

En utilisant la dernière méthode mentionnée, il est possible d'accéder au coefficient associé à une cellule en particulier d'une chaîne. Par exemple, si *chain2* est un identificateur pour une 2-chaîne ayant des coefficients de type *Type* et *cell2*, étant un identificateur d'une 2-cellule, alors nous pouvons affecter une valeur de la classe *Type* à *chain2.coefficient(cell2).* Ainsi, par exemple avec *Real3(1,2,2)*, une instance de l'objet vecteur $[1,2,2]$ de la classe *Real3* (vecteurs en R^3), nous pouvons écrire :

$$\texttt{chain2.coefficient(cell2)} = \texttt{Real3(1,2,2)}.$$

Nous allons habituellement écrire une expression comme celle à gauche de l'affectation sous sa forme abrégée : `chain2[cell2]`. Notez que la forme abrégée est aussi valide dans notre interface de programmation.

Nous avons, comme dans [1, définition 13], que les méthodes[15] m qui sont définies sur les éléments de l'ensemble G, le seront aussi sur une chaîne définie sur le même ensemble G. Symboliquement, nous définissons que :

$$(\sum_i g_i C_i^p).m = \sum_i g_i.m\ C_i^p.$$

[15]Dans [1], ils ont utilisé des fonctions plutôt que des méthodes.

Autrement dit, lorsque nous appliquons une méthode qui est définie sur les éléments d'un ensemble G, à une chaîne, nous appliquons cette méthode à tous les coefficients de celle-ci.

Par exemple, soit une p-chaîne (désignée par l'identificateur *chainp*) définie sur un complexe K et une méthode *meth* définie sur les coefficients de la chaîne, lorsque nous écrivons :

`chainp.meth`();

il en résultera que :

$$chainp[cell].meth()$$

sera appliquée à tous les coefficients associés aux p-cellules *cell* du complexe K.

Nous pouvons aussi définir cela sur la chaîne sous sa forme de vecteur de coefficients :

$$chainp.meth() \quad \longleftrightarrow \quad (g_1.meth(),\ \ldots,\ g_n.meth())$$

où *chainp* est une chaîne de dimension p ayant des coefficients de l'ensemble G.

Lorsqu'une opération binaire $\otimes$ est définie entre les éléments d'un ensemble G_a et les éléments d'un ensemble G_b, nous avons que l'opération binaire $\otimes$ entre une chaîne *chainp_Ga* ayant des coefficients de l'ensemble G_a et une chaîne *chainp_Gb* ayant des coefficients de l'ensemble G_b est définie comme suit :

$$chainp\_Ga \otimes chainp\_Gb \quad \longleftrightarrow \quad (g_1 \otimes h_1,\ \ldots,\ g_i \otimes h_i,\ \ldots,\ g_n \otimes h_n),$$

avec les g_i étant tous les coefficients de la chaîne *chainp_Ga* et les h_i étant tous les coefficients de la chaîne *chainp_Gb*. Notez que les deux chaînes doivent être de même dimension p et être définies sur le même complexe K.

Alors si nous écrivons :

`chainp_Ga` $\otimes$ `chainp_Gb`;

nous aurons que le coefficient associé à la $i^{\text{ième}}$ p-cellule *cellp* de la chaîne résultante sera de :

$$chainp\_Ga[cellp] \otimes chainp\_Gb[cellp],$$

où $cellp = K.cell(i, p)$.

Cela nous permet de décrire de façon concise, une addition (ou une soustraction, une multiplication, une équivalence, ...) cellule par cellule des coefficients des chaînes. Prenons un exemple plus pratique, soit deux 1-chaînes sur R^3, *chain1_a* et *chain1_b*. Supposons que l'opération binaire $\otimes$ entre les deux 1-chaînes soit le produit scalaire, noté $*$ dans notre interface de programmation, il en résultera alors une chaîne de réels (que nous emmagasinons dans *chain1_real*). Nous pouvons donc écrire :

`chain1_real` = `chain1_a` $*$ `chain1_b`;

qui effectuera, cellule par cellule, les produits scalaires suivants :

$$chain1\_real[cellp] = chain1\_a[cellp] * chain1\_a[cellp].$$

Rappelons que les deux p-chaînes peuvent être définies sur deux ensembles de coefficients différents. Par exemple, une première p-chaîne peut être définie sur les réels et une deuxième, sur R^3. La multiplication entres celles-ci résulterait alors en une p-chaîne sur R^3.

2.5 Relation entre les chaînes de différentes dimensions

Une des méthodes les plus puissantes développées dans notre cadre de travail est la méthode *dim_inc*. Celle-ci nous permet d'obtenir, à partir d'une chaîne de départ, une chaîne de dimension inférieure ou supérieure. Comme nous l'avons

mentionné à la section 1.3, *dim_inc* généralise les opérations *boundary* et *co-boundary* sur une chaîne. Elle permet d'établir des relations entre des chaînes de différentes dimensions définies sur un complexe K en tenant compte ou non de l'orientation relative.

Montrons avec un exemple le fonctionnement de cette méthode. Pour l'instant, nous ne tenons pas compte de l'orientation relative. Nous voyons à la figure 2.9 une 1-chaîne d'entiers (huit) définie sur un 2-complexe. Nous appliquons la méthode *dim_inc* sur cette 1-chaîne et nous indiquons que nous voulons une chaîne de dimension relative +1 (c'est-à-dire, une 2-chaîne). Nous indiquons aussi que nous voulons l'opération d'addition. Cette méthode effectuera, pour chacune des 2-cellules, l'addition des coefficients de la 1-chaîne dont les 1-cellules font partie de la 2-cellule. Le processus est répété pour toutes les 2-cellules et les résultats des additions sont conservés dans une 2-chaîne.

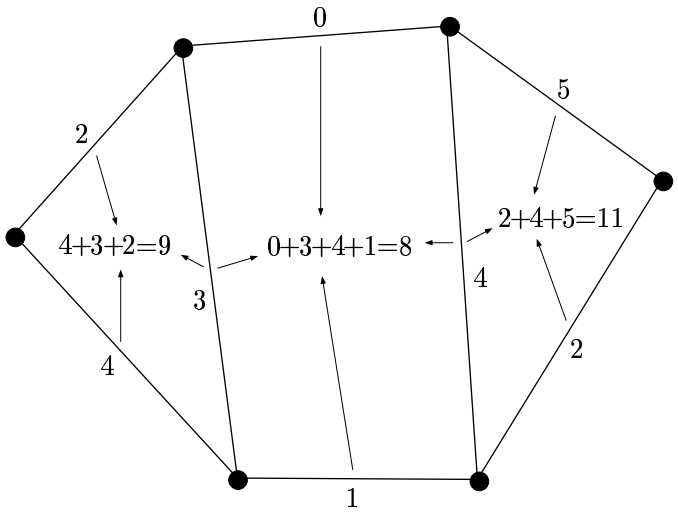

FIG. 2.9 – *dim_inc* appliquée sur une 1-chaîne qui résulte en une 2-chaîne.

La méthode *dim_inc* peut être appliquée sur n'importe quelle p-chaîne. Nous indiquons une dimension relative δp et il en résultera alors en une q-chaîne ($q =$

$p + \delta p$). Il faut par contre que $0 \leq p, q \leq n$, avec n étant la dimension du complexe sur lequel sont définies les deux chaînes ; ici $0 + 0^- < 0$ et $n + 0^+ > n$. Nous avons un autre exemple à la figure 2.10. Cette fois nous débutons avec une 2-chaîne pour obtenir une 0-chaîne, encore une fois sans tenir compte de l'orientation relative, et avec une opération d'addition. Dans cet exemple, la dimension relative est de -2.

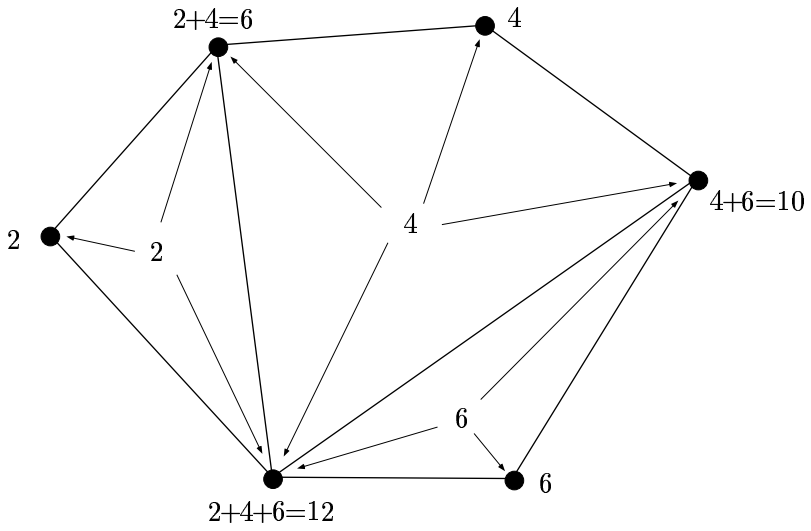

FIG. 2.10 – *dim_inc* appliquée sur une 2-chaîne qui résulte en une 0-chaîne.

Nous pouvons aussi tenir compte de l'orientation relative des cellules. Un exemple est illustré à la figure 2.11. Ici, nous appliquons la méthode *dim_inc* sur une 2-chaîne pour une dimension relative de -1 avec l'opération d'addition et en tenant compte de l'orientation relative. Ceci résultera en une 1-chaîne. La méthode effectuera pour chacune des 1-cellules, l'addition des coefficients de la 2-chaîne dont les 2-cellules sont adjacentes à la 1-cellule et en multipliant chacun des coefficients par l'orientation relative de la 2-cellule et de la 1-cellule impliquées. Le processus est répété pour toutes les 1-cellules et les résultats des additions sont conservés dans une 1-chaîne.

Nous pouvons tenir compte de l'orientation relative, seulement lorsque la di-

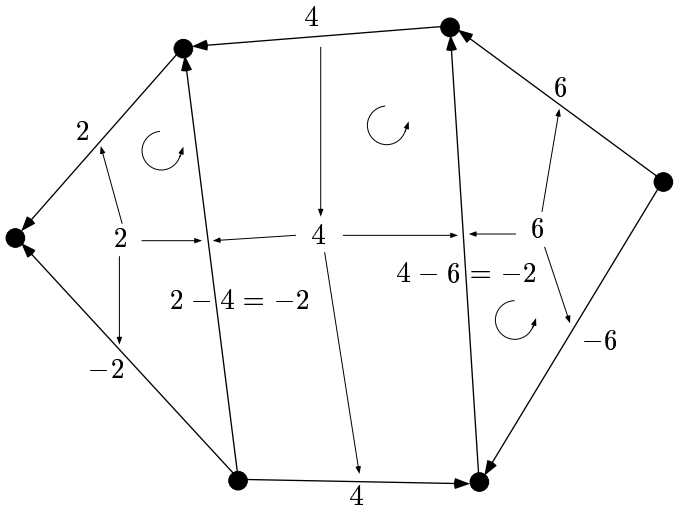

FIG. 2.11 – En tenant compte de l'orientation relative, *dim_inc* est appliquée sur une 2-chaîne qui résulte en une 1-chaîne.

mension relative δp est de $+1$ ou -1 car l'orientation relative est définie seulement entre des cellules dont la différence de dimension est de 1. Lorsque l'on débute avec une p-chaîne pour obtenir une $(p-1)$-chaîne, cette opération est appelée *boundary* sur une chaîne [1]. Lorsque l'on débute avec une p-chaîne pour obtenir une $(p+1)$-chaîne, cette opération est appelée *coboundary* sur une chaîne [1]. À travers les exemples du chapitre 3, nous démontrons l'utilité de *dim_inc* lorsque nous tenons compte de l'orientation, mais aussi lorsque nous n'en tenons pas compte, ce qui dans [1] n'avait pas été envisagé et qui peut être très utile. Il faut dire que nous n'avons pas limité l'opération entre les coefficients d'une chaîne qu'à l'addition, elle peut être aussi remplacée par une opération commutative.

La méthode *dim_inc* d'une classe *Chain_Type* a la description suivante :

- *Chain_Type dim_inc(int delta_p, Operation op, Boolean is_signed).* Retourne une chaîne (de la classe *Chain_Type*) de dimension $p + \delta p$ où δp est un

entier spécifié par *delta_p* (la dimension relative). L'opération[16] entre les coefficients est spécifiée par *op* et l'orientation relative est considérée lorsque *is_signed* est à *true*.

Operation est un type énuméré, qui permet de spécifier l'addition (*plus*), la moyenne (*average*), la multiplication (*mult*), la moyenne géométrique (*geom_mean*), ou d'autres opérations commutatives ajoutées par le programmeur. Lorsqu'elles ne sont pas spécifiées, les valeurs par défaut de *op* et *is_signed* sont respectivement de *plus* et *true*. Nous pouvons, par exemple écrire :

$$chain3 = chain1.dim\_inc(+2, average, false);$$

pour avoir une 3-chaîne à partir d'une 1-chaîne (la dimension relative est de $3-1 = +2$), où les coefficients de la 3-chaîne sont obtenus en faisant la moyenne des coefficients associés aux 1-cellules pertinentes, sans tenir compte de l'orientation relative[17].

En général, si une chaîne qui invoque la méthode *dim_inc* est $\sum_j g_j C_j^p$, les valeurs de j faisant une contribution à chacun des coefficients h_i associé à la cellule $C_i^{p+\delta p}$ dans la nouvelle chaîne, sont celles, tel que :

$$C_j^p = C_i^{p+\delta p}.adj(k, -\delta p) \quad \text{pour certains } k.$$

Plus formellement, la méthode *dim_inc* retourne la $(p+\delta p)$-chaîne $\sum_i h_i C_i^{p+\delta p}$, où les cellules $C_i^{p+\delta p}$ sont définies par $K.cell(i, p+\delta p)$, pour des valeurs de i allant de 1 à $K.nb\_cells(p+\delta p)$. Les valeurs de h_i lorsque nous ne tenons pas compte de l'orientation relative sont :

$$h_i = \Omega_{k=1}^{m_i} g_{j_k}$$

où

[16] L'opération doit être valide pour les coefficients de la chaîne qui invoque la méthode.

[17] C'est en fait le seul choix possible lorsque la dimension relative est de +2.

- Ω est défini par le paramètre *op* (par exemple, $\Omega = \Sigma$, Π, ..., l'opération doit être commutative) ;
- m_i est égal à $C_i^{p+\delta p}.nb\_adj(-\delta p)$;
- la séquence d'indices j_1, j_2, ... , j_{m_i} est définie pour chacune des valeurs j_k tel que : $C_{j_k}^p = C_i^{p+\delta p}.adj(k, -\delta p)$ pour un $k \in \{1, \ldots, m_i\}$.
- $g_{j_k} = chainp.coefficient(\ C_{j_k}^p)$. (Ici, *chainp* est la chaîne qui invoque la méthode *dim_inc*.)

De façon similaire, les valeurs de h_i lorsque nous tenons compte de l'orientation relative sont :

$$h_i = \Omega_{k=1}^{m_i} \sigma_{ij_k} g_{j_k}$$

où

$$\sigma_{ij_k} = C_i^{p+\delta p}.orientation(C_{j_k}^p) \text{ si } \delta p = 1,$$
$$\sigma_{ij_k} = C_{j_k}^p.orientation(C_i^{p+\delta p}) \text{ si } \delta p = -1,$$

et les autres quantités étant définies comme précédemment. Tel que déjà mentionné, lorsque nous tenons compte de l'orientation relative *dim_inc* est définie seulement pour $\delta p \in \{-1, 1\}$.

2.6 Géométrie sur les complexes

Notre cadre de travail est utile pour des complexes géométriques, tels que ceux utilisés dans [1, 12]. Comme il est dit dans [1, section 3.2], les cellules et les complexes cellulaires peuvent être utilisés afin de décomposer R^3 en régions plus simples et les chaînes peuvent être utilisées pour associer des quantités physiques avec ces régions. En fait, ce sera le cas des applications principales de notre cadre de travail. Cependant, nous avons choisi de mettre l'emphase sur le fait que les complexes cellulaires représentent une topologie sur un objet ou un système. Nous avons deux raisons pour cela. Premièrement, nous avons observé que les chaînes

et les complexes peuvent être utilisés dans des applications où il n'y a pas de géométrie, par exemple, pour représenter la topologie d'un réseau de communication ou circuit électrique [9, chapître 12]. Deuxièmement, il y a des applications où nous voulons représenter la géométrie d'un système à l'aide de chaînes. La géométrie étant représentée dans des chaînes, elle peut être manipulée de la même façon que les informations non géométriques.

Ces remarques motivent notre approche pour la définition de la géométrie sur un complexe. Habituellement, la géométrie sur un complexe est définie avant même d'avoir défini les chaînes (pas toujours en fait, [1, section 5]). Nous avons, par contre, choisi d'utiliser les chaînes afin de représenter la géométrie des objets.

Dans un contexte de programmation objet, nous créons une nouvelle classe représentant une géométrie particulière pour un complexe. Cette nouvelle classe est une classe dérivée de la classe *Complex* car elle conserve ses propriétés topologiques. Nous y ajouterons certaines chaînes qui seront des membres de cette nouvelle classe, afin de représenter la géométrie. Nous y ajouterons aussi des méthodes pour travailler avec cette nouvelle classe.

Décrivons deux nouvelles classes dérivées de la classe *Complex* illustrant différentes géométries :

1. Supposons que nous voulions représenter un système composé de sphères dans R^3 ayant des interactions entre elles (le type d'interaction n'est pas important ici). Les 1-cellules et des 1-chaînes pourraient représenter ces interactions et des 0-chaînes représenteraient la géométrie. Les sphères seraient centrées sur des points en R^3 associés aux 0-cellules, c'est-à-dire dans une 0-chaîne. Les rayons des sphères seraient aussi emmagasinés dans une 0-chaîne qui dans ce cas, serait une chaîne de réels. Nommons cette nouvelle classe représentant cette géométrie *Complex_s*.

2. Supposons que nous voulions représenter des surfaces triangulaires en deux dimensions dans R^3. (Dans ce cas, chacune des 2-cellules du complexe aura

trois 1-cellules adjacentes.) Ici, la géométrie serait conservée dans une 0-chaîne de points en R^3, représentant les sommets des triangles et dans une 2-chaîne représentant les vecteurs normaux (de façon redondante) aux surfaces. Nous référerons plus tard (entre autres, à la sous-section 3.1.1) à cette classe comme étant *Complex_p*.

Une autre classe dérivée de la classe *Complex* pourrait être définie, par exemple, avec des surfaces B-splines cubiques. Chacune de ces classes dérivées contiennent des méthodes reliées à la géométrie du complexe. Par exemple, une méthode appropriée à toutes les classes dérivées représentant une géométrie dans R^3 est une méthode déterminant l'intersection entre un rayon (ou un axe) et la géométrie du complexe.

Nous avons ici la possibilité de plusieurs choix de géométries en dérivant de nouvelles classes. Par contre, nous voulons mettre l'emphase sur le fait que nous représentons la géométrie sur un complexe à l'aide de chaînes. Nous avons alors à notre disposition pour manipuler ces chaînes (donc la géométrie) tous les outils qui sont disponibles pour les chaînes en général. Dans l'exemple avec des masses et des ressorts au chapitre 4, il y a une chaîne représentant la géométrie (les positions des 0-cellules) qui est intégrée, au même titre que les autres chaînes, aux expressions de chaînes décrivant le système.

Chapitre 3

Exemples de systèmes

Dans ce chapitre, nous présentons deux exemples illustrant l'utilité du cadre de travail. Le premier exemple montre comment le cadre de travail peut être utilisé pour décrire une simulation d'un réseau de masses-ressorts en résolvant des équations différentielles du premier ordre avec la méthode de Euler. Nous développons ensuite cet exemple pour la modélisation de tissus, qui est une application basée sur les réseaux de masses-ressorts. Le second exemple illustre une application totalement différente : le cadre de travail est utilisé pour décrire un algorithme classique de subdivision de surface (Catmull-Clark). Dans les deux cas, nous obtenons de très bons résultats, mais le but de ce chapitre est de montrer comment il est facile de décrire ces exemples en utilisant notre cadre de travail.

3.1 Réseau de masses-ressorts et modélisation de tissus

3.1.1 Réseau de masses-ressorts

Notre premier exemple implique un modèle physique assez simple mais qui démontre tout de même la puissance de notre cadre de travail. Nous voulons simuler

un objet contenant des masses reliées par des ressorts. Ce type d'objet est utilisé, par exemple pour modéliser des objets flexibles ou pour la modélisation de tissus [3]. Un cas particulier de modélisation de tissus, soit un drapeau, est présenté à la prochaine sous-section.

Nous utilisons un complexe de dimension 1 (1-complexe) pour représenter la topologie du réseau de ressorts reliant les masses. On associe les masses aux 0-cellules et les ressorts aux 1-cellules. Différentes chaînes seront définies sur le complexe afin de représenter les différentes quantités dans le système et leurs évolutions.

La classe[1] décrivant la géométrie sera décrite avec plus de précisions à la fin de cette sous-section et à la sous-section suivante (sous-section 3.1.2), mais disons pour l'instant qu'elle contient une 0-chaîne sur R^3 donnant les positions $\vec{p}$ des 0-cellules. Nous avons une instance de cette classe que nous notons *network* pour désigner l'identificateur du réseau de masses-ressorts. Alors, supposons que la géométrie a été définie et que nous avons cette 0-chaîne :

- $network.chain0\_p$ sur R^3 (positions $\vec{p}$ des 0-cellules).

Le processus général permettant de simuler le réseau de masses-ressorts, se déroule comme suit : nous calculons toutes les forces (les forces d'amortissement[2] des ressorts $\vec{f}_d$, la force gravitationnelle $\vec{f}_g$, les forces élastiques des ressorts $\vec{f}_s$ et la force totale $\vec{f}$), pour chacune des 0-cellules afin de calculer l'accélération $\vec{a}$ à chacune des 0-cellules. Nous effectuons ensuite les mises à jour des positions $\vec{p}$ et des vitesses $\vec{v}$ aux 0-cellules en utilisant la méthode de Euler. Pour ce faire, nous avons besoin des sept 0-chaînes suivantes :

- $chain0\_f\_damp$ sur R^3 (forces d'amortissement des ressorts $\vec{f}_d$ appliquées sur chacune des 0-cellules) ;

[1] Dans un contexte de programmation objet.

[2] Étant donné que plusieurs ressorts peuvent être attachés à une masse, il n'y a pas une, mais plusieurs forces d'amortissement qui influenceront une masse à une 0-cellule. Même remarque concernant les forces élastiques des ressorts.

- *chain*0_*f*_*g* sur R^3 (force gravitationnelle $\vec{f_g}$ appliquée sur chacune des 0-cellules) ;
- *chain*0_*f*_*spring* sur R^3 (forces élastiques des ressorts $\vec{f_s}$ appliquées sur chacune des 0-cellules) ;
- *chain*0_*f* sur R^3 (force totale $\vec{f}$ appliquée sur chacune des 0-cellules) ;
- *chain*0_*m* sur $R^+ = \{x \geq 0 : x \in R\}$ (la masse m à chacune des 0-cellules) ;
- *chain*0_*a* sur R^3 (accélération $\vec{a}$ à chacune des 0-cellules) ;
- *chain*0_*v* sur R^3 (vitesse $\vec{v}$ à chacune des 0-cellules).

Afin d'évaluer les forces physiques aux ressorts, nous avons également besoin de sept quantités associées à chacun de ceux-ci : trois scalaires, k_s (constante d'élasticité du ressort), l (longueur du ressort au repos), k_d (constante d'amortissement du ressort), ainsi que trois vecteurs dans R^3, $\vec{e}$ (étirement du ressort), $\vec{x}$ (étirement relatif du ressort, c'est-à-dire, la différence entre l'étirement courant et l'étirement au repos), $\vec{f_s}$ (force élastique du ressort), et $\vec{f_d}$ (force d'amortissement du ressort).

Nous avons alors besoin des trois 1-chaînes définies sur R^+ suivantes :

- *chain*1_*ks* (constante de ressort k_s pour chacune des 1-cellules) ;
- *chain*1_*l* (longueur l du ressort au repos pour chacune des 1-cellules) ;
- *chain*1_*kd* (constante d'amortissement du ressort k_d pour chacune des 1-cellules) ;

ainsi que quatre 1-chaînes définies sur R^3 :

- *chain*1_*e* (étirement du ressort $\vec{e}$ pour chacune des 1-cellules) ;
- *chain*1_*x* (étirement relatif du ressort $\vec{x}$ pour chacune des 1-cellules) ;
- *chain*1_*f*_*spring* (force élastique du ressort $\vec{f_s}$ pour chacune des 1-cellules) ;
- *chain*1_*f*_*damp* (force d'amortissement du ressort $\vec{f_d}$ pour chacune des 1-cellules).

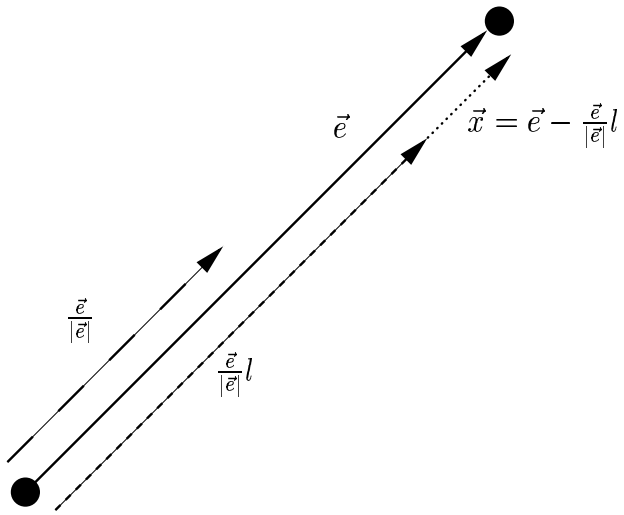

FIG. 3.1 – Étirement d'un ressort.

Avec ces chaînes ainsi définies et avec notre interface de programmation[3], nous pouvons maintenant décrire tout le processus itératif afin de simuler le réseau de masses-ressorts. Notez comment *dim_inc* utilise adéquatement l'orientation relative afin d'assigner correctement les différentes quantités aux étapes 1, 4, 5 et 6 ci-dessous. Par exemple, à l'étape 1, pour chacune des paires de 0-cellules reliées par une 1-cellule, la méthode *dim_inc* fait une soustraction entre les deux coefficients (positions) et le résultat est conservé dans le coefficient associé à la 1-cellule (dans la chaîne *chain1_e*). Le terme de la soustraction qui est soustrait de l'autre est déterminé par l'orientation relative. En fait, pour chacune des paires de 0-cellules reliées par une 1-cellule, la méthode *dim_inc* additionne le coefficient qui est associé avec la 0-cellule qui a une orientation relative de $+1$ avec la 1-cellule, et soustrait le coefficient qui est associé avec la 0-cellule qui a une orientation relative de -1 avec la 1-cellule.

[3]Il y a aussi dans le code la méthode *normalize* de la classe *Chain_Real3* qui retourne une chaîne dont chacun des vecteurs est normalisé. L'opération unaire "−" est aussi définie sur la classe *Chain_Real* : retourne une chaîne dont chacun des réels est inversé.

1. `chain1_e` = `network.chain0_p.dim_inc`$(+1)$;

   Calcule l'allongement du ressort $\vec{e}$ pour chacune des 1-cellules en soustrayant les positions $\vec{p}$ associées aux 0-cellules reliées par la 1-cellule.

2. `chain1_x` = `chain1_e` $-$ (`chain1_e.normalize`$() *$ `chain1_l`) ;

   Calcule l'allongement relatif du ressort $\vec{x}$ pour chacune des 1-cellules (voir la figure 3.1) : $\vec{x} = \vec{e} - \frac{\vec{e}}{|\vec{e}|} l$.

3. `chain1_f_spring` $= -$`chain1_ks` $*$ `chain1_x` ;

   Calcule la force élastique du ressort $\vec{f_s} = -k_s \vec{x}$ pour chacune des 1-cellules.

4. `chain1_f_damp` = `chain0_v.dim_inc`$(+1) *$ `chain1_kd` ;

   Calcule la force d'amortissement du ressort $\vec{f_d}$ à chacune des 1-cellules en calculant la vitesse relative des vitesses associées aux 0-cellules reliant la 1-cellule, et en multipliant par la constante d'amortissement à la 1-cellule.

5. `chain0_f_spring` = `chain1_f_spring.dim_inc`(-1) ;

   Cumule les forces élastiques des ressorts $\vec{f_s}$ sur chacune des 0-cellules, en prenant les forces élastiques des ressorts $\vec{f_s}$ associées aux 1-cellules ayant la 0-cellule dans sa frontière.

6. `chain0_f_damp` = `chain1_f_damp.dim_inc`(-1) ;

   Cumule les forces d'amortissement des ressorts $\vec{f_d}$ sur chacune des 0-cellules, en prenant les forces d'amortissement des ressorts $\vec{f_d}$ associées aux 1-cellules ayant la 0-cellule dans sa frontière.

7. `chain0_f` = `chain0_f_spring` + `chain0_f_g` + `chain_0_f_damp` ;

   Calcule la force totale $\vec{f}$ à chacune des 0-cellules en additionnant les forces élastiques des ressorts, les forces d'amortissement des ressorts et la force gravitationnelle : $\vec{f} = \vec{f_s} + \vec{f_d} + \vec{f_g}$.

8. `network.chain0_p` = `network.chain0_p` + `chain_0_v` $*$ `Chain_Real`(`delta_t`) ;

   Met à jour la position[4] à chacune des 0-cellules : $\vec{p} = \vec{p} + \Delta t\, \vec{v}$.

[4]Notez que "*Chain_Real*(*delta_t*)" représente une chaîne dont tous les coefficients sont égaux à Δt.

9. `chain0_v` = `chain0_v` + `chain0_a` * `Chain_Real(delta_t)` ;

   Met à jour la vitesse à chacune des 0-cellules : $\vec{v} = \vec{v} + \Delta t\ \vec{a}$.

10. `chain0_a` = `chain0_f` / `chain0_m`;

    Met à jour l'accélération à chacune des 0-cellules : $\vec{a} = \vec{f}/m$.

11. `t` = `t` + `delta_t` ;

    Passe à la tranche de temps suivante.

Ce programme, qui utilise notre interface de programmation, démontre que nous pouvons exprimer de façon claire et concise un problème relativement complexe. Développer, modifier et corriger un programme équivalent sans notre interface de programmation, est une tâche ardue et parfois très frustrante, car nous avons à travailler avec du code de bas niveau qui nous éloigne de la tâche principale.

Nous avons ajouté une détection de collisions des positions aux 0-cellules avec un plancher (sans entrer dans les détails, ceci est très facilement réalisé). En définissant une géométrie au complexe (par exemple avec la classe *Complex_s*, voir section 2.6), nous obtenons des objets, plus ou moins flexibles, selon la rigidité des ressorts, qui "rebondissent" sur le plancher. Un tétrahèdre gardera approximativement sa forme, mais pas un cube (voir la section 5.1 pour résoudre ce cas).

3.1.2 Drapeau

Une des approches pour modéliser les tissus utilise comme base un réseau de masses-ressorts. Nous illustrons ici cette approche en continuant l'exemple du réseau de masses-ressorts de la sous-section précédente, tout en montrant la puissance de notre cadre de travail. Nous modélisons un cas particulier de tissu, un drapeau flottant au vent. Le modèle est très semblable à celui présenté dans [3]. Ce n'est pas un modèle physique exact, mais il est suffisamment précis pour générer des animations très réalistes [30].

Nous utilisons un réseau de masses-ressorts dont la structure est illustrée à la

figure 3.2. Nous remarquons dans cette figure les carrés qui connectent les sommets (masses). Les arêtes de ces carrés représentent les ressorts horizontaux et verticaux, ils ont pour but d'empêcher que le tissu (réseau) ne s'étire, ni ne se compresse excessivement. Les lignes diagonales, illustrées avec des lignes en pointillé, représentent les ressorts de cisaillement (*shear springs*). Leur but est de limiter le cisaillement du tissu. Et enfin, il y a d'illustrées à cette figure, les lignes courbes qui rejoignent deux sommets, en laissant un sommet entre les deux. Elles représentent les ressorts de flexion qui empêchent le tissu de se replier excessivement sur lui-même. Bien que les lignes courbes représentent les ressorts de flexion, ils ne doivent pas être vus comme étant des 1-cellules courbes dans le complexe géométrique. En fait, beaucoup des ressorts ne seront pas représentés dans le complexe géométrique.

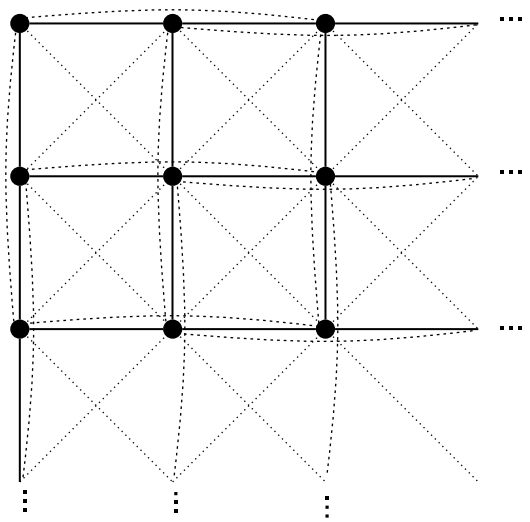

FIG. 3.2 – Réseau de masses-ressorts pour le drapeau.

Au réseau de masses-ressorts tel que décrit, nous ajoutons des 2-cellules triangulaires afin d'avoir une géométrie de surface pour le drapeau. Nous utilisons le complexe géométrique *Complex_p* (voir la section 2.6) pour la géométrie du drapeau. Pour créer les 2-cellules, nous prenons quatre 1-cellules formant les carrés

de la figure 3.2 et un des ressorts de cisaillement (lignes diagonales) pour former deux 2-cellules triangulaires ; le processus est répété pour chacun des carrés. Nous utilisons des 2-cellules triangulaires afin de s'assurer que les positions des 0-cellules de chacune des 2-cellules soient coplanaires. Le vecteur normal à chacune des 2-cellules (nous supposons que les positions des 0-cellules ne sont pas colinéaires) est prédéfini dans la chaîne :

- *network.chain2_n* sur R^3 (vecteur normal à la surface $\vec{n}$ pour chacune des 2-cellules).

Les 2-cellules sont aussi utilisées pour simuler l'action de l'air sur la surface du drapeau. Pour modéliser cela, nous avons besoin des 2-chaînes suivantes :

- *chain2_f_air* sur R^3 (la force $\vec{f_a}$ exercée par l'air sur la surface de chacune des 2-cellules) ;
- *chain2_kf* sur R (une constante k_f quantifiant la friction entre l'air et la surface pour chacune des 2-cellules) ;
- *chain2_v_surf* sur R^3 (la vitesse $\vec{v}_s$ de la surface, définie comme étant la vitesse moyenne des vitesses des trois 0-cellules de chacune des 2-cellules).

Nous avons aussi besoin de la 0-chaîne suivante :

- *chain0_f_air* sur R^3 (les forces $\vec{f_a}$ exercées par l'air sur les surfaces pour chacune des 0-cellules).

Pour calculer la force de l'air sur les surfaces, nous utilisons la formule de [3] (dans nos notations) :

$$\vec{f_a} = k_f(\vec{n} \cdot (\vec{v}_w - \vec{v}_s))\vec{n},$$

où $\vec{v}_w$ désigne la vitesse courante du vent. Nous voulons appliquer cette équation sur chacune des 2-cellules. Avec notre interface de programmation, nous pouvons faire cela directement :

```
chain2_f_air = chain2_kf * (network.chain2_n*
                            (chain2_v_wind - chain2_v_surf)) * network.chain2_n.
```

Dans cette expression, la chaîne `Chain_Real3(v_wind)` est affectée à la chaîne `chain2_v_wind`, avec `v_wind` étant égal à la vitesse courante du vent $\vec{v}_w$ lorsque la vitesse du vent est uniforme dans R^3.

La vitesse d'une surface est calculée comme étant la vitesse moyenne des 0-cellules qui la composent. Pour faire cela, nous utilisons *dim_inc* avec $\delta p = +2$, sans tenir compte de l'orientation relative et avec Ω égal à $\frac{1}{3}\sum_{k=1}^{3}$:

$$\texttt{chain2\_v\_surf} = \texttt{chain0\_v.dim\_inc}(+2, \texttt{average}, \texttt{false});$$

(dans le programme, cette expression devra évidemment précéder l'expression pour calculer `chain2_f_air`.)

La prochaine étape consiste à cumuler les forces $\vec{f}_s$, à chacune des 0-cellules qui sont appliquées aux 2-cellules adjacentes. Dans ce cas, le paramètre δp pour la méthode *dim_inc* est de -2, et Ω correspond à Σ :

$$\texttt{chain0\_f\_air} = \texttt{chain2\_f\_air.dim\_inc}(-2, \texttt{plus}, \texttt{false});$$

et nous ajoutons ensuite cette chaîne à l'expression de chaînes cumulant les différentes sortes de force à la ligne 7 de la simulation du réseau de masses-ressorts vue à la sous-section précédente :

$$\texttt{chain0\_f} = \texttt{chain0\_f\_spring} + \texttt{chain0\_f\_g} + \texttt{chain\_0\_f\_damp} + \texttt{chain\_0\_f\_air}.$$

Finalement, afin que le drapeau soit fixé au poteau, nous affectons des vitesses nulles aux points d'attache. Ceci est effectué grâce au mécanisme de sélection, qui est détaillé à la section 4.2.

Les expressions de chaînes que nous avons données ici avec les chaînes décrivant le réseau de masses-ressorts, suffisent à décrire et à simuler un drapeau flottant au vent. Cela démontre la simplicité et l'utilité du cadre de travail. Il y a deux remarques importantes que nous voulons souligner ici. Premièrement, l'exemple

décrit ci-dessus n'aurait pas pu être décrit avec les opérateurs de chaînes *boundary* et *coboundary* car la méthode *dim_inc* avec $\delta p = +2$ n'est pas équivalente à appliquer deux fois la méthode *dim_inc* avec $\delta p = +1$, que l'on tienne compte ou non de l'orientation relative. Deuxièmement, le complexe que nous avons utilisé ici, n'est pas totalement géométrique : en particulier, si nous modélisons les 1-cellules de façon géométrique, les ressorts de flexion pourraient certainement avoir des intersections avec les ressorts formant les carrés et nous aurions obtenu un complexe géométrique invalide selon la définition habituelle [1]. La possibilité de combiner des informations géométriques et non géométriques de manière uniforme est un des avantages principaux de notre cadre de travail.

Remarquez que nous avons aussi fait quelques expérimentations pour simuler une feuille de papier. Nous avons utilisé exactement le même modèle mais en affectant des constantes de ressorts beaucoup plus élevées (ressorts plus rigides). En utilisant le mécanisme de sélection, nous avons contraint certaines 0-cellules à avoir certaines positions qui varient dans le temps. Encore une fois, nous avons pu constater qu'il était très facile de modifier un modèle décrit avec des chaînes afin d'obtenir une nouvelle application.

3.2 Catmull-Clark

Dans cette section, nous allons décrire un processus très différent de celui de la section précédente ; encore une fois, en utilisant notre formalisme et notre interface de programmation. Il s'agit d'une méthode de subdivision de surface récursive, soit la méthode de Catmull-Clark [4]. Par contre, nous n'allons pas décrire le processus avec autant de détails qu'à la section 3.1.

La méthode de Catmull-Clark débute avec un maillage arbitraire, qui peut être vu comme étant un 2-complexe contenant des 0-cellules, des 1-cellules et des 2-cellules. Chacune des frontières des 2-cellules est composée de 1-cellules formant une boucle fermée, c'est-à-dire, qui forme un cycle [9, p. 504].

Le but de la méthode de Catmull-Clark est de calculer une surface qui fait une approximation d'un maillage de départ. Une étape de subdivision de la méthode permet de raffiner le maillage en ajoutant des 0-cellules, des 1-cellules et des 2-cellules pour ainsi obtenir un nouveau maillage. Celui-ci peut être à nouveau subdivisé, jusqu'à ce que le maillage soit suffisamment fin selon les besoins de l'application. Ensuite le complexe est affiché avec une géométrie semblable à la classe *Complex_p* afin de représenter la géométrie de la surface, qui contient, nous le rappelons, les positions des 0-cellules. L'instance de cette classe est nommée *mesh*. Nous avons alors une 0-chaîne $mesh.chain0\_p$, définie sur R^3, de vecteurs $\vec{p} \in R^3$ précisant les positions des 0-cellules. Nous notons cette chaîne simplement $chain0\_p$ dans cette section.

Nous expliquons brièvement comment l'on procède pour subdiviser un maillage et obtenir un nouveau maillage plus fin (voir la figure 3.3). Le processus de subdivision se déroule comme suit : nous gardons les 0-cellules du maillage courant (*anciens points*) et nous ajoutons de nouvelles 0-cellules (*nouveaux points*). À la figure 3.3, il y a un gros carré (2-cellule) subdivisé en quatre plus petits. Les coins du gros carré sont les anciens points. Il y a aussi des nouveaux points, classés dans deux catégories, les *points sur les faces* (il y en a cinq sur la figure 3.3 dont deux sont pointés avec des flèches) et les *points sur les arêtes* (il y en a quatre sur la figure dont deux sont pointés avec des flèches). Le *point milieu* indiqué sur la figure 3.3, sera utilisé temporairement lors d'une étape de subdivision.

Les nouveaux *points sur les faces*, un par 2-cellule, sont calculés en faisant la moyenne des anciens points (coins) de chacune des 2-cellules. Ces valeurs sont facilement calculées et emmagasinées dans une 2-chaîne sur R^3 :

$$\texttt{chain2\_face\_points} = \texttt{chain0\_p.dim\_inc}(+2, \texttt{average}, \texttt{false}).$$

Ici, nous avons utilisé la méthode *dim_inc* avec une dimension relative de +2, qui emmagasine les valeurs pour chacune des 2-cellules en calculant la moyenne des positions des 0-cellules adjacentes. Puisque $op = \texttt{average}$, l'opération Ω (section

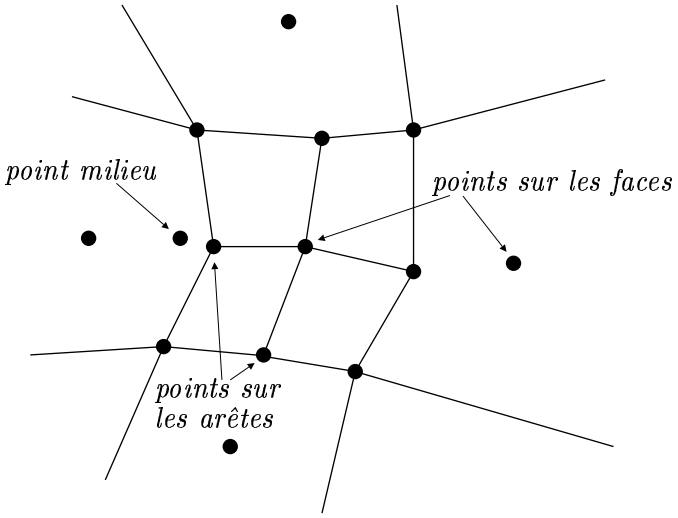

FIG. 3.3 – Subdivision de Catmull-Clark.

2.5) est égale à $\frac{1}{m_i}\sum_{k=1}^{m_i}$. Le paramètre *is_signed* est à `false` car considérer l'orientation relative n'est pas pertinent dans ce cas.

Nous devons maintenant calculer un *point milieu* par 1-cellule. Celui-ci est calculé en faisant la moyenne des positions des 0-cellules qui la compose :

$$\texttt{chain1\_midpoints} = \texttt{chain0\_p.dim\_inc}(+1, \texttt{average}, \texttt{false});$$

ici, l'opération Ω est égale à $\frac{1}{2}\sum_{k=1}^{2}$.

Chacun des *points sur les arêtes* associés aux 1-cellules est calculé en deux étapes. Premièrement, nous calculons la moyenne (que nous nommons α) des deux nouveaux *points sur les faces* qui sont associés aux 2-cellules partageant la 1-cellule. Ensuite nous calculons la moyenne entre α et le *point milieu* que nous venons de décrire. Dans notre interface de programmation, nous calculons d'abord :

$$\texttt{chain1\_average\_face\_points} = \texttt{chain2\_face\_points.dim\_inc}(-1, \texttt{average}, \texttt{false});$$

ensuite, nous calculons :

$$\texttt{chain1\_edge\_points} = (\texttt{chain1\_midpoints} + \texttt{chain1\_average\_face\_points})/\texttt{Chain\_Real}(2.0).$$

Toutes ces chaînes sont définies sur R^3.

Les positions associées aux 0-cellules doivent maintenant être mises à jour. Nous avons besoin pour chacune des 0-cellules, du nombre de 1-cellules adjacentes. Nous affectons d'abord à une 1-chaîne sur les réels, la valeur 1.0 pour chacun des coefficients :

$$\texttt{chain1\_one} = \texttt{Chain\_Real}(1.0);$$

ensuite nous pouvons calculer les nombres (réels) de 1-cellules qui sont adjacentes aux 0-cellules :

$$\texttt{chain0\_nb\_1cells} = \texttt{chain1\_one.dim\_inc}(-1, \texttt{plus}, \texttt{false}). \tag{3.1}$$

Ensuite, nous avons besoin pour chacune des 0-cellules, de la moyenne des *points sur les faces* des 2-cellules adjacentes :

$$\texttt{chain0\_Q} = \texttt{chain2\_face\_points.dim\_inc}(-2, \texttt{average}, \texttt{false});$$

et la moyenne des *points milieux* des 1-cellules adjacentes :

$$\texttt{chain0\_R} = \texttt{chain1\_midpoints.dim\_inc}(-1, \texttt{average}, \texttt{false}).$$

Nous pouvons maintenant mettre à jour les positions des 0-cellules, tel que spécifié dans [4] :

$$\texttt{chain0\_pos} = (\texttt{chain0\_Q} + \texttt{Chain\_Real}(2.0) * \texttt{chain0\_R} + \texttt{chain0\_pos} * (\texttt{chain0\_nb\_1cells} - \texttt{Chain\_Real}(3.0)))/\texttt{chain0\_nb\_1cells}.$$

Après avoir trouvé les positions des *nouveaux points* et avoir mis à jour les positions des 0-cellules, nous devons mettre à jour le maillage à partir des chaînes

que nous avons obtenues. C'est un processus assez simple, mais tout de même plus complexe que ce que nous avons décrit jusqu'à présent dans cette section. Le processus est facilité par l'utilisation de chaînes de p-cellules (une 1-chaîne de 0-cellules et deux 1-chaînes de 1-cellules), et il est résumé dans le pseudo-code[5] suivant :

pour *toutes les anciennes* 1*-cellules du complexe mesh* **faire**
- Créer une nouvelle 0-cellule correspondant au *point sur l'arête.*
- Créer deux nouvelles 1-cellules en utilisant la nouvelle 0-cellule et deux anciennes 0-cellules (de l'ancienne 1-cellule correspondant à l'*arête*).
- Enregistrer la nouvelle 0-cellule dans la 1-chaîne de 0-cellules.
- Enregistrer la première nouvelle 1-cellule dans la première 1-chaîne de 1-cellules.
- Enregistrer la deuxième nouvelle 1-cellule dans la deuxième 1-chaîne de 1-cellules.

pour *toutes les anciennes* 2*-cellules du complexe mesh* **faire**
- **pour** *toutes les anciennes* 1*-cellules de l'ancienne* 2*-cellule* **faire**
  - Créer une nouvelle 1-cellule, en utilisant une 0-cellule de la 1-chaîne de 0-cellule et une nouvelle 0-cellule correspondant au *point sur la face.*
  - Créer une nouvelle 2-cellule en utilisant ces nouvelles 1-cellules et les 1-cellules dans les 1-chaînes de 1-cellules.

Un nouveau complexe *mesh* est maintenant composé des nouvelles cellules et des anciennes 0-cellules. L'opération de création indiquée dans le pseudo-code implique seulement des affectations et des interrogations des valeurs des coefficients (p-cellules). C'est une façon pratique d'obtenir les 0-cellules et les 1-cellules perti-

[5]Le code complet est disponible [30].

nentes pour la construction du nouveau complexe.

Chapitre 4

Extensions au cadre de travail

À la première section de ce chapitre, nous explorons le concept de la cellule qui pourrait représenter autre chose que la géométrie d'un système. Ensuite, le mécanisme de sélection, qui permet la manipulation d'une partie d'une chaîne, est étudié à la deuxième section. La partie majeure de ce chapitre est à la troisième section. Nous allons étendre notre cadre de travail sur des chaînes ayant des coefficients très particuliers, c'est-à-dire, des sous-chaînes. Les chaînes de sous-chaînes permettent d'associer des coefficients, non pas seulement à une cellule, mais à deux. Les extensions à cette dernière section sont utilisées par la suite au chapitre 5 pour montrer de nouvelles applications possibles avec le cadre de travail.

4.1 Cellules généralisées

Nous sommes maintenant en position de distinguer différents niveaux de généralité pour la partie théorique du cadre de travail, selon le niveau de structure sur les p-cellules.

En examinant notre cadre de travail lorsqu'il est utilisé pour des applications

qui requièrent que les cellules soient orientées[1], nous constatons qu'il est très général et qu'il peut supporter un grand nombre d'applications en modélisation solide, en ingénierie [1, 2, 9, 12] et en infographie comme nous l'avons vu dans cette thèse. En fait, ce niveau de généralité est suffisant pour supporter des applications avec de la géométrie. D'un autre côté, nous pensons qu'il y a d'autres applications travaillant avec des p-cellules, sans définir l'opérateur *boundary* ou l'intersection de cellules comme nous l'avons fait aux sections 2.1 et 2.3. Une p-cellule généralisée de cette façon, est simplement un regroupement de $(p-1)$-cellules ayant certaines propriétés en commun. Remarquez que ces cellules pourraient ne représenter aucune géométrie. Prenons par exemple la méthode de rendu décrite dans [31], qui groupe des rayons dans une boîte (*bounding shaft*) ; ces rayons peuvent être vus comme étant des 1-cellules et les boîtes comme étant des 2-cellules.

Les p-cellules généralisées de la sorte, pourraient plutôt être appelées des p-*grouping*. Avec cette terminologie, nous pouvons reformuler les définitions du début de la section 2.1. Un p-grouping est un ensemble fini de $(p-1)$-grouping, où les 0-grouping sont considérés comme étant les primitives. Nous notons les ensembles de p-groupings par S_p, $0 \leq p \leq n$, et nous définissons un n-complexe généralisé K comme étant la collection d'ensembles $S_0, \ldots, S_n$. Il n'y a pas nécessairement une relation géométrique entre les $(p-1)$-groupings formant un p-grouping, mais même s'il n'y en a pas, il peut y avoir une relation à un autre niveau.

Si nous introduisons le concept d'orientation et l'opérateur *boundary* ∂_p, $0 \leq p \leq n$, et que nous ajoutons la contrainte que l'intersection entre deux cellules est vide ou résulte en une cellule, et qu'en plus, nous ajoutons que chacune des frontières d'un p-grouping doit former un cycle [9, page 504], alors nous obtenons un p-grouping qui est une cellule et nous sommes dans le cas spécial de la section 2.3.

Notez que le concept de "face-coface" (et la méthode *adj*) a du sens, même

[1] L'opération *boundary* sur une cellule est alors définie.

en l'absence de l'opérateur *boundary*. Nous avons encore les informations permettant d'avoir les relations d'adjacence entre deux cellules du complexe. Nous pouvons aussi utiliser la méthode *dim_inc* sans utiliser, par contre, l'orientation relative. Nous répétons qu'un p-grouping peut ne pas ressembler à une cellule, mais représenter seulement une relation topologique quelconque.

4.2 Mécanisme de sélection

Dans cette section, nous voulons effectuer un traitement sur des coefficients associés non pas à toutes les p-cellules d'un complexe, mais seulement à certaines d'entres elles. Autrement dit, nous voulons manipuler seulement une partie d'une chaîne.

Cela peut être utile dans l'exemple du drapeau à la sous-section 3.1.2, où l'usager veut affecter des constantes d'élasticité de ressorts différentes selon les types de ressorts (horizontaux, verticaux, de cisaillement, ou de flexion).

Pour ce faire, nous avons besoin d'une *sélection* que nous définissons comme étant un vecteur de p-cellules qui sont dans le complexe K. Avec ce vecteur, que l'on note S^p, dont la longueur est l, nous pouvons accéder aux différentes p-cellules S^p_j, pour $j = 1, \ldots, l$.

Supposons que nous ayons une chaîne :

$$\sum_i g_i C^p_i,$$

nous pouvons ainsi avoir une *sélection sur la chaîne* :

$$\sum_{j=1}^{l} g_{i_j} S^p_j,$$

avec les i_j étant une séquence d'indices $i_1, i_2, \ldots, i_l$ tel que nous avons que $C^p_{i_j} = S^p_j$.

Comme c'est le cas avec une chaîne, on peut faire une correspondance entre

la somme formelle et un vecteur de coefficients (voir section 2.4 et [6, page 226]), soit :

$$\sum_{j=1}^{l} g_{i_j} S_j^p \quad \longleftrightarrow \quad (g_{i_1}, g_{i_2}, \ldots, g_{i_j}, \ldots, g_{i_l}).$$

Avec une sélection sur des chaînes, nous pouvons appliquer des opérations binaires et des méthodes[2], cellule par cellule, comme nous l'avons fait sur les chaînes à la sous-section 2.4.1. Nous exigeons cependant que les chaînes impliquées, le soient sur la même sélection.

Dans l'interface de programmation, pour avoir une sélection sur une chaîne, il suffit de faire :

`chainp[vect_pcells]`,

avec `chainp` comme étant une p-chaîne sur le complexe K et `vect_pcells` étant la sélection, soit un vecteur de p-cellules[3] qui sont dans le complexe K. Nous pouvons ainsi faire, par exemple :

`chain1_a[vect_1cells] = chain1_a[vect_1cells] + chain1_b[vect_1cells]`.

Nous soulignons que cet exemple est équivalent à faire :

`chain1_a = chain1_a + chain1_b;`

lorsque $vect\_1cells[i]$ est égal à $K.cell(i,p)$ pour $i = 1, \ldots, K.nb\_cells(p)$ avec $vect\_1cells[i]$ qui dénote la $i^{\text{ième}}$ 1-cellule de la sélection $vect\_1cells$.

Examinons le processus général lorsque nous avons une expression avec une sélection sur des chaînes, suivie de l'affectation de cette sélection sur une chaîne.

[2]Il s'agit des opérations binaires et des méthodes qui sont aussi définies sur les coefficients.

[3]Dans notre interface de programmation, nous avons une classe représentant un vecteur de p-cellules, avec des méthodes permettant d'ajouter des p-cellules, de demander la longueur du vecteur, etc.

On note chacune des p-chaînes X_i, et la sélection S^p, qui contient l p-cellules. L'expression contient alors i termes de la forme $X_i[S^p]$. Pour évaluer cette expression de p-chaînes avec la sélection, l'expression comportant i termes $X_i[S^p[j]]$ est évaluée pour $j = 1, \ldots, l$. Notez que $S^p[j]$ retourne la $j^{\text{ième}}$ p-cellule de la sélection S^p et que $X_i[c]$ retourne le coefficient qui est associé à la cellule c. Finalement, supposons que l'expression avec une sélection sur des p-chaînes est affectée à cette sélection sur une autre p-chaîne X_r. Dans ce cas, le résultat pour chacun des $j = 1, \ldots, l$ est affecté au coefficient $X_r[S^p[j]]$.

Revenons à l'exemple du drapeau pour illustrer un cas pratique. Lors de la construction de celui-ci, nous pouvons ajouter les 1-cellules qui représentent les ressorts dans différentes sélections (vecteurs), soient :

- `vect1cells_square` (ressorts horizontaux et verticaux),
- `vect1cells_shear` (ressorts de cisaillement),
- `vect1cells_flex` (ressorts de flexion).

Ensuite, pour affecter des constantes particulières à chacun des types de ressorts, il suffit de faire :

```
chain1_ks[vect1cells_square] = Chain_Real(10000.0);
chain1_ks[vect1cells_shear] = Chain_Real(1000.0);
chain1_ks[vect1cells_flex] = Chain_Real(100.0).
```

Remarquez que les chaînes à droite des égalités n'ont pas de sélection. Ce n'est pas nécessaire ici, car chacune de ces chaînes retourne toujours le même coefficient, peu importe les cellules associées.

Le mécanisme de sélection est un ajout utile au cadre de travail pour effectuer un traitement particulier sur une partie d'une chaîne, et comme ce fut le cas dans l'exemple du drapeau, certaines valeurs sont affectées aux coefficients associés à une sélection de p-cellules.

4.3 Chaînes de sous-chaînes

Il est difficile d'utiliser dans certains cas, les chaînes pour décrire des quantités associées aux éléments topologiques d'un objet. C'est notamment le cas avec des moments de force dans un réseau de masses-ressorts où nous voulons associer un angle de position de repos pour chacune des 0-cellules adjacentes aux 2-cellules (voir la figure 4.1). Remarquez qu'il y a de un à quatre angles par 0-cellule, ou si l'on considère les angles par 2-cellule, il y a quatre angles par 2-cellule dans le complexe illustré à cette figure. On ne peut donc pas représenter ces angles simplement par une 0-chaîne ou une 2-chaîne de réels.

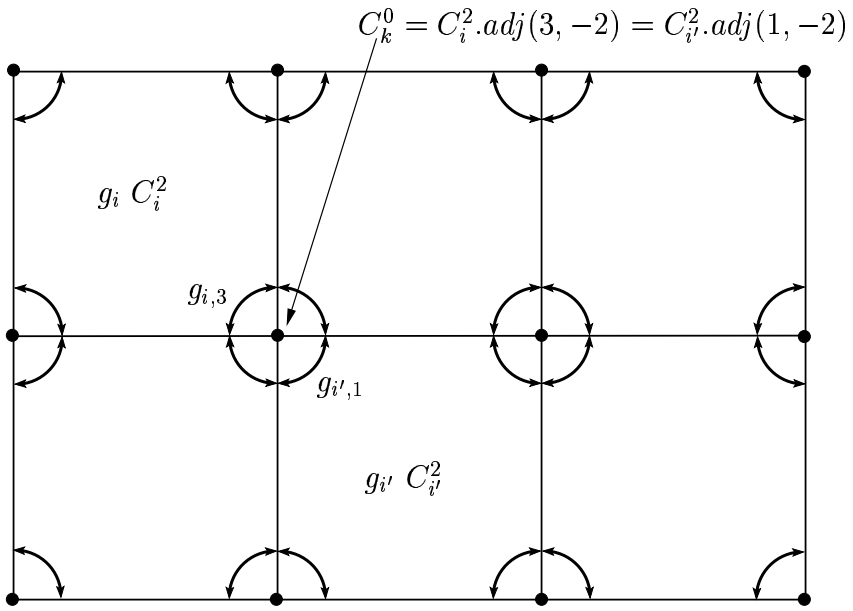

FIG. 4.1 – 2-complexe avec des angles de repos.

Si par contre, nous avions un 2-complexe où chacune des 2-cellules contenait exactement quatre 0-cellules adjacentes (comme à la figure 4.1), nous pourrions avoir une 2-chaîne où chacun de ses coefficients g_i serait un vecteur de quatre nombres réels. Notons $g_{i,j}$ le $j^{\text{ième}}$ réel du coefficient g_i. Ainsi, l'angle de repos associé à la 2-cellule C_i^2 et à la 0-cellule $C_i^2.adj(j,-2)$ est conservé dans $g_{i,j}$. Par

exemple, à la figure 4.1, on voit le coefficient $g_{i,3}$ qui est associé à la paire formée de la 2-cellule C_i^2 et de la 0-cellule $C_i^2.adj(3,-2)$. Nous avons aussi le coefficient $g_{i',1}$ qui est associé à la 2-cellule $C_{i'}^2$ et à la 0-cellule $C_{i'}^2.adj(1,-2)$. Remarquez que dans notre exemple la 0-cellule C_k^0 du complexe correspond à la 0-cellule $C_i^2.adj(3,-2)$ et à la 0-cellule $C_{i'}^2.adj(1,-2)$.

Par contre, nous savons qu'en général, une p-cellule contient un nombre variable de $(p+\delta p)$-cellules adjacentes. C'est pourquoi nous introduisons un tout nouveau type de chaîne : une *chaîne de sous-chaînes*[4]. Nous définissons une p-chaîne de $(p+\delta p)$-sous-chaînes sur un n-complexe K par un vecteur de vecteurs de coefficients $g_{i,j}$:

$$\left[(g_{1,1}, g_{1,2}, ..., g_{1,\nu_1^{\delta p}}), (g_{2,1}, g_{2,2}, ..., g_{2,\nu_2^{\delta p}}), ..., (g_{N_p,1}, g_{N_p,2}, ..., g_{N_p,\nu_{N_p}^{\delta p}})\right] \tag{4.1}$$

avec $N_p = K.nb\_cells(p)$, $\nu_i^{\delta p} = C_i^p.nb\_adj(\delta p)$ et $i = 1, ..., N_p$. Le coefficient $g_{i,j}$ est associé à la paire de cellules C_i^p et $C_i^p.adj(j, \delta p)$.

Cette p-chaîne de $(p+\delta p)$-sous-chaînes[5] est notée (p,q)-chaîne avec $q = p+\delta p$. Il est possible que δp soit égal à 0^+ ou 0^-. Nous permettons ainsi $q = p + 0^+$, ce qui nous donne une $(p, p+0^+)$-chaîne, avec

$$\delta p = q - p = p + 0^+ - p = 0^+,$$

et nous permettons aussi $p = q + 0^+$, ce qui nous donne une $(q+0^+, q)$-chaîne, avec

$$\delta p = q - p = q - (q + 0^+) = -0^+$$

(voir la section 2.2 pour la convention que $-0^+ = 0^+$).

La $(p, p+0^+)$-chaîne et la $(q+0^+, q)$-chaîne sont constituées toutes les deux de chaînes de sous-chaînes impliquant des cellules de même dimension et dans les

[4]Dans l'exemple présenté à la figure 4.1, nous avons une 2-chaîne de 0-sous-chaînes.

[5]Nous utilisons le terme "sous-chaînes" car ce n'est pas un vecteur de coefficients sur toutes les $(p+\delta p)$-cellules du complexe, mais seulement sur les $(p+\delta p)$-cellules qui sont adjacentes à C_i^p.

deux cas $\delta p = 0^+$, nous avons alors la même représentation de chaînes de sous-chaînes à la ligne 4.1 ci-dessus. Nous pouvons faire la même remarque pour une $(p, p+0^-)$-chaîne et une $(q+0^-, q)$-chaîne.

Il est requis dans une (p,q)-chaîne que $0 \leq p \leq n$, $0 \leq p+\delta p \leq n$, où n est la dimension du complexe. Ici $0+0^- < 0$ et $n+0^+ > n$.

La (p,q)-chaîne peut être vue, d'une façon équivalente, comme étant une p-chaîne de $(p+\delta p)$-sous-chaînes, avec le $i^{\text{ième}}$ coefficient g_i associé à la cellule C_i^p égal à la sous-chaîne :

$$(g_{i,1}, g_{i,2}, ...g_{i,\nu_i^{\delta p}}), \tag{4.2}$$

avec $i = 1, ..., N_p$. Cette façon de voir nous permet de considérer une chaîne de sous-chaînes comme étant une chaîne telle que vu précédemment (section 2.4), mais avec des coefficients un peu spéciaux.

Le coefficient g_i peut aussi être écrit avec une somme formelle :

$$\sum_{j=1}^{\nu_i^{\delta p}} g_{i,j}\ C_i^p.adj(j, \delta p).$$

Ainsi, avec une somme formelle, une (p,q)-chaîne peut être vue comme :

$$\sum_i \sum_{j=1}^{\nu_i^{\delta p}} g_{i,j}\ C_i^p.adj(j, \delta p).$$

4.3.1 Chaînes de sous-chaînes duales

Dans la p-chaîne de $(p+\delta p)$-sous-chaînes, le coefficient $g_{i,j}$ est associé avec la paire de cellules[6] :

$$[C_i^p, C_i^p.adj(j, \delta p)]\ .$$

[6]Notez que l'ordre correspondant à l'index i est l'ordre de l'ajout de la p-cellule au complexe, et que l'ordre correspondant à l'index j dépend de l'assemblage des cellules.

Mais l'adjacence est une relation symétrique comme nous l'avons vu à la section 2.2, c'est-à-dire que pour toutes cellules b et c d'un complexe K, nous avons que :

$$b = c.adj(i, \delta p) \implies \text{il existe un } j \text{ unique, tel que } c = b.adj(j, -\delta p).$$

Ainsi, pour chaque $(p, p+\delta p)$-chaîne (ou de façon équivalente, une p-chaîne de $(p+\delta p)$-sous-chaînes), il y a une $(p+\delta p, p)$-chaîne *duale* (ou une $(p+\delta p)$-chaîne de p-sous-chaînes). Encore une fois, nous répétons qu'il n'y a pas de restriction sur le signe de δp, il faut seulement que $0 \leq p \leq n$, $0 \leq p+\delta p \leq n$, où n est la dimension du complexe.

Un exemple d'une $(2,0)$-chaîne d'entiers est illustrée à la figure 4.2. Selon l'ordre d'ajout des cellules au complexe et l'ordre d'assemblage des cellules, la $(2,0)$-chaîne pourrait être définie par le vecteur de vecteurs suivant :

$$[(7,5,2,1),(4,2,3,11),(1,9,-2,2),(3,4,1,7),(1,6,2,-3),(1,10,4,3)]\,.$$

La chaîne duale de cette $(2,0)$-chaîne est une $(0,2)$-chaîne et pourrait être définie par le vecteur de vecteurs suivant :

$$[(7),(5,4),(2,1),(9),(1,3),(2,11,4,1),\ldots]\,.$$

Cette chaîne duale est aussi illustrée à la figure 4.2. Remarquez que les deux chaînes sont illustrées de la même façon.

4.3.2 Interface de programmation avec des chaînes de sous-chaînes

La chaîne duale d'une $(p, p+\delta p)$-chaîne est unique et bien définie car, comme nous l'avons mentionné ci-dessus, l'adjacence est une relation symétrique. Elle dépend de l'ordre dans lequel les cellules ont été assemblées. L'algorithme pour trouver la chaîne duale est décrit à la sous-section 4.3.3.

Comme nous l'avons vu ci-dessus (équation 4.2), une sous-chaîne peut être représentée par un vecteur de coefficients. Nous avons des sous-chaînes sur R,

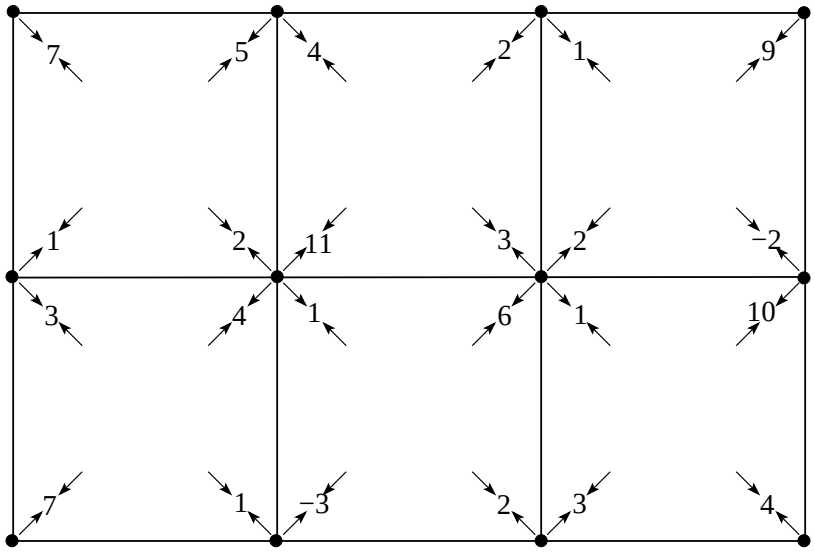

FIG. 4.2 – Une $(2,0)$-chaîne et de sa chaîne duale, une $(0,2)$-chaîne.

R^2, etc.[7] ; c'est-à-dire des sous-chaînes dont les coefficients sont de la classe *Type*. Nous définissons que les opérations binaires et les méthodes qui sont définies entre des coefficients, le seront également entre les sous-chaînes sur les mêmes types de coefficient[8]. Nous avons également la méthode *coefficient*(j) (et l'opérateur $[j]$) sur une sous-chaîne qui retourne la référence à la $j^{\text{ième}}$ composante du vecteur de coefficients (de type *Type*) représentant la sous-chaîne.

Dans notre interface de programmation, nous avons ajouté des classes de chaînes de sous-chaînes qui sont dérivées de la classe de base *Chain*. Nous avons

[7]Dans notre système, nous avons des chaînes sur différents types de coefficient, et nous avons les sous-chaînes sur les mêmes types, sauf pour les types de coefficient qui sont des sous-chaînes (voir annexe B).

[8]Nous appliquons les opérations binaires et les méthodes, coefficient par coefficient, comme ce fut le cas avec les chaînes (sous-section 2.4.1) de même que pour une sélection sur des chaînes (sous-section 2.4.1).

des chaînes de sous-chaînes de R, R^2, R^3, nommées[9] *Chain2_Real*, *Chain2_Real2*, *Chain2_Real3* (voir l'annexe B pour plus de détails). Notez que le programmeur peut ajouter facilement des chaînes de sous-chaînes d'un autre type *Type*, qu'il nommera *Chain2_Type*. Une classe typique *Chain2_Type* a le constructeur suivant :

- *Chain2_Type(int p, int q, Complex complex_K).* Constructeur pour une (p, q)-chaîne dont les coefficients sont de la classe *Type*. Requiert les paramètres d'entrées p, q, et *complex_K* (de la classe *Complex*), le complexe sur lequel la (p, q)-chaîne est définie.

Nous pouvons accéder à un coefficient $g_{i,j}$ associé à la paire de cellules C_i^p et $C_i^p.adj(j, \delta p)$, d'une p-chaîne de $(p + \delta p)$-sous-chaînes, soit *chainpq* avec $chainpq.coefficient(i).coefficient(j)$, ou plus simplement, avec $chainpq[i][j]$. Ceci est dû au fait que premièrement, *chainpq* est une p-chaîne, alors $coefficient(i)$ est défini et retourne la référence à la sous-chaîne g_i. Deuxièmement, la méthode $coefficient(j)$ que nous avons défini ci-dessus, retourne la référence à la $j^{\text{ième}}$ composante du vecteur (équation 4.2) la représentant, c'est-à-dire $g_{i,j}$. Nous pouvons aussi remplacer i par C_i^p avec le même résultat, car l'opérateur "[ ]" et la méthode *coefficient* sont définis sur une p-chaîne, que ce soit une cellule ou un entier comme paramètre d'entrée (section 2.4). Notez que dans les deux cas, avec notre implantation, nous avons accès à $g_{i,j}$ en $O(1)$.

Afin d'accéder à un coefficient $g_{i,j}$ avec une paire de cellules, nous utilisons la méthode *coefficient* de *Chain2_Type*, définie comme suit :

- *Type& coefficient(Cell cp, Cell cq).* Retourne la référence sur le coefficient (de la classe *Type*) associé à la paire de cellules *cp* et *cq* de la (p, q)-chaîne qui invoque la méthode. Les cellules *cp* et *cq*, sont respectivement de dimension p et q et doivent être adjacentes.

L'implantation de la méthode effectue une recherche pour trouver le j tel que : $cp.adj(j, q - p) = cq$. Ensuite elle retourne le coefficient chainpq[cp][j]. Celle-ci est

[9]Le nom *Chain2_...* est une abréviation pour *Chain_of_subchains_....*

de l'ordre du nombre de cellules adjacentes.

Dans le but de manipuler les chaînes de sous-chaînes, comme nous le ferons par exemple dans le chapitre 5, nous allons définir des opérations et des méthodes sur celles-ci.

Nous définissons que les opérations binaires et méthodes définies entre les coefficients, le sont également entre les sous-chaînes sur les mêmes types de coefficient. Chacune des sous-chaînes sur un certain type de coefficient étant un nouveau type de coefficient pour une chaîne, les opérations binaires et les méthodes définies sur des sous-chaînes sur certains types de coefficient, le seront également sur les chaînes de sous-chaînes sur les mêmes types de coefficient.

En fait, nous ne faisons qu'appliquer la définition d'une opération binaire ou d'une méthode sur une chaîne (sous-section 2.4.1) dont les coefficients sont dans notre cas, des sous-chaînes.

C'est le cas par exemple de l'opérateur binaire "+", qui est défini entre des coefficients de type réel (de la classe *Real*) et qui par conséquent, le sera aussi entre deux q-sous-chaînes sur les réels (addition des coefficients, cellule par cellule). Nous avons ensuite que l'opérateur binaire "+" sera aussi défini entre deux p-chaînes de q-sous-chaînes sur les réels (encore là, addition des coefficients, cellule par cellule, les coefficients étant des sous-chaînes). Ainsi lors de l'addition de deux (p, q)-chaînes de réels, il en résultera une (p, q)-chaîne de réels. Pour chacune des paires C_i^p et $C_i^p.adj(j, \delta p)$ du complexe $(\delta p = q - p)$, nous avons deux coefficients (réels) associés dans les deux (p, q)-chaînes, la somme de ceux-ci est emmagasinée dans la chaîne résultante à la même paire de cellules. Voici un exemple dans l'interface de programmation de deux $(2, 0)$-chaînes `chain20_a` et `chain20_b` qui sont additionnées et dont le résultat est conservé dans la $(2, 0)$-chaîne `chain20_c` :

`chain20_c` = `chain20_a` + `chain20_b`.

Notez que nous utilisons comme convention pour étiqueter une (p, q)-chaîne

comme étant *chainpq_nom*, la partie "_*nom*" étant optionnelle.

4.3.3 La méthode transpose

Supposons une $(2,0)$-chaîne *chain20* (comme l'exemple à la figure 4.1) ; nous avons aussi une chaîne duale $(0,2)$-chaîne *chain02*. Les deux chaînes de sous-chaînes ont exactement le même nombre de coefficients mais la première chaîne est une 2-chaîne de 0-sous-chaînes, tandis que la deuxième est une 0-chaîne de 2-sous-chaînes. Autrement dit, la première chaîne est organisée en fonction des 2-cellules du complexe et la deuxième, en fonction des 0-cellules. On peut faire une conversion entre les deux chaînes de sous-chaînes avec la méthode *transpose*. C'est-à-dire que $chain02 = chain20.transpose()$. En généralisant, nous avons qu'une (p,q)-chaîne peut aussi être conservée dans une (q,p)-chaîne (chaîne duale) en utilisant la méthode *transpose*. La méthode *transpose* est définie dans les classes *Chain2_Type* (voir annexe B) :

- *Chain2_Type transpose()*. Retourne la (q,p)-chaîne duale (de la classe *Chain2_Type*) à partir de la (p,q)-chaîne qui invoque la méthode.

Avant de définir l'algorithme de *transpose*, rappelons la propriété vue à la section 2.2, que pour toutes cellules b et c d'un complexe K, nous avons :

$$b = c.adj(i, \delta p) \implies \text{il existe un } j \text{ unique tel que } c = b.adj(j, -\delta p). \qquad (4.3)$$

En utilisant cette propriété, nous pouvons définir l'algorithme qui est utilisé lorsque la méthode *transpose* est appliquée à une (p,q)-chaîne *chainpq* qui retourne la (q,p)-chaîne duale *chainqp* :

```
δp = q − p;
pour toutes les cellules C_i^q du complexe K faire
    pour j=1 à C_i^q.nb_adj(−δp) faire
        c^p ← C_i^q.adj(j, −δp) ;
        k ← c^p.identifier() // identifier trouve k tel que C_k^p = c^p ;
        trouver l tel que
            C_k^p.adj(l, δp) = C_i^q // d'après la propriété 4.3 ci-dessus ;
        chainqp[i][j] ← chainpq[k][l] ;
```

Notez que ce qu'il y a à l'intérieur des deux boucles imbriquées, peut simplement être la première ligne et $chainqp[i][j] \leftarrow chainpq.coefficient(c^p, C_i^q)$.

Nous avons la propriété qu'en appliquant la méthode *transpose* deux fois sur une (p, q)-chaîne *chainpq*, nous obtenons une chaîne équivalente. C'est-à-dire que :

$$chainpq \equiv chainpq.transpose().transpose().$$

Nous avons des exemples d'utilisation de la méthode *transpose* à la prochaine sous-section.

4.3.4 D'autres méthodes sur un ensemble de coefficients

Les méthodes et les opérations binaires sur des p-chaînes que nous avons présentées en exemples jusqu'ici, retournaient une p-chaîne[10]. Nous pouvons aussi envisager des méthodes qui retournent une seule valeur plutôt qu'une chaîne. Définissons par exemple la méthode *sum* qui retourne la somme de tous les coefficients d'une chaîne de R^n, pour $n \geq 1$. Cette méthode peut être utilisée par exemple pour savoir la position moyenne des 0-cellules d'un complexe K de la classe

[10]Ici il est question des méthodes et des opérations binaires dans une expression de chaînes. Il ne s'agit pas de la méthode *coefficient*.

Complex_p (voir section 2.6). Il suffit d'effectuer : $K.chain\_p.sum()/K.nb\_cells(0)$. Nous pourrions aussi définir la méthode *mult* qui retourne le produit de tous les coefficients d'une chaîne de R^n, pour $n \geq 1$. Toutes les méthodes appliquées sur une chaîne et qui retournent une valeur (par exemple *add* et *mult*) sont aussi définies sur une sous-chaîne.

Dans le contexte d'une p-chaîne de q-sous-chaînes, nous avons que la méthode *sum* (ou la méthode *mult*) est appliquée à chacune des sous-chaînes de dimension q (cellule par cellule, voir la sous-section 2.4.1). Il en résulte alors une p-chaîne. Chacun des coefficients g_i de cette p-chaîne est égal à la somme des coefficients $g_{i,j}$, pour j allant de 1 à $C_i^p.nb\_adj(\delta p)$. En d'autres termes, tous les coefficients associés à une cellule C_i^p d'une (p,q)-chaîne sont additionnés pour produire un coefficient g_i d'une p-chaîne. Un exemple est illustré à la figure 4.3, qui montre l'application de la méthode *sum* sur une $(2,0)$-chaîne, pour résulter en une 2-chaîne. Chacun des coefficients de la $(2,0)$-chaîne est indiqué entre deux flèches droites pointant l'une vers l'autre qui originent de la paire de cellules (une 2-cellule et une 0-cellule) impliquée. Les flèches courbes, quant à elles, indiquent comment les coefficients de la 2-chaîne sont obtenus des coefficients de la (2,0)-chaîne.

On remarque que cette opération sur une chaîne de sous-chaînes est très proche de la méthode *dim_inc*. En fait, nous verrons à la section 4.3.6 que l'on peut utiliser la méthode *sum* sur une (p,q)-chaîne pour faire l'équivalent de la méthode *dim_inc* avec comme opération *plus*.

Une suite d'opérations qui produit une q-chaîne *chainq* à partir d'une (p,q)-chaîne *chainpq* est d'appliquer la méthode *transpose* sur celle-ci, et d'ensuite appliquer par exemple la méthode *sum* sur la (q,p)-chaîne résultante. Soit : $chainq = chainpq.transpose().sum()$. À la figure 4.2, nous avons au départ une $(2,0)$-chaîne. Nous appliquons la méthode *transpose* sur celle-ci pour obtenir la $(0,2)$-chaîne de départ à la figure 4.4 sur laquelle on applique la méthode *sum* pour obtenir une 0-chaîne. Remarquez que les illustrations de la $(2,0)$-chaîne et

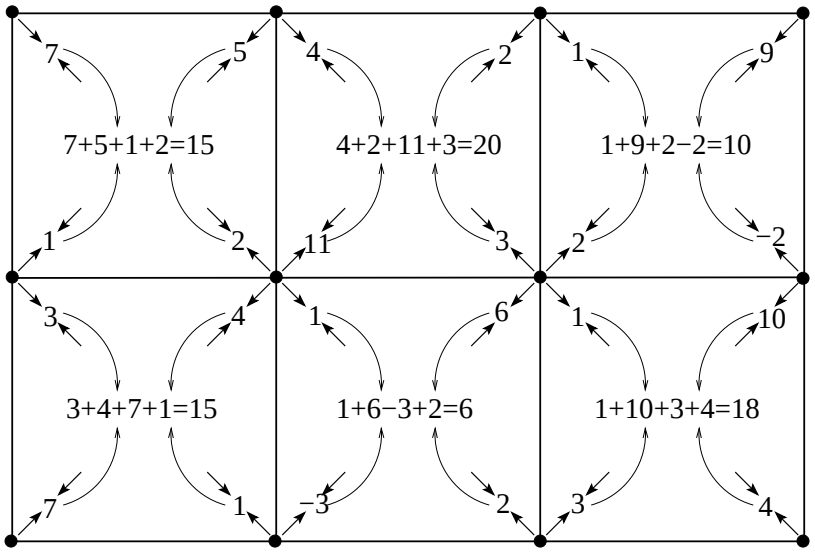

FIG. 4.3 – Méthode *sum* appliquée sur une $(2,0)$-chaîne qui résulte en une 2-chaîne.

de la $(0,2)$-chaîne aux figures 4.2 et 4.4 sont identiques car la méthode *transpose* ne modifie pas notre illustration d'une chaîne de sous-chaînes.

Nous avons pris la méthode *sum* en exemple, mais nous pourrions aussi définir la méthode *mult* qui effectue un produit entre les coefficients, ou alors toute méthode qui effectue une opération commutative entre les coefficients afin de produire une seule valeur. Nous notons ce genre d'opération Ψ.

4.3.5 Orientation dans une chaîne de sous-chaînes

Il peut être utile de conserver, plutôt que de calculer à chaque fois, l'orientation relative (section 2.3) entre les p-cellules d'un complexe et ses faces. Nous pouvons emmagasiner ces orientations relatives dans une $(p, p-1)$-chaîne d'entiers ou de réels. Pour faire cela, les valeurs $C_i^p.orientation(C_i^p.adj(j,-1))$ sont affectées à chacun des coefficients $g_{i,j}$. Nous avons ajouté une méthode à la classe *Complex* afin de produire une $(p, p-1)$-chaîne d'orientations relatives, soit :

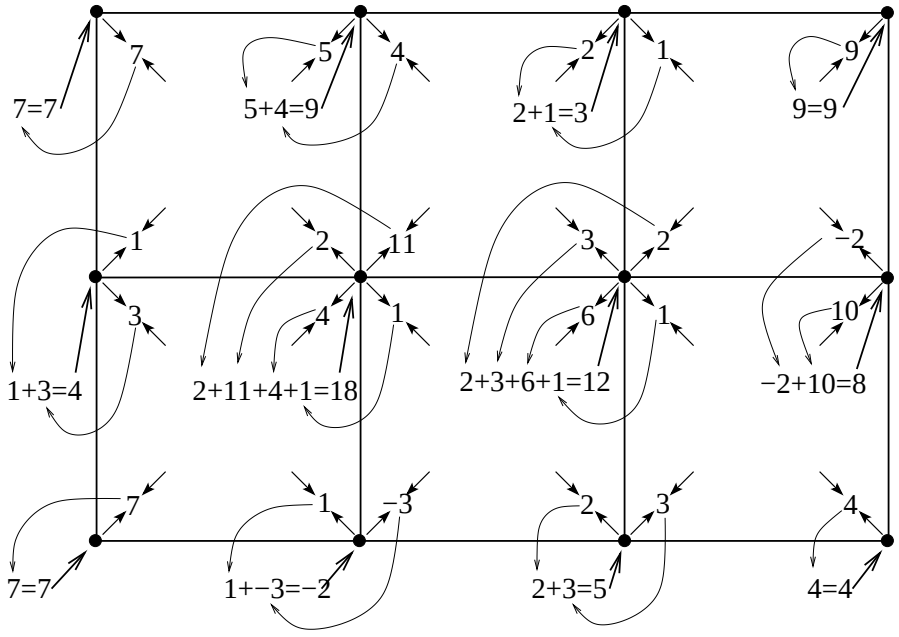

FIG. 4.4 – Méthode *sum* appliquée sur une $(0,2)$-chaîne qui résulte en une 0-chaîne.

- *Chain2_Real orientation(int p).* Retourne la $(p, p-1)$-chaîne d'orientations relatives.

Nous verrons aussi que cette $(p, p-1)$-chaîne d'orientations relatives est utile pour la manipulation de chaînes de sous-chaînes.

4.3.6 La méthode *out* et la méthode *in*

Nous introduisons la méthode *out* qui produit une (p,q)-chaîne *chainpq* à partir d'une p-chaîne. Étant donné en entrée une p-chaine *chainp* et δp, les valeurs g_i de la p-chaîne sont affectées aux coefficients $g_{i,j}$ de la (p,q)-chaîne, avec $j = 1, ..., C_i^p.nb\_adj(\delta p)$ et $q = p + \delta p$. Voici donc l'algorithme qui est utilisé pour cela :

pour *toutes les cellules* C_i^p *du complexe* K **faire**
  pour j=*1 à* $C_i^p.nb\_adj(\delta p)$ **faire**
    $chainpq[i][j] \leftarrow chainp[i]$;

Dans notre interface de programmation nous avons la méthode *out* qui est valide dans les classes *Chain_Type* :

- *Chain2_Type out(int delta_p, Boolean is_signed).* Retourne une $(p, p+\delta p)$-chaîne (de la classe *Chain2_Type*) à partir de la p-chaîne qui invoque la méthode. L'orientation relative est considérée lorsque *is_signed* est à *true* (*true* est la valeur par défaut) et δp est spécifié par l'entier *delta_p*.

Lorsque l'on tient compte de l'orientation relative (valide seulement lorsque δp est égal à $+1$ ou -1), la $(p, p+\delta p)$-chaîne est multipliée par la $(p, p-1)$-chaîne $K.orientation(p)$ d'orientations relatives (lorsque δp est égal à -1) ou est multipliée par la $(p+1, p)$-chaîne $K.orientation(p+1)$ d'orientations relatives sur laquelle on a d'abord appliqué la méthode *transpose* (lorsque δp est égal à $+1$).

Un exemple de l'application de la méthode $out(-2, false)$ sur une 2-chaîne est illustré à la figure 4.5.

Nous introduisons aussi la méthode *in* qui produit une $(p+\delta p, p)$-chaîne *chainqp* à partir d'une p-chaîne *chainp*. La méthode *transpose* est appliquée sur la p-chaîne sur laquelle la méthode *out(δp, is_signed)* est appliquée, soit : $chainqp = chainp.out(\delta p, is\_signed).transpose()$, avec *is_signed* qui est égal à *true* ou $false$. Dans notre interface de programmation nous avons la méthode *in* qui est valide dans les classes *Chain_Type* :

- *Chain2_Type in(int delta_p, Boolean is_signed).* Retourne une $(p+\delta p, p)$-chaîne (de la classe *Chain2_Type*) à partir de la p-chaîne qui invoque la méthode. L'orientation relative est considérée lorsque *is_signed* est égal à

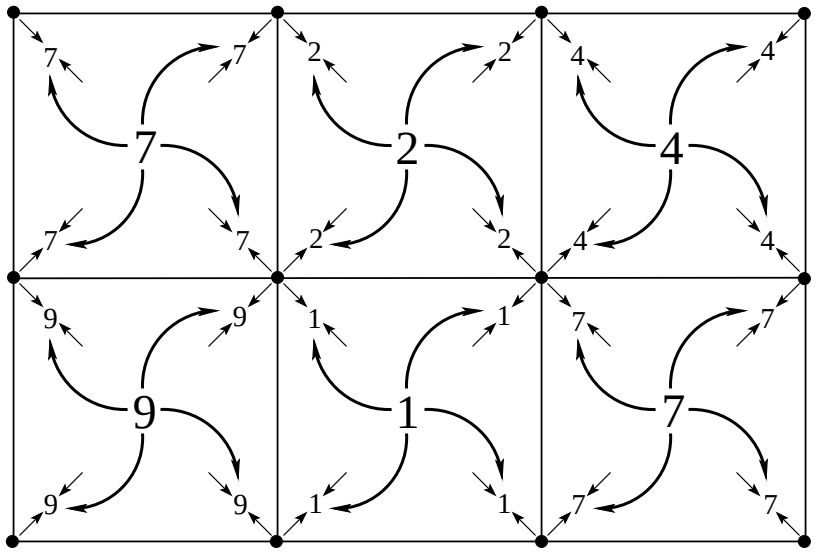

FIG. 4.5 – À partir d'une 2-chaîne, on définit une $(2,0)$-chaîne en effectuant $out(-2, false)$.

true (*true* est la valeur par défaut) et δp est spécifié par l'entier *delta_p*.

Rappelons que la méthode *transpose* ne modifie pas l'illustration d'une chaîne de sous-chaînes, ainsi nous avons à la figure 4.5 qu'en appliquant la méthode $in(-2, false)$ sur une 2-chaîne, nous obtenons la même illustration qu'en appliquant la méthode $out(-2, false)$ sur cette même 2-chaîne. Par contre avec la méthode *in*, le résultat est une $(0,2)$-chaîne, tandis qu'avec la méthode *out*, le résultat est une $(2,0)$-chaîne.

Un exemple de l'application de la méthode $in(+1, false)$ sur une 1-chaîne est illustré à la figure 4.6. Puisque les illustrations de chaînes de sous-chaînes ne sont pas modifiées par la méthode *transpose*, nous avons exactement la même illustration à la figure 4.6 si nous appliquons la méthode $out(+1, false)$ sur la 1-chaîne. Avec la méthode *in* le résultat est une $(2,1)$-chaîne, tandis qu'avec la méthode *out*, le résultat est une $(1,2)$-chaîne.

Reprenons l'exemple précédent mais en tenant compte de l'orientation des

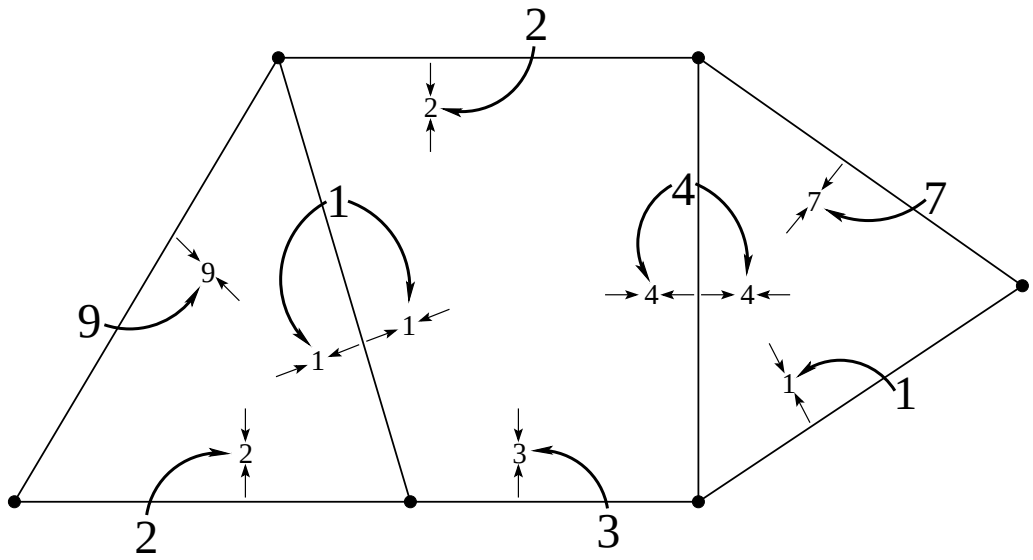

FIG. 4.6 – À partir d'une 1-chaîne, on définit une $(2,1)$-chaîne en effectuant $in(+1, false)$.

cellules. Cet exemple est illustré avec les orientations absolues dans le but de schématiser les orientations à la figure 4.7. Rappelons, par contre, que notre système fonctionne avec des orientations relatives. On remarque que certains coefficients de la $(2,1)$-chaîne changent de signe lorsqu'ils sont affectés d'un coefficient de la 1-chaîne, car pour chacun de ces cas, la paire de 2-cellule et de 1-cellule impliquée, a une orientation relative de -1.

Les méthodes *in* et *out* sont utiles dans le cas où l'on veut faire des opérations binaires ou appliquer des méthodes entre des chaînes et des chaînes de sous-chaînes. Par exemple, nous voudrions additionner les coefficients associés à une 2-cellule d'une $(2,1)$-chaîne *chain21* avec le coefficient associé à cette 2-cellule dans une 2-chaîne *chain2* et ce, pour toutes les 2-cellules du complexe (il en résultera une $(2,1)$-chaîne). Ceci est fait comme suit : $chain21 + chain2.out(-1, false)$. Nous verrons dans les exemples d'applications avec des chaînes de sous-chaînes, que nous effectuons régulièrement ce genre d'opération.

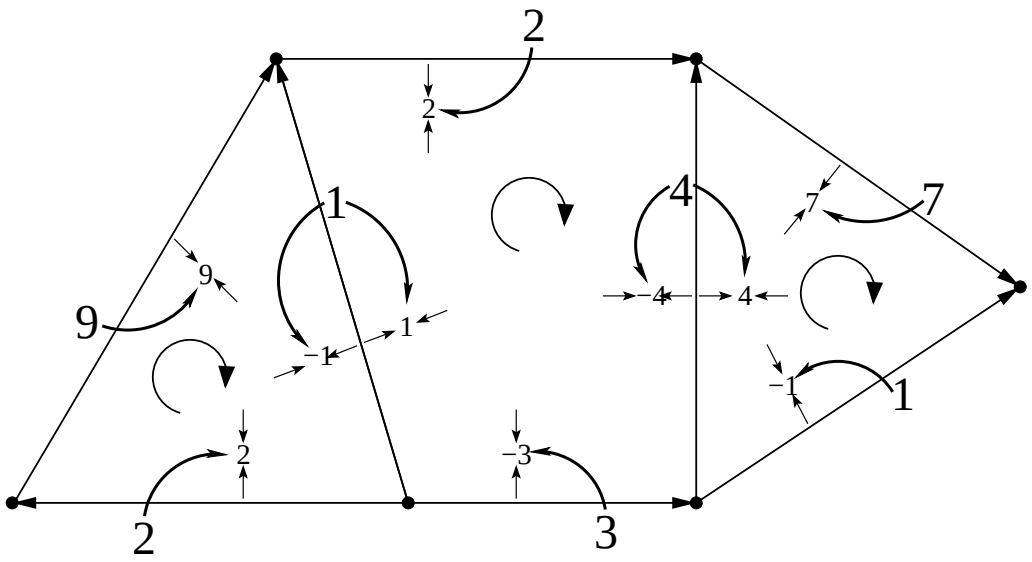

FIG. 4.7 – À partir d'une 1-chaîne, on définit une $(2,1)$-chaîne en tenant compte de l'orientation relative.

Voici quelques exemples pour illustrer des opérations valides comportant des chaînes de sous-chaînes et des chaînes : soit une 0-chaîne $chain0$, une 2-chaîne $chain2$, une $(2,0)$-chaîne $chain20$ et une $(0,2)$-chaîne $chain02$,

$$chain20 = chain2.out(-2, false);$$

$$chain20 = (chain2.in(-2, false)).transpose(); \quad // \text{ équivalent à la}$$

$$// \text{ ligne précédente.}$$

$$chain20 = chain20 + chain2.out(-2, false);$$

$$chain20 = chain0.in(+2, false);$$

$$chain02 = chain2.in(-2, false).$$

Nous avons les équivalences suivantes reliant la méthode *dim_inc* et la méthode *in* :

$$chainp.dim\_inc(\delta p, plus, false) \equiv chainp.in(\delta p, false).sum()$$

et

$$chainp.dim\_inc(\delta p, plus, true) \equiv chainp.in(\delta p, true).sum().$$

Cette dernière propriété est valide seulement pour un δp égal à 1 ou -1.

Nous avons des propriétés semblables pour les opérations Ω de *dim_inc*, telles que *mult*, *average* et *geom_mean*.

Ces propriétés indiquent qu'avec ces nouvelles méthodes (*in*,*sum*,...), qui sont essentielles dans un contexte de chaînes de sous-chaînes, on effectue l'équivalent de *dim_inc* sur des p-chaînes.

Avec la méthode *dim_inc*, nous sommes limités aux opérations commutatives Ω qui ont été programmées dans la méthode *dim_inc*. Avec la méthode *in*, des opérations commutatives[11] Ψ sur les coefficients d'une chaîne (voir sous-section 4.3.4) peuvent être ajoutées, sans avoir à modifier la méthode *in*. Un autre avantage encore plus important, c'est qu'avec les chaînes de sous-chaînes, nous ne sommes pas limités à effectuer une seule opération commutative : nous pouvons faire des manipulations intermédiaires entre l'application de la méthode *in* et l'application de l'opération Ψ. Par exemple, nous pourrions mettre à zéro les coefficients négatifs d'une chaîne de sous-chaînes avant d'effectuer une opération commutative Ψ afin d'obtenir une p-chaîne. Cet exemple est développé dans la modélisation de l'air et de la fumée à la section 5.2.

Il demeure néanmoins que nous voulons conserver la méthode *dim_inc* pour sa facilité d'utilisation et sa concision. Elle est aussi plus rapide, car elle n'a pas à faire une allocation temporaire d'espace mémoire d'une chaîne de sous-chaînes.

[11] Par exemple, les méthodes *sum*, *mult*, etc. sont des opérations commutatives Ψ.

4.3.7 La méthode *dim_inc* appliquée sur une chaîne de sous-chaînes

Nous avons vu qu'il est possible d'utiliser les (p, q)-chaînes afin de définir la méthode *dim_inc* sur une p-chaîne. Par contre, nous n'avons pas défini la signification de la méthode *dim_inc* sur une (p, q)-chaîne.

Nous avons démontré que la méthode *dim_inc* est très utile dans un système avec des p-chaînes, et il en est certainement ainsi avec des (p, q)-chaînes. Nous utilisons cela dans l'exemple des moments de force dans un réseau de masses-ressorts (sous-section 5.1).

Notez que nous n'avons pas trouvé de signification à la méthode *dim_inc* sur une chaîne de sous-chaînes en appliquant l'opération Ω entre les sous-chaînes, car les vecteurs les représentant peuvent avoir des longueurs différentes. Nous avons quand même défini la méthode *dim_inc* sur une chaîne de sous-chaînes, en appliquant celle-ci, aux coefficients de la chaîne, c'est-à-dire aux sous-chaînes, comme c'est le cas pour les autres méthodes et opérations binaires (sous-section 2.4.1). Nous devons alors définir la signification de *dim_inc* sur une sous-chaîne. Attention ici, nous n'appliquons pas la méthode *dim_inc* sur la p-chaîne, mais bien sur chacune des q-sous-chaînes.

Lorsque l'on applique $dim\_inc(\delta p)$ à une p-chaîne sur des coefficients primitifs, il en résulte une $(p + \delta p)$-chaîne. Il en sera de même pour les sous-chaînes. Ainsi, lorsque l'on applique $dim\_inc(\delta q)$ sur une q-sous-chaîne, il en résulte une $(q + \delta q)$-sous-chaîne.

Nous définissons que lorsque l'on applique la méthode $dim\_inc(\delta q)$ sur une (p, q)-chaîne *chainpq*, il en résulte une $(p, q + \delta q)$-chaîne $chainpq_2$ qui est calculée comme suit[12] :

[12]Le calcul de chacune des $(q + \delta q)$-sous-chaînes associées à la cellule C_i^p est à l'intérieur de la première boucle.

$\delta p \leftarrow q - p$;

$\delta p_2 \leftarrow q + \delta q - p$; // $(= \delta p + \delta q)$;

pour *toutes les cellules* C_i^p *du complexe* K **faire**

  pour j=*1 à* $C_i^p.nb\_adj(\delta p_2)$ **faire**

    $c^{q+\delta q} \leftarrow C_i^p.adj(j, \delta p_2)$;

    $S \leftarrow C_i^p.adj(\delta p) \sqcap c^{q+\delta q}.adj(-\delta q)$;

    $chainpq_2.coefficient(C_i^p).coefficient(j) \leftarrow$

$$\Omega_{k=1}^{S.length} chainpq.coefficient(C_i^p, S[k]);$$

Notez que les cellules dans le vecteur S sont de dimension q. Notez aussi que cet algorithme utilise l'intersection $\sqcap$ entre deux vecteurs v_1 et v_2 de cellules, celle-ci étant définie comme suit :

$v_1 \sqcap v_2$. Seules les cellules qui sont dans v_1 et dans v_2 sont dans le vecteur résultant. L'ordre des cellules dans le vecteur résultant n'a pas d'importance ici.

Lorsque nous tenons compte de l'orientation relative (il faut que δq soit de $+1$ ou -1), la dernière ligne de l'algorithme est plutôt :

$$chainpq2.coefficient(C_i^p).coefficient(j) \leftarrow$$
$$\Omega_{k=1}^{S.length} \sigma\ chainpq.coefficient(C_i^p, S[k])$$

avec σ étant l'orientation relative entre $S[k]$ et $c^{q+\delta q}$.

Nous imposons aussi certaines conditions à la (p,q)-chaîne ($q = p + \delta p$) et à δq :

1. $0 \leq q + \delta q \leq n$, où n est la dimension du complexe,
2. δp et δq ne sont pas égaux à 0^+ ou à 0^-,
3. $q + \delta q - p$ n'est pas égal à 0.

Illustrons cet algorithme par un exemple à la figure 4.8. Nous avons une $(2,1)$-chaîne de départ $chain21$ et nous appliquons la méthode $dim\_inc(-1, plus, false)$. Les paramètres d'entrée de dim_inc indiquent que l'opération Ω entre les coefficients est l'addition et que l'on ne tient pas compte de l'orientation relative. Il en résulte une $(2,0)$-chaîne $chain20$, c'est-à-dire que : $chain20 = chain21.dim\_inc(-1, plus, false)$.

Pour ce faire, il faut appliquer la méthode dim_inc sur chacune des 1-sous-chaînes qui résultera en des 0-sous-chaînes. Chacune de ces 1-sous-chaînes est définie à l'intérieur d'une 2-cellule (car nous avons une $(2,1)$-chaîne). Il en résultera des 0-sous-chaînes qui sont aussi définies à l'intérieur d'une 2-cellule (car le résultat est une $(2,0)$-chaîne). L'addition des coefficients pour chacune des 0-sous-chaînes ne se fera qu'avec les coefficients de la 1-sous-chaîne à l'intérieur d'une même 2-cellule.

Autrement dit, pour chacune des 2-cellules du complexe, nous considérons les 0-cellules et les 1-cellules adjacentes à la 2-cellule. Ensuite, nous additionnons les coefficients associés aux 1-cellules qui sont adjacentes à chacune des 0-cellules et nous emmagasinons le résultat dans le coefficient associé à la 2-cellule et à la 0-cellule.

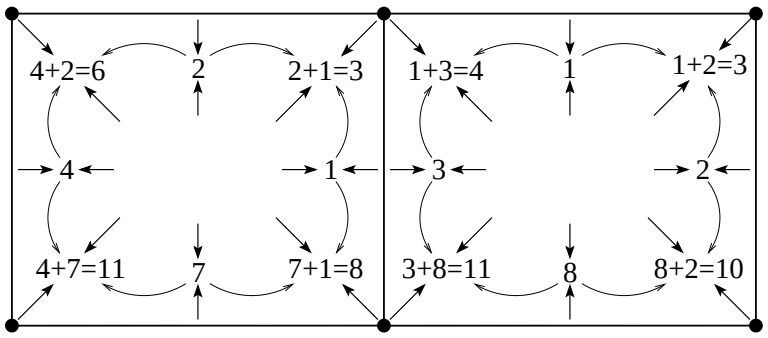

FIG. 4.8 – La méthode $dim\_inc(-1, plus, false)$ est appliquée à une $(2,1)$-chaîne et retourne une $(2,0)$-chaîne.

Un autre exemple est illustré à la figure 4.9. Nous avons une $(0,1)$-chaîne de départ et nous appliquons la méthode $dim\_inc(+1, plus, false)$. Les paramètres d'entrée de *dim_inc* indiquent que l'opération Ω entre les coefficients est l'addition et que l'on ne tient pas compte de l'orientation relative. Il en résulte une $(0,2)$-chaîne. Les calculs de *dim_inc* se font comme suit : nous avons que pour chacune des 0-cellules du complexe, nous considérons les 1-cellules et les 2-cellules adjacentes à la 0-cellule. Ensuite, nous additionnons les coefficients associés aux 1-cellules qui sont adjacentes à chacune des 0-cellules et nous emmagasinons le résultat dans le coefficient associé à la 0-cellule et à la 2-cellule.

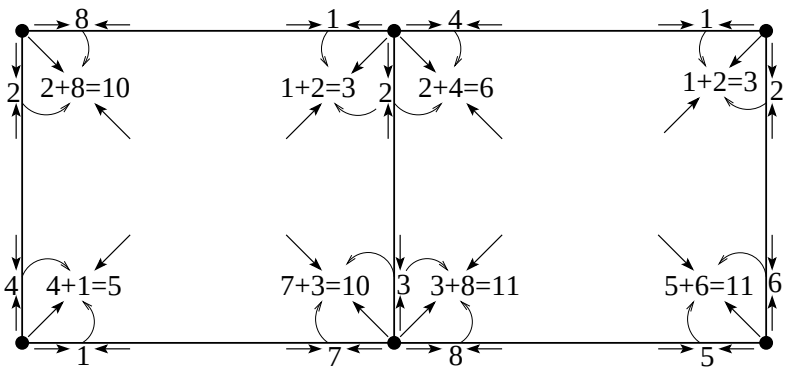

FIG. 4.9 – La méthode $dim\_inc(+1, plus, false)$ est appliquée à une $(0,1)$-chaîne et retourne une $(0,2)$-chaîne.

Chapitre 5

Exemples de systèmes avec les extensions

Dans ce chapitre, nous allons utiliser les extensions au modèle de chaînes proposées à la section 4.3 afin d'élaborer de nouveaux exemples. Dans un premier temps, nous allons illustrer comment nous pouvons calculer des moments de force dans un réseau de masses-ressorts. Il peut être intéressant de spécifier un angle de repos dans un réseau de masses-ressorts, au lieu d'ajouter des ressorts, comme ce fut le cas dans l'exemple du drapeau (ressorts diagonaux, sous-section 3.1.2). Dans un deuxième temps, nous allons développer un système modélisant un fluide et le déplacement de particules dans celui-ci.

5.1 Réseau de masses-ressorts avec des moments de force

Dans cette section, nous allons ajouter des moments de force dans le réseau de masses-ressorts de la section 3.1.1. Nous définissons des 2-cellules sur le complexe pour ensuite spécifier des angles de repos à chacune des paires de cellules 2-cellule,

0-cellule (la 0-cellule et la 2-cellule doivent être adjacentes). Ces quantités sont conservées dans une $(2,0)$-chaîne.

Un usager, après avoir construit un objet géométrique en trois dimensions, voudrait que son objet (complexe cellulaire) puisse être immergé dans un environnement physique avec des forces (gravité, entre autres), de sorte qu'il rebondisse lorsqu'il entre en collision avec un autre objet. Nous avons fait cela avec un tetrahèdre et un plan (plancher) à la sous-section 3.1.1 en spécifiant des ressorts aux arêtes. Mais si nous construisons un cube, nous devrons définir des arêtes (ressorts) supplémentaires afin que le cube garde approximativement sa forme.

En ajoutant des moments de force, nous n'avons pas pour le cube à ajouter des arêtes. En affectant tous les angles de repos à $\pi/2$, le cube gardera approximativement sa forme. Nous pouvons utiliser, pour plusieurs formes d'objets, des moments de force, sans avoir à spécifier de nouvelles arêtes, et obtenir des résultats satisfaisants [30]. Il existe un système [32] avec des particules orientées qui utilise ce principe. Les objets dans ce système peuvent, par contre, se séparer en plusieurs morceaux (changement dans la topologie), ce qui n'est pas notre cas.

Voyons d'abord le concept de *moment de force* (*torque* en anglais).

5.1.1 Moment de force

À la figure 5.1, nous avons trois masses reliées par deux ressorts. Un moment de force $\vec{T}$ (le vecteur sort de la page) est produit au point de pivot sur la masse m_2 lorsque le système n'est pas en position de repos (la position de repos est indiquée en pointillé sur la figure). Nous avons la relation suivante [33, page 303] avec le moment de force $\vec{T}$:

$$\vec{T} = \vec{r} \times \vec{F} \tag{5.1}$$

où "$\times$" est un produit vectoriel dans un système droitier et $\vec{r}$ est la différence de positions entre la masse m_1 et la masse m_2 et où le vecteur $\vec{F}$ est la force

appliqué sur la masse m_1. Cette formule est habituellement utilisée afin de calculer le moment de force $\vec{T}$ lorsqu'une force $\vec{F}$ est appliquée sur la masse m_1. Nous calculerons plutôt le moment de force $\vec{T}$ (lorsque le système n'est pas au repos) et nous en déduirons la force $\vec{F}$.

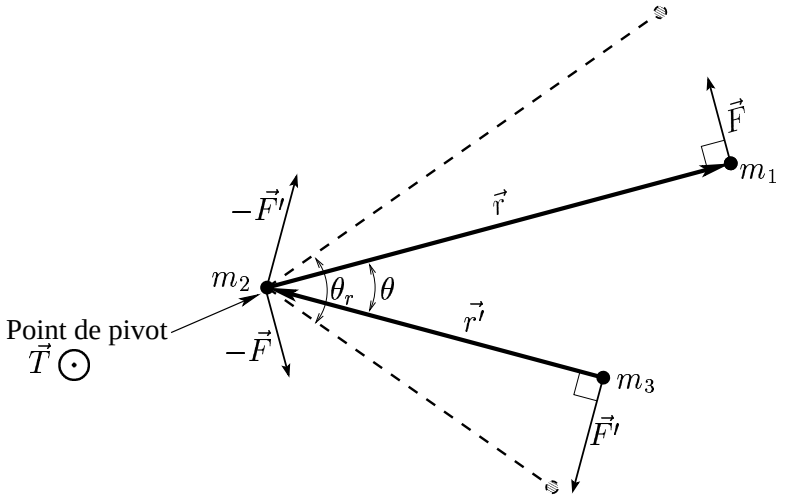

FIG. 5.1 – Moment de force calculé à partir de la différence entre l'angle θ et l'angle de repos θ_r.

Le moment $\vec{T}$ est orthogonal au plan P défini par les positions des trois masses (nous supposons qu'elles ne sont pas colinéaires). Soit :

$$\vec{v_n} = \frac{\vec{r} \times \vec{r'}}{|\vec{r} \times \vec{r'}|}, \tag{5.2}$$

avec $\vec{r'}$ étant la différence de positions entre la masse m_2 et la masse m_3. Ainsi, $\vec{v_n}$ est unitaire et orthogonal au plan P.

Avec ("$\cdot$" désigne le produit scalaire entre deux vecteurs) :

$$\cos\theta = \frac{\vec{r}}{|\vec{r}|} \cdot -\frac{\vec{r'}}{|\vec{r'}|} = -\left(\frac{\vec{r}}{|\vec{r}|} \cdot \frac{\vec{r'}}{|\vec{r'}|}\right), \tag{5.3}$$

(θ est l'angle formé par les deux ressorts) nous faisons l'hypothèse que le moment

de force est donné par :

$$\vec{T} = \vec{v_n}\ [k_a(\cos\theta - \cos\theta_r)], \tag{5.4}$$

où θ_r est l'angle lorsque le système est au repos et k_a est la constante de moment de force (pour un $\cos\theta - \cos\theta_r$ donné, plus la constante k_a est grande, plus l'amplitude du moment de force sera grande). Nous avons alors :

$$|\vec{T}| = |k_a(\cos\theta - \cos\theta_r)|.$$

Normalement, il y a plusieurs vecteurs $\vec{F}$ sur le plan P qui satisfont l'équation 5.1. Nous choisissons celui qui est orthogonal à $\vec{r}$. Étant donné cela, nous avons la relation suivante :

$$|\vec{T}| = |\vec{r}||\vec{F}|$$

ou

$$|\vec{F}| = \frac{|\vec{T}|}{|\vec{r}|}.$$

Nous avons alors :

$$\vec{F} = \frac{\vec{T} \times \frac{\vec{r}}{|\vec{r}|}}{|\vec{r}|}. \tag{5.5}$$

Avec $\vec{T}$ et $\vec{r}$, nous pouvons ainsi déduire la force $\vec{F}$ à la masse m_1. À la masse m_3, nous appliquons la même formule mais avec $\vec{r'}$ et $\vec{F'}$. Lorsque la norme de $\vec{r}$ diminue, la norme de la force $\vec{F}$ augmente. Ainsi, la norme de la force $\vec{F'}$ indiquée à la figure 5.1 est plus grande que la norme de $\vec{F}$ car la norme de $\vec{r'}$ est plus petite que la norme de $\vec{r}$.

Dans un système, la somme de toutes les forces doit être égale à zéro. Dans notre système, nous appliquons alors les forces $-\vec{F}$ et $-\vec{F'}$ sur la masse m_2. Ainsi, nous remarquons à la figure 5.1 que pour un moment de force $\vec{T}$ nous obtenons quatre forces ($\vec{F}$, $-\vec{F}$, $\vec{F'}$, $-\vec{F'}$) dont la somme est égale à zéro.

Il faudra propager ces quatre forces dans un réseau de masses-ressorts pour un moment de force à chacune des paires de cellules 2-cellule, 0-cellule, qui sont adjacentes. Les moments de force sont emmagasinés dans une $(2,0)$-chaîne. Heureusement, nous avons les méthodes sur les chaînes de sous-chaînes afin de faciliter la description de la propagation de toutes ces forces qui résultent des moments de force. Ceci est détaillé à la sous-section 5.1.3.

5.1.2 Construction du complexe

Jusqu'à présent dans la thèse, les exemples définis avec des chaînes ne nécessitaient pas des orientations de cellules particulières. Nous supposions qu'il y avait une seule $(p-2)$-cellule entre deux $(p-1)$-cellules ajoutées successivement à une p-cellule afin que le système puisse calculer les orientations relatives. Avec les complexes respectant cela, les expressions avec des chaînes utilisant l'orientation relative étaient toujours valides.

Dans notre exemple qui calcule des moments de force, nous avons besoin d'un vecteur orthogonal au plan P, soit $\vec{v}_n$ défini à la sous-section précédente. Ce vecteur doit être orienté de façon cohérente. Pour être plus précis, nous exigeons que les 2-cellules du complexe soient orientées de façon cohérente [1, section 3.3] : pour toutes les paires de 2-cellules C_i^2, C_j^2, partageant une 1-cellule C_k^1, celles-ci doivent avoir une orientation relative inverse avec la 1-cellule. C'est-à-dire que : $C_i^2.orientation(C_k^1) = -C_j^2.orientation(C_k^1)$.

Nous avons un exemple d'un complexe illustré à la figure 5.2 qui possède cette propriété. Pour avoir toutes les 2-cellules orientées de façon cohérente, il s'agit de porter attention à la première 1-cellule que l'on ajoute à celles-ci lors de son assemblage, car c'est la première 1-cellule ajoutée qui détermine l'orientation de la 2-cellule. Dans notre complexe à la figure 5.2, les premières 1-cellules ajoutées aux 2-cellules C_i^2 sont les 1-cellules C_i^1. (Nous ne suggérons pas ici que nous pouvons satisfaire cette propriété pour tous les complexes : on ne peut pas satisfaire cette

propriété avec une bande de Möbius par exemple.)

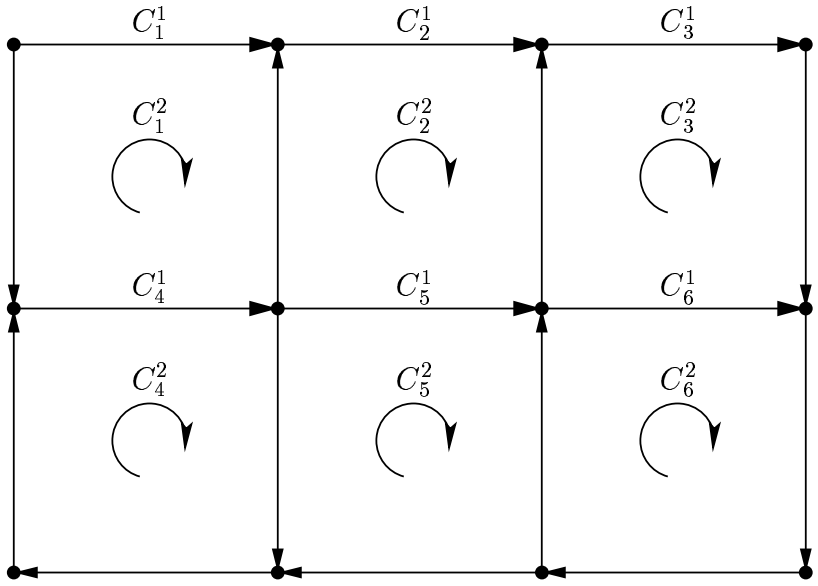

FIG. 5.2 – Exemple d'un 2-complexe dont les 2-cellules sont orientées de façon cohérente.

5.1.3 Moments de force avec des chaînes

Notre but est de calculer une 0-chaîne de somme de forces $\vec{F}$ qui résultent des moments de force. Nous pourrons ainsi ajouter la somme de forces $\vec{F}$ aux autres forces qui agissent à chacune des 0-cellules du réseau de masses-ressorts. Nous allons procéder de la façon suivante :

- calculer les vecteurs $\vec{r}$ dans la direction des 1-cellules ;
- calculer les cosinus des angles actuels $\cos\theta$;
- calculer les différences entre les cosinus des angles actuels et les cosinus des angles de repos, soit $\cos\theta - \cos\theta_r$;
- calculer les moments de force $\vec{T}$ dans une $(2,0)$-chaîne ;

- calculer les sommes de forces $\vec{F}$ qui résultent des moments de force dans une $(2,1)$-chaîne ;
- calculer les sommes de forces $\vec{F}$ qui résultent des moments de force dans une $(2,0)$-chaîne ;
- et finalement, calculer les sommes de forces $\vec{F}$ qui résultent des moments de force dans une 0-chaîne ;

Pour ce faire, nous avons besoin des chaînes suivantes :

- *chain*20_*ka* sur R (constante de moment de force k_a pour chacune des paires 2-cellule, 0-cellule adjacentes) ;
- *chain*20_*vn* sur R^3 (vecteur $\vec{v}_n$ orthogonal au plan P pour chacune des paires 2-cellule, 0-cellule adjacentes) ;
- *chain*1_*r* sur R^3 (vecteur $\vec{r}$ selon l'orientation de la 1-cellule pour chacune des 1-cellules) ;
- *chain*21_*r* sur R^3 (vecteur $\vec{r}$ selon l'orientation de la 1-cellule, pour chacune des paires 2-cellule, 1-cellule adjacentes) ;
- *chain*1_*r_norm* sur R^3 (vecteur $\frac{\vec{r}}{|\vec{r}|}$ selon l'orientation de la 1-cellule, pour chacune des 1-cellules) ;
- *chain*21_*r_norm* sur R^3 (vecteur $\frac{\vec{r}}{|\vec{r}|}$ selon l'orientation de la 1-cellule, pour chacune des paires 2-cellule, 1-cellule adjacentes) ;
- *chain*20_*cos_theta* sur R (la valeur $\cos\theta$ pour chacune des paires 2-cellule, 0-cellule adjacentes) ;
- *chain*20_*cos_theta_rest* sur R (la valeur $\cos\theta_r$ pour chacune des paires 2-cellule, 0-cellule adjacentes) ;
- *chain20_diff_cos_theta* sur R (la valeur $\cos\theta - \cos\theta_r$ pour chacune des paires 2-cellule, 0-cellule adjacentes) ;
- *chain*20_T sur R^3 (moment de force $\vec{T}$ pour chacune des paires 2-cellule, 0-cellule adjacentes) ;

- *chain*21_T sur R^3 (somme des moments de force $\vec{T}$ pour chacune des paires 2-cellule, 1-cellule adjacentes) ;
- *chain*20_F sur R^3 (somme de sommes de forces $\vec{F}$ qui résultent des moments de force pour chacune des paires 2-cellule, 0-cellule adjacentes) ;
- *chain*21_F sur R^3 (somme de forces $\vec{F}$ qui résultent des moments de force pour chacune des paires 2-cellule, 1-cellule adjacentes) ;
- *chain*0_F sur R^3 (somme totale de forces $\vec{F}$ qui résultent des moments de force pour chacune des 0-cellules) ;

Ayant défini ces chaînes, nous pouvons maintenant décrire le processus avec notre interface de programmation.

Dans un premier temps, nous allons calculer avec notre interface de programmation, la 1-chaîne `chain1_r` qui contient le vecteur $\vec{r}$ pour chacune des 1-cellules en considérant la position[1] et l'orientation relative des 0-cellules qui la composent. Nous procédons comme suit :

`chain1_r` = `network.chain0_p.dim_inc`$(+1,$ `plus`, `true`$)$.

Les vecteurs calculés dans R^3 correspondent aux vecteurs sur les 1-cellules que l'on retrouve par exemple à la figure 5.2. Notez que les vecteurs à cette figure servent à indiquer l'orientation des 1-cellules, mais les vecteurs $\vec{r}$ seraient illustrés de la même façon.

Nous calculons aussi ces vecteurs normalisés :

`chain1_r_norm` = `chain1_r.normalize`$()$.

Maintenant, nous allons convertir cette 1-chaîne à une $(2,1)$-chaîne :

`chain21_r_norm` = `chain1_r_norm.in`$(+1,$ `true`$)$.

[1]Rappelons que nous avons une géométrie définie sur le réseau de masses-ressorts. En particulier, nous avons la 0-chaîne *network.chain*0_p qui indique les positions des 0-cellules.

Cela nous permet d'avoir les vecteurs dans R^3 sur les 1-cellules modifiés par l'orientation relative des 2-cellules avec chacune de ses 1-cellules. À la figure 5.3, nous avons une illustration de la $(2,1)$-chaîne `chain21_r_norm` qui est définie à partir de la 1-chaîne `chain1_r_norm` du complexe de la figure 5.2. Entre chacune des paires de flèches droites pointant l'une vers l'autre, nous avons le coefficient (le vecteur $\frac{\vec{r}}{|\vec{r}|}$) impliquant une paire 2-cellule, 1-cellule adjacentes.

Nous faisons de même avec la 1-chaîne `chain1_r`, soit :

$$\texttt{chain21\_r} = \texttt{chain1\_r.in}(+1, \texttt{true}).$$

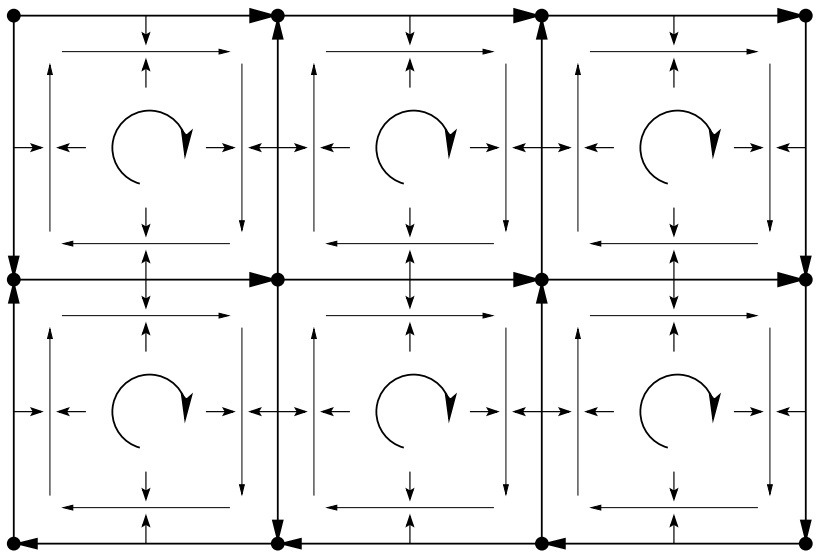

FIG. 5.3 – Une $(2,1)$-chaîne est définie à partir de la chaîne `chain1_r_norm`.

Avec la $(2,1)$-chaîne `chain21_r_norm`, nous pouvons calculer les cosinus des angles θ (équation 5.3) dans une $(2,0)$-chaîne, comme suit :

$$\texttt{chain20\_cos\_theta} = -\texttt{chain21\_r\_norm.dim\_inc}(-1, \texttt{mult}, \texttt{false})$$
$$.\texttt{sum\_components}().$$

Sur la chaîne `chain21_r_norm`, la méthode `dim_inc` est appliquée afin d'obtenir une $(2,0)$-chaîne. L'opération Ω (indiquée par le deuxième paramètre de `dim_inc`, soit `mult`) est la multiplication, composante par composante, et l'on ne tient pas compte de l'orientation relative. Ensuite la méthode `sum_components` est appliquée à cette $(2,0)$-chaîne sur R^3 qui résulte en une $(2,0)$-chaîne sur R. La méthode `sum_components` calcule la somme des composantes des vecteurs dans R^3. Après avoir appliqué l'opération unaire "$-$", nous obtenons alors dans la $(2,0)$-chaîne `chain20_cos_theta` le cosinus de l'angle θ (un nombre élément de R) pour chacune des paires 2-cellule c^2, 0-cellule c^0 adjacentes. Notez que cela est équivalent à faire le produit scalaire entre les deux vecteurs $\frac{\vec{r}}{|\vec{r}|}$ et $-\frac{\vec{r'}}{|\vec{r'}|}$ dans la chaîne `chain21_r_norm` qui sont associés à la 2-cellule c^2 et les deux 1-cellules adjacentes à la 0-cellule c^0, puis d'emmagasiner le résultat dans la chaîne `chain20_cos_theta` dans le coefficient associé à la 2-cellule c^2 et la 0-cellule c^0 (le processus est répété pour chacune des paires 2-cellule, 0-cellule adjacentes). Mais nous utilisons plutôt la somme des composantes et le produit, composante par composante, car le produit scalaire ne peut pas être utilisé comme opération Ω dans la méthode *dim_inc* (celui-ci n'est pas une opération commutative).

Nous calculons ensuite $\cos\theta - \cos\theta_r$ comme suit :

$$\texttt{chain20\_diff\_cos\_theta} = \texttt{chain20\_cos\_theta} - \texttt{chain20\_cos\_theta\_rest}.$$

Nous voilà prêts à calculer la $(2,0)$-chaîne de moments de force $\vec{T}$:

$$\texttt{chain20\_T} = \texttt{chain20\_vn} * \texttt{chain20\_ka} * \texttt{chain20\_diff\_cos\_theta};$$

ce qui correspond à l'équation 5.4, de la sous-section 5.1.1.

Nous discuterons à la fin de cette sous-section sur la façon de calculer la chaîne `chain20_vn` qui contient les vecteurs $\vec{v}_n$.

Il faut maintenant propager les quatre forces qui résultent de chacun des moments de force. Mais avant cela, examinons en détail ce que nous devons faire dans une 2-cellule, pour chacune des 1-cellules. Il y a deux 0-cellules c_1^0 et c_2^0 adjacentes

à la 1-cellule. Des moments de force $\vec{T_1}$ et $\vec{T_2}$ sont respectivement calculées à c_1^0 et à c_2^0. Nous avons aussi les forces $\vec{F_1}$ et $\vec{F_2}$ qui résultent des moments $\vec{T_1}$ et $\vec{T_2}$. Les forces $-\vec{F_1}$ et $\vec{F_2}$ devront être appliquées à c_1^0 et les forces $-\vec{F_2}$ et $\vec{F_1}$ devront être appliquées à c_2^0. Car comme nous l'avons vu à la sous-section 5.1.1, nous soustrayons la force résultante à la masse (0-cellule) associée au moment de force, et nous additionnons la force résultante à l'autre masse. Nous le répétons ici, nous ne considérons qu'une seule 1-cellule pour l'instant.

En posant que l'orientation relative entre la 1-cellule avec la 0-cellule c_1^0 est de $+1$ et avec la 0-cellule c_2^0, de -1 ; voici comment nous procédons :

1. nous additionnons les moments de force $\vec{T_1}$ et $\vec{T_2}$ sur la 1-cellule en tenant compte de l'orientation relative, nous avons alors $\vec{T_1} - \vec{T_2}$;
2. nous inversons le signe de cette somme de moments pour obtenir $\vec{T_2} - \vec{T_1}$;
3. nous calculons la somme des forces avec cette somme de moments de force, soit $\vec{F_2} - \vec{F_1}$;
4. nous distribuons cette somme de forces $\vec{F_2} - \vec{F_1}$ sur les 0-cellules en tenant compte de l'orientation relative.

Nous obtenons ainsi ce que nous voulons, soit $\vec{F_2} - \vec{F_1}$ sur la 0-cellule c_1^0 et $\vec{F_1} - \vec{F_2}$ sur la 0-cellule c_2^0.

Il faut garder à l'esprit par contre, qu'une 0-cellule a deux 1-cellules adjacentes. La somme de forces est alors distribuée pour chacune des 1-cellules adjacentes.

Afin de cumuler dans le complexe, toutes les sommes de forces à chacune des 0-cellules nous utilisons deux $(2,0)$-chaînes et une $(2,1)$-chaîne. Nous allons dans un premier temps additionner dans la $(2,1)$-chaîne `chain21_T` les moments de force provenant de la $(2,0)$-chaîne `chain20_T` (correspond à l'étape 1 et 2 ci-dessus, le signe "$-$" à l'étape 2). Ceci est fait dans notre interface de programmation comme suit :

$$\texttt{chain21\_T} = -\texttt{chain20\_T.dim\_inc}(+1, \texttt{plus}, \texttt{true}). \tag{5.6}$$

L'opération Ω utilisée est l'addition et l'on tient compte de l'orientation relative. On illustre à la figure 5.4, sur la partie de gauche, ce que fait dans ce cas la méthode `dim_inc`. Seulement une 2-cellule[2] du complexe est représentée avec $\vec{T}_1$, $\vec{T}_2$ et $\vec{T}_3$ qui sont les valeurs des moments de force $\vec{T}$ pour chacune des paires 2-cellule, 0-cellule.

Ensuite nous calculons sur chacune des 1-cellules la somme des forces qui est produite par la somme des moments de force (étape 3, ci-dessus) :

$$\texttt{chain21\_F} = (\texttt{chain21\_T} \wedge \texttt{chain21\_r\_norm})/\texttt{chain21\_r.norm}().$$

Notez que la méthode `norm` retourne la 2-norme (norme euclidienne) et que l'opération "$\wedge$" correspond au produit vectoriel dans notre interface de programmation. Cette expression avec des chaînes correspond à l'équation 5.5 de la sous-section 5.1.1 et est représentée par les équations au centre de la figure 5.4 avec $\vec{f}_1$, $\vec{f}_2$ et $\vec{f}_3$ qui sont les sommes des forces pour chacune des paires 2-cellule, 1-cellule et avec $\vec{r}_1$, $\vec{r}_2$ et $\vec{r}_3$ qui sont les vecteurs $\vec{r}$ pour chacune des 1-cellules.

Maintenant, il s'agit de convertir la $(2,1)$-chaîne `chain21_F` dans la $(2,0)$-chaîne `chain20_F`. Ceci est illustré à la figure 5.4 sur la partie de droite. Avec notre interface de programmation, il suffit de faire (correspond à l'étape 4) :

$$\texttt{chain20\_F} = \texttt{chain21\_F.dim\_inc}(-1, \texttt{plus}, \texttt{true});$$

l'opération Ω utilisée est l'addition et l'on tient compte de l'orientation relative.

Nous avons maintenant quatre forces[3] à chacune des paires 2-cellule, 0-cellule, qui résultent des moments de force aux paires 2-cellule, 0-cellule. Il nous faut maintenant cumuler les forces de la $(2,0)$-chaîne `chain20_F` dans la 0-chaîne `chain0_F`. Autrement dit, nous cumulons, dans le coefficient associé à chacune des 0-cellules

[2]Nous avons à la figure une 2-cellule triangulaire, mais les 2-cellules peuvent avoir plus de trois 1-cellules adjacentes.

[3]Quatre forces, car c'est la somme de deux sommes de deux forces.

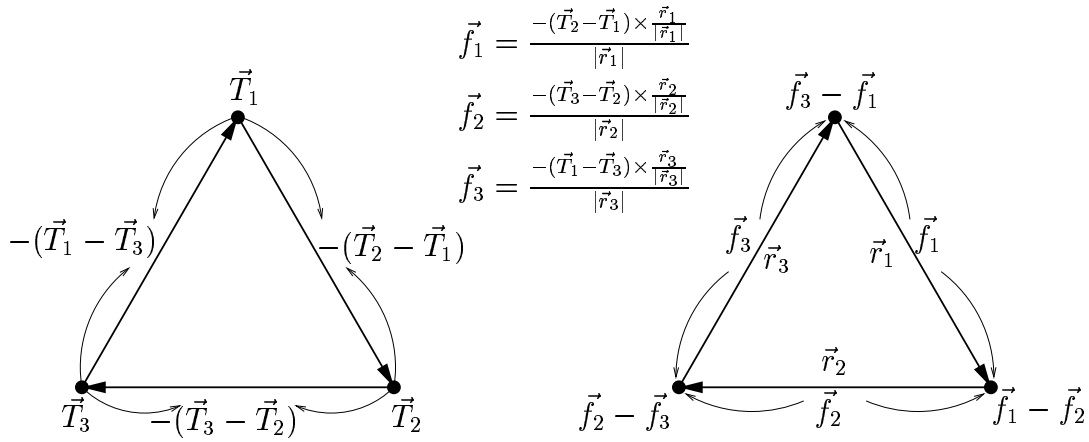

FIG. 5.4 – Passage d'une $(2,0)$-chaîne de moments de force à une $(2,0)$-chaîne de forces.

(dans `chain0_F`), toutes les sommes de forces associées à chacune des paires 2-cellule (adjacente à la 0-cellule), 0-cellule. Il suffit de faire :

`chain02_F` = `chain20_F.transpose`();

et

`chain0_F` = `chain02_F.sum`();

ou simplement :

`chain0_F` = `chain20_F.transpose().sum`().

Par la suite, nous ajoutons ces forces aux 0-cellules comme les autres forces dans le réseau de masses-ressorts à la section 3.1.1, c'est-à-dire que nous additionnons la 0-chaîne `chain0_F` en plus des autres forces à la chaîne *chain*0_*f*, soit :

`chain0_f` = `chain0_f_spring` + `chain0_f_g` + `chain0_f_damp` + `chain0_F`.

Maintenant voyons comment nous calculons les vecteurs $\vec{v}_n$ (équation 5.2). Nous n'avons pas trouvé de façon de définir la chaîne de sous-chaînes *chain*20_*vn*

qui contient les vecteurs $\vec{v}_n$ avec une expression de chaînes. Nous avons dû effectuer deux boucles imbriquées. Voici donc le pseudo-code qui a été utilisé pour calculer cette chaîne de sous-chaînes :

```
pour toutes cellules C_i^2 du complexe K faire
    pour j égal 1 à C_i^2.nb_adj(-1) faire
        c_1^1 ← C_i^2.adj(j, -1) ;
        si j = C_i^2.nb_adj(-1) alors
            c_2^1 ← C_i^2.adj(1, -1) ;
        sinon
            c_2^1 ← C_i^2.adj(j + 1, -1) ;
        c^0 ← c_1^1.intersection(c_2^1) ;
        chain20_vn.coefficient(C_i^2, c^0) ←
                chain21_r_norm.coefficient(C_i^2, c_2^1) ∧
                chain21_r_norm.coefficient(C_i^2, c_1^1) ;
```

Notez que la nouvelle méthode *intersection* de la classe *Cell* a en entrée une cellule c_2^p de la même dimension que l'instance c_1^p qui appelle la méthode. Elle retourne, quand elle existe, la cellule de dimension $p-1$ qui est adjacente (ou qui est la frontière) aux p-cellules c_1^p et c_2^p.

Même si les 2-cellules du complexe sont orientées de façon cohérente, en choisissant l'ordre dans le produit vectoriel à la fin de cet algorithme, nous supposons que les 1-cellules ont été ajoutées dans le sens de l'orientation des 2-cellules.

Avec ce programme utilisant notre interface de programmation, l'usager peut ainsi construire un réseau de masses-ressorts en spécifiant le cosinus des angles de repos[4] $\cos\theta_r$ dans la chaîne `chain20_theta_rest` pour chacune des paires 2-cellule,

[4]En fait, il peut spécifier les angles θ_r et calculer la chaîne `chain20_theta_rest`.

0-cellule adjacentes. Il peut s'éviter de spécifier les angles de repos, si le complexe géométrique qu'il a construit est dans une configuration qui lui convient. Il suffit, lors de l'initialisation, de calculer la chaîne `chain20_cos_theta` et d'affecter celle-ci à la chaîne `chain20_theta_rest`.

5.2 Fluide

Dans cette section, nous allons discuter d'une modélisation d'un fluide avec des particules se déplaçant à l'intérieur de celui-ci. Ce modèle pourrait être utilisé par exemple, pour la simulation d'air et de fumée.

À la section 5.1, nous avons utilisé un modèle qui comporte des coefficients qui doivent être emmagasinés dans des chaînes de sous-chaînes (comme par exemple, les moments de force).

Dans l'exemple que nous allons élaborer dans cette section, nous avons au départ un modèle, dont la description des quantités au temps t est dans une *chaîne sur des coefficients primitifs* (chaînes comme à la section 2.4), c'est-à-dire des chaînes avec des coefficients qui ne sont pas des sous-chaînes.

Pour faire évoluer ces coefficients dans le temps, nous nous sommes aperçus qu'il y avait des limitations à utiliser seulement des chaînes de coefficients primitifs : il était très difficile, dans certaines situations, d'effectuer les calculs voulus. Nous avons utilisé des chaînes de sous-chaînes afin de faire évoluer les chaînes de coefficients décrivant notre système dans le temps.

Un autre problème intéressant, qui est plus facile à résoudre avec des chaînes de sous-chaînes, est un des sous-problèmes que nous développons dans cette section. Ce sous-problème est le calcul de l'aire de toutes les 2-cellules triangulaires du complexe géométrique *Complex_p* (que nous utilisons dans la modélisation du fluide). C'est un petit problème, facile à comprendre, qui est discuté à la sous-section 5.2.2 ci-dessous.

Pour décrire le modèle plus précisément, nous avons au départ un modèle de chaînes de coefficients primitifs au temps t. Ensuite, des chaînes de sous-chaînes sont définies à partir de celles-ci. Nous faisons ensuite évoluer ces chaînes de sous-chaînes pour une tranche de temps Δt, pour finalement les convertir en chaînes de coefficients primitifs. Ces chaînes de coefficients primitifs décrivent ainsi le système au temps $t + \Delta t$.

Cet exemple consiste à modéliser le comportement d'un fluide et de particules se déplaçant dans celui-ci. Pour ce faire, nous allons utiliser des heuristiques. Depuis longtemps, la physique modélise le comportement de fluides grâce aux équations de Navier-Stokes. En infographie, une forme simplifiée de ces équations et une subdivision de l'espace de façon régulière (voxel) et grossière a permis de produire des animations réalistes [16, 17] et ce, en beaucoup moins de temps que des simulations plus précises. Lors d'une simulation en infographie, nous n'avons pas besoin que le résultat soit très précis, nous avons seulement besoin qu'il ait l'air réaliste.

Nous avons développé pour modéliser les fluides, des heuristiques basées sur la physique avec une décomposition triangulaire irrégulière. Comme toutes les heuristiques, elles ne sont pas parfaites, mais nous voulons démontrer qu'il est relativement facile d'implanter ces heuristiques avec notre système. Il est facile d'imaginer des comportements d'un système dans un complexe cellulaire quelconque, mais faire une implantation qui calculera ces comportements est une toute autre histoire. Notre système, avec l'ajout des chaînes de sous-chaînes, permet de décrire plus facilement de tels comportements.

Notez que pour modéliser les fluides dans [16, 17], ils utilisent le fait que l'espace est subdivisé de façon régulière et les équations sont formulées en fonction de cela. Notre cadre de travail n'étant pas basé sur un complexe régulier, il est vrai que l'utilisation de celui-ci est moins approprié dans ce cas, car les simplifications proviennent du fait qu'il s'agit d'un cas très spécial.

5.2.1 Heuristiques pour un fluide

Nous allons décrire les heuristiques qui modélisent le comportement d'un fluide, tel que l'air. Contrairement à [16, 17], nous ne supposons pas que le fluide soit incompressible. Cela nous évite d'utiliser un algorithme numérique itératif à chacune des étapes de temps t. Cependant les résultats sont sans doute moins précis, mais nous avons tout de même obtenu des résultats très satisfaisants [30].

Nous avons utilisé un espace à deux dimensions pour modéliser le fluide, ceci pour limiter les temps de calcul et pour faciliter le rendu. Par contre, grâce à la formulation avec des chaînes, il serait relativement facile de modéliser un fluide en trois dimensions (et même en quatre dimensions, comme dans [29]). Il y aurait, entre autres, la partie concernant la position d'une particule où plus d'analyse serait à effectuer (voir la sous-section 5.2.4), et aussi, la partie concernant le calcul des vecteurs normaux aux faces (voir la sous-section 5.2.3).

Nous partons du principe que chacune des 2-cellules contient une certaine masse de fluide ayant une vitesse moyenne. Nous utilisons le complexe géométrique *Complex_p* pour définir la géométrie sur notre complexe[5].

Nous avons alors des quantités au temps t qui sont associées à chacune des 2-cellules, soient :

- la masse m ;
- la vitesse moyenne $\vec{v}_a$;
- la pression p ($p = m/s$, où s est l'aire de la 2-cellule géométrique).

Nous utilisons quatre principes qui définissent les quantités qui sont associées aux 2-cellules, au temps $t + \Delta t$:

1. une masse de fluide à une 2-cellule se déplace dans la direction de $\vec{v}_a$, et une partie est transférée aux 2-cellules voisines[6] ;

[5] En fait, l'équivalent du complexe géométrique *Complex_p*, mais en deux dimensions.

[6] Nous utilisons ici l'adjectif "voisines" pour décrire que deux p-cellules c_1 et c_2 ne sont pas seulement adjacentes, mais qu'il existe un j, tel que $c_1 = c_2.adj(j, 0^-)$.

2. une autre partie de la masse de fluide à une 2-cellule se déplace vers les 2-cellules voisines qui ont moins de pression qu'elle ;
3. il y a de la friction entre une masse de fluide à une 2-cellule et les masses aux 2-cellules voisines ; il y a aussi de la friction interne à une 2-cellule ;
4. la direction de la vitesse moyenne $\vec{v}_a$ à une 2-cellule qui touche la frontière de l'objet[7], a tendance à suivre la frontière.

Dans cette sous-section, nous allons définir les heuristiques seulement pour une 2-cellule avec ses 2-cellules voisines (lorsque l'heuristique s'applique). Dans la prochaine sous-section (5.2.3), avec les chaînes de sous-chaînes, nous appliquerons globalement les heuristiques sur tout le complexe.

Premier principe

Examinons plus en détail, le premier principe. Lorsqu'une masse de fluide dans une 2-cellule, se déplace dans une direction $\vec{v}_a$, une partie de la masse quittera la 2-cellule et ira dans les 2-cellules voisines[8]. Pour une 2-cellule triangulaire, nous utilisons l'heuristique suivante afin d'estimer la masse du fluide m_{v_out} qui quitte une 2-cellule en Δt temps :

$$m_{v\_out} = \frac{|\vec{v}_a|\Delta t}{l_a} m, \tag{5.7}$$

où l_a est la longueur moyenne de la géométrie des 1-cellules qui composent la 2-cellule. Notez que ce n'est qu'une approximation. La figure 5.5 illustre l'heuristique que nous avons composée. En faisant l'hypothèse que la masse m est distribuée également sur la 2-cellule, il s'agit d'évaluer la proportion de la surface de la 2-cellule qui ne sera plus dans la 2-cellule après un temps Δt. Elle aura parcouru une distance $\vec{v}_a \Delta t$. La proportion est évaluée approximativement par $\frac{|\vec{v}_a|\Delta t}{l_a}$ et le tout est multiplié par la masse m, pour obtenir m_{v_out}.

[7]Toutes les 1-cellules qui ont une seule 2-cellule adjacente font partie de la frontière de l'objet.

[8]Une masse de fluide peut aussi se déplacer d'une 2-cellule voisine vers la 2-cellule, mais le phénomène dans son ensemble sera traité à la prochaine sous-section.

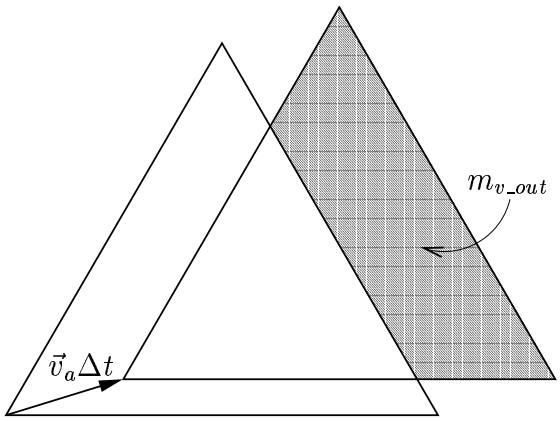

FIG. 5.5 – Une partie de la masse quitte la 2-cellule après un temps Δt.

Nous illustrons à la figure 5.6 comment est distribuée la masse m_{v_out} entre les 2-cellules voisines pour une 2-cellule c_4^2.

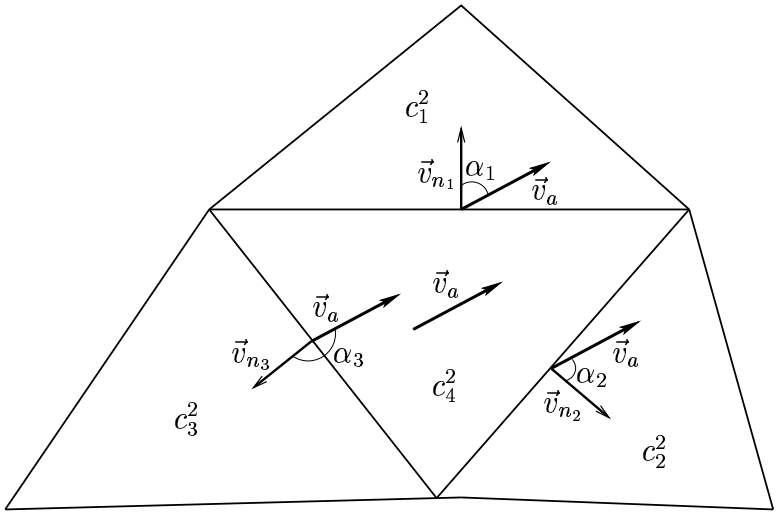

FIG. 5.6 – Propagation de la masse de fluide dans la direction $\vec{v}_a$.

Nous remarquons sur cette figure que, les vecteurs $\vec{v}_{n_i}$ sont orthogonaux aux géométries des 1-cellules qui sont les frontières entre la cellule c_4^2 et les 2-cellules c_i^2, $i = 1, 2, 3$. Ces vecteurs sont unitaires. Remarquez que les vecteurs $\vec{v}_{n_i}$, lorsqu'ils originent de la 1-cellule qui leur est associée, s'éloignent de la 2-cellule c_4^2. Les

angles α_i sont définis comme étant les angles entre $\vec{v}_a$ et les $\vec{v}_{n_i}$, $0 \leq \alpha_i \leq \pi$.

Nous choisissons comme heuristique, que plus α_i est petit, plus une portion de la masse m_{v_out} de fluide ira dans la 2-cellule c_i^2. Plus précisément, la masse $m_{v\_out_i}$ qui est transférée à la 2-cellule c_i^2 est de :

$$m_{v\_out_i} = \max\left\{0,\ (\frac{\vec{v}_a}{|\vec{v}_a|} \cdot \vec{v}_{n_i})\, m_{v\_out}\right\}. \tag{5.8}$$

Lorsque qu'une masse de fluide est transférée d'une 2-cellule à une autre 2-cellule, il faut mettre à jour les vitesses moyennes de chacune des 2-cellules. Ainsi dans notre exemple, la vitesse moyenne $\vec{v}_a$ à la cellule c_4^2 demeure inchangée après un temps Δt, et les vitesses moyennes $\vec{v}_{a_i}$ aux autres 2-cellules c_i^2 deviennent[9] :

$$\vec{v}_{a_i} \leftarrow \frac{m_i \vec{v}_{a_i} + m_{v\_out_i} \vec{v}_a}{m_i + m_{v\_out_i}}, \tag{5.9}$$

avec m_i étant les masses aux 2-cellules c_i^2 voisines de c_4^2.

À la fin de la tranche de temps, il faut aussi mettre à jour les masses associées aux 2-cellules. Ainsi la masse m à la 2-cellule c_4^2, sera de :

$$m \leftarrow m - \sum_i m_{v\_out_i} \tag{5.10}$$

et les masses m_i aux 2-cellules c_i^2 seront de :

$$m_i \leftarrow m_i + m_{v\_out_i}. \tag{5.11}$$

Remarquez que la sommation $\sum_i m_{v\_out_i}$ est approximativement égale à m_{v_out}, voilà pourquoi, afin de préserver la masse totale du système, nous soustrayons à m, $\sum_i m_{v\_out_i}$ plutôt que m_{v_out}. Remarquez aussi qu'il faut que cette sommation soit plus petite que m. Si tel n'est pas le cas, on peut entre autres diminuer la valeur de Δt.

[9]Remarques sur la notation. La variable $\vec{v}_{a_i}$ à gauche de l'affectation, est la vitesse moyenne, au temps $t + \Delta t$, et à droite de l'affectation, au temps t. Il en sera de même pour les autres équations impliquant la même variable des deux côtés de l'affectation. Aussi la notation $\vec{v}_{a_i}$, n'est pas satisfaisante, car le i est un indice sur la variable $\vec{v}_a$, et non pas sur une variable a. La notation précise devrait être $(\vec{v}_a)_i$, mais nous l'avons rejetée, afin d'avoir une notation plus simple. Des remarques similaires s'appliquent à $\vec{v}_{n_i}$, $m_{v\_out_i}$ et $m_{p\_out_i}$ (ce dernier étant défini plus loin).

Deuxième principe

Examinons maintenant le deuxième principe, qui dit qu'une masse de fluide à une 2-cellule se déplace vers les 2-cellules voisines qui ont moins de pression qu'elle. Utilisons encore le même exemple, en considérant qu'une partie de la masse de fluide à une 2-cellule c_4^2 peut se déplacer vers les 2-cellules voisines c_i^2 (nous ignorons encore pour l'instant le déplacement inverse, ceci est traité à la sous-section 5.2.3 ci-dessous).

Notez que les coefficients $\vec{v}_a$, $\vec{v}_{a_i}$, m et m_i, calculés à la fin de la tranche de temps peuvent être choisis ou non comme étant les valeurs à utiliser au début de la tranche de temps avant d'appliquer le deuxième principe. Sans avoir fait beaucoup d'expérimentations, il semble qu'en mettant à jour les coefficients immédiatement avant d'appliquer le deuxième principe, nous obtenons des résultats plus satisfaisants.

À la figure 5.7, nous avons respectivement des pressions ($p = m/s$) de 60, 75, 120 et 100, pour les 2-cellules c_1^2, c_2^2, c_3^2 et c_4^2. Les différences de pressions Δp_i entre la 2-cellule c_4^2 et les cellules c_i^2, $i = 1, 2, 3$, sont respectivement de $+40$, $+25$ et -20. Une différence positive indique que la pression est plus élevée dans la cellule c_4^2 qu'une 2-cellule voisine, et par conséquent, une partie de la masse de fluide à la cellule c_4^2 se déplacera vers la 2-cellule voisine.

Nous définissons la quantité de masse de fluide $m_{p\_out_i}$ qui quittera la 2-cellule c_4^2 vers la 2-cellule c_i^2 en raison de la différence de pressions, comme suit :

$$m_{p\_out_i} = \max\left\{0,\ \frac{\Delta p_i\ \Delta t}{p\ k_v}\, m\right\}, \tag{5.12}$$

avec k_v comme étant une mesure similaire à une constante de viscosité du fluide. Plus k_v est élevée, plus le fluide est visqueux et moins grande est la quantité de fluide transférée.

La masse de fluide $m_{p\_out_i}$ qui quitte la 2-cellule c_4^2 vers c_i^2, a une vitesse moyenne de $\vec{v}_a$ au temps t mais en quittant la 2-cellule, elle acquiert une vitesse

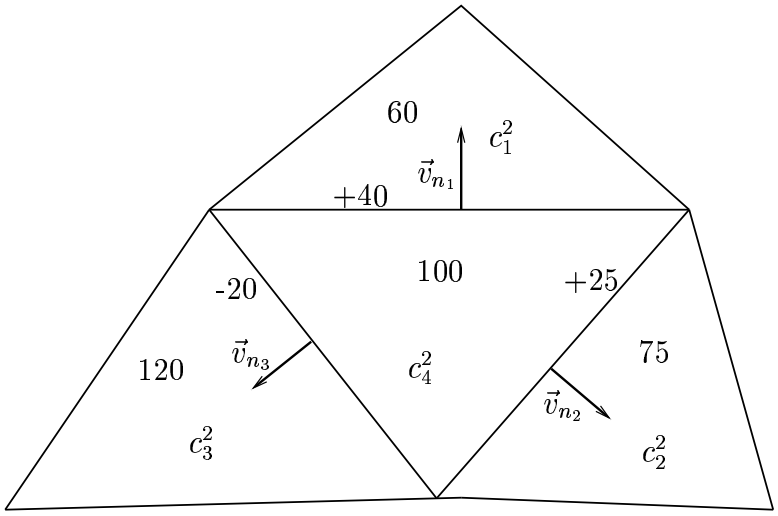

FIG. 5.7 – Propagation de la masse de fluide dans les directions où il y a moins de pression.

supplémentaire[10] en passant par la 1-cellule qui forme la frontière entre la 2-cellule c_4^2 et la 2-cellule c_i^2. La vitesse $\vec{v}_{p_i}$ de la masse de fluide $m_{p\_out_i}$ est calculée comme suit :

$$\vec{v}_{p_i} \leftarrow \vec{v}_a + k_p \vec{v}_{n_i}, \tag{5.13}$$

avec k_p qui est une constante pour indiquer l'influence sur la vitesse supplémentaire qu'acquiert la masse de fluide lorsqu'elle traverse la 1-cellule frontalière. Plus la constante est grande, plus l'influence est grande.

Les vitesses des masses de fluide pour les 2-cellules c_i^2 sont donc mises à jour comme suit :

$$\vec{v}_{a_i} \leftarrow \frac{m_i \vec{v}_{a_i} + m_{p\_out_i} \vec{v}_{p_i}}{m_i + m_{p\_out_i}}. \tag{5.14}$$

Il faut aussi mettre à jour la vitesse de la masse de fluide à la 2-cellule c_4^2. Lorsqu'une partie de la masse de fluide quitte une 2-cellule en raison d'une différence de

[10] Ceci en raison de la différence de pressions.

pressions, nous pouvons penser que pendant son trajet, elle influencera la vitesse $\vec{v}_a$ de la masse de fluide. Nous utilisons l'heuristique suivante :

$$\vec{v}_a \leftarrow \frac{m\vec{v}_a + \sum_i m_{p\_out_i}\vec{v}_{p_i}}{m + \sum_i m_{p\_out_i}}. \tag{5.15}$$

Il faut aussi mettre à jour les masses associées aux 2-cellules, ainsi la masse m à la 2-cellule c_4^2, sera de[11] :

$$m \leftarrow m - \sum_i m_{p\_out_i} \tag{5.16}$$

et les masses m_i aux 2-cellules c_i^2 seront de :

$$m_i \leftarrow m_i + m_{p\_out_i}. \tag{5.17}$$

Troisième principe

Appliquons maintenant le troisième principe. Nous recommençons au début de la tranche de temps avec les valeurs des coefficients mises à jour par les deux premiers principes. À cause de la friction interne, la vitesse $\vec{v}_a$ de la masse de fluide à une 2-cellule diminue avec le temps. Nous utilisons la formule suivante :

$$\vec{v}_a \leftarrow (1 - k_f\Delta t)\vec{v}_a, \tag{5.18}$$

avec k_f comme étant la constante de friction interne. Il faut que $k_f\Delta t < 1$ et que $k_f \geq 0$, 0 étant une friction nulle.

La vitesse d'une masse de fluide à une 2-cellule est aussi influencée par les vitesses des masses de fluide aux 2-cellules voisines. En débutant encore au début de la tranche de temps, nous utilisons l'heuristique suivante pour calculer la nouvelle vitesse $\vec{v}_a$:

$$\vec{v}_a \leftarrow (1 - k_n\Delta t)\vec{v}_a + k_n\Delta t\,\frac{1}{j}\sum_{i=1}^{j}\vec{v}_{a_i} \tag{5.19}$$

[11] Remarquez encore ici qu'il faut que la sommation soit plus petite que m.

où k_n est la constante de friction entre la 2-cellule c_4^2 et les 2-cellules voisines. Plus la constante est élevée, plus la friction est grande (la valeur est typiquement entre 0 et 1). Dans notre exemple, $j = 3$, mais en général, j est égal au nombre de 2-cellules voisines à la 2-cellule où la vitesse $\vec{v}_a$ est mise à jour.

Quatrième principe

Pour le quatrième principe, nous avons ajouté une heuristique afin que les directions des vitesses $\vec{v}_a$ à chacune des 2-cellules qui sont en contact avec la frontière de l'objet, aient tendance à suivre la direction de la frontière. Toutes les 1-cellules qui ont une seule 2-cellule adjacente font partie de la frontière de l'objet.

Les vitesses $\vec{v}_a$ associées aux 2-cellules qui touchent une frontière, sont corrigées comme suit :

$$\vec{v}_a \leftarrow (1 - k_b\,\Delta t)\vec{v}_a + k_b\,\Delta t\,\vec{v}_b, \tag{5.20}$$

où k_b est une constante réelle qui indique comment la direction de la vitesse aura tendance à suivre la frontière (plus elle est élevée, plus l'influence est grande). Il faut que $0 \leq k_b\Delta t \leq 1$. Dans cette équation, $\vec{v}_b$ est la projection de la vitesse $\vec{v}_a$ sur la frontière. C'est-à-dire :

$$\vec{v}_b = \frac{\vec{r}}{|\vec{r}|}\left(\vec{v}_a \cdot \frac{\vec{r}}{|\vec{r}|}\right), \tag{5.21}$$

où $\vec{r}$ est la différence des positions des 0-cellules adjacentes à la 1-cellule.

5.2.2 Calcul d'aires de triangles

Avant d'aborder la prochaine sous-section (5.2.3) qui explique comment calculer le mouvement des masses de fluide dans son ensemble, nous allons traiter séparément un sous-problème de cette sous-section. Car ce sous-problème justifie à lui seul, l'utilisation de chaînes de sous-chaînes.

Cet exemple relativement simple consiste à calculer l'aire des 2-cellules triangulaires du complexe géométrique *Complex_p*. Nous utilisons la formule qui suit pour calculer l'aire s, avec a, b et c étant les longueurs des côtés :

$$s = \sqrt{w(w-a)(w-b)(w-c)} \tag{5.22}$$

où

$$w = \frac{a+b+c}{2}. \tag{5.23}$$

Nous commençons par calculer les longueurs a, b et c de chacune des 1-cellules géométriques dans la 1-chaîne `chain1_L` sur R, soit :

$$\texttt{chain1\_L} = \texttt{chain1\_r.norm}().$$

Nous avons indiqué comment calculer[12] la 1-chaîne `chain1_r` à la sous-section 5.1.3.

Ensuite, nous évaluons un w (équation 5.23) pour chacun des triangles. Nous emmagasinons donc les résultats dans une 2-chaîne `chain2_w` sur R :

$$\texttt{chain2\_w} = \texttt{Chain\_Real}(0.5) * \texttt{chain1\_L.dim\_inc}(+1, \texttt{plus}, \texttt{false}).$$

Remarquez que la méthode `dim_inc` fait une addition ($\Omega = \Sigma$) des longueurs de chacune des 1-cellules, sans tenir compte de l'orientation relative.

Maintenant, pour évaluer l'équation 5.22, il faut combiner des coefficients sur des 1-cellules (a, b, c), et sur des 2-cellules (w). Nous allons mettre ces coefficients dans des $(2,1)$-chaînes sur R, sans tenir compte de l'orientation relative, afin de les combiner :

$$\texttt{chain21\_L} = \texttt{chain1\_L.in}(+1, \texttt{false});$$

[12] *chain1_r = name_complex.chain0_p.dim_inc(+1,plus,true);* où *name_complex* est *complex_fluid* à la prochaine sous-section.

et

$$\texttt{chain21\_w} = \texttt{chain2\_w.out}(-1, \texttt{false}).$$

Nous évaluons d'abord la partie $(w-a)(w-b)(w-c)$ de l'équation 5.22 pour chacun des triangles :

$$\texttt{chain2\_wL} = (\texttt{chain21\_w} - \texttt{chain21\_L}).\texttt{mult}();$$

et finalement, l'équation 5.22 au complet :

$$\texttt{chain2\_s} = (\texttt{chain2\_w} * \texttt{chain2\_wL}).\texttt{squareroot}();$$

où la méthode `squareroot` retourne une chaîne de réels dont les coefficients sont les racines carrées des réels de la chaîne qui invoque la méthode. Remarquez que l'on aurait pu faire les quatre dernières étapes en une seule, mais l'expression de chaînes n'aurait pas été aussi facile à comprendre.

Nous n'avons pu effectuer l'étape qui calcule la chaîne `chain2_wL` avec seulement la méthode `dim_inc` sur une chaîne de coefficients primitifs. Nous n'affirmons pas que c'est impossible, mais nous ne savons pas comment faire. La difficulté provient du fait que ce n'est pas seulement une opération simple (par exemple : $\Omega = \Sigma,\ \Pi,\ \ldots$) qui est effectuée entre les longueurs des 1-cellules géométriques ; elles doivent aussi être combinées avec w, qui est associé avec les 2-cellules.

5.2.3 Heuristiques pour un fluide avec des chaînes

Nous allons maintenant appliquer les heuristiques de la sous-section 5.2.1 sur tout le complexe, et non pas sur une seule 2-cellule avec ses 2-cellules voisines. Nous utiliserons le cadre de travail avec les chaînes de sous-chaînes.

Notre exemple est en deux dimensions et implique des 2-cellules et des 1-cellules. Nous allons alors utiliser des 2-chaînes, des 1-chaînes, des 1-chaînes de 2-sous-chaînes et des 2-chaînes de 1-sous-chaînes. Remarquez que si nous avions

fait notre exemple en n dimensions, avec $n = 3$ ou $n = 4$ par exemple, nous aurions utilisé des n-chaînes, des $(n-1)$-chaînes, des $(n-1)$-chaînes de n-sous-chaînes et des n-chaînes de $(n-1)$-sous-chaînes.

Rappelons que nous utilisons la géométrie *Complex_p* sur le complexe et que nous avons nommé celui-ci *complex_fluid*.

Définissons les chaînes principales dont nous avons besoin :

- *chain*2_*va* sur R^2 (vitesse $\vec{v}_a$ de la masse de fluide pour chacune des 2-cellules) ;
- *chain*2_*va_norm* sur R^2 (vitesse normalisée $\frac{\vec{v}_a}{|\vec{v}_a|}$ de la masse de fluide pour chacune des 2-cellules) ;
- *chain*2_*m* sur R (masse m de fluide pour chacune des 2-cellules) ;
- *chain*2_*p* sur R (pression p pour chacune des 2-cellules) ;
- *chain*2_*s* sur R (aire s de la 2-cellule géométrique triangulaire pour chacune des 2-cellules) ;
- *chain*1_*L* sur R (longueur de la 1-cellule géométrique pour chacune des 1-cellules) ;
- *chain*1_*deltapi* sur R (valeur 0 lorsque la 1-cellule associée à une seule 2-cellule adjacente, sinon, différence entre les deux pressions aux 2-cellules adjacentes à chacune des 1-cellules) ;
- *chain*1_*vn* sur R^2 (vecteur orthogonal à la géométrie de la 1-cellule, pour chacune des 1-cellules) ;
- *chain*2_*mvout* sur R (masse de fluide m_{v_out} pour chacune des 2-cellules) ;
- *chain*21_*mvouti* sur R (masse de fluide $m_{v\_out_i}$ associée à chacune des paires 2-cellule C_j^2, 1-cellule $C_j^2.adj(i,-1)$) ;
- *chain*12_*mvouti* sur R (chaîne duale de *chain*21_*mvouti*) ;
- *chain*21_*mpouti* sur R (masse de fluide $m_{p\_out_i}$ associée à chacune des paires 2-cellule C_j^2, 1-cellule $C_j^2.adj(i,-1)$) ;
- *chain*12_*mpouti* sur R (chaîne duale de *chain*21_*mpouti*) ;
- *chain*21_*vni* sur R^2 (vecteur $\vec{v}_{n_i}$ associé à chacune des paires 2-cellule C_j^2,

1-cellule $C_j^2.adj(i,-1)$) ;[13]

- *chain*12_*vni* sur R^2 (chaîne duale de *chain*21_*vni*) ;
- *chain*21_*deltapi* sur R (différence de pressions Δp_i associée à chacune des paires 2-cellule C_j^2, 1-cellule $C_j^2.adj(i,-1)$) ;[14]
- *chain*12_*deltapi* sur R (chaîne duale de *chain*21_*vni*).

Nous aurons aussi besoin d'autres chaînes, dont les définitions seront évidentes dans le contexte et qui ne seront utilisées qu'une fois.

Exemple d'une opération entre une 2-chaîne et une $(1,2)$-chaîne

Lorsque nous effectuons une opération entre deux chaînes de sous-chaînes, elles doivent être de la même dimension. Ainsi, par exemple, dans l'équation 5.8, nous multiplions la vitesse normalisée $\frac{\vec{v}_a}{|\vec{v}_a|}$ à chacune des 2-cellules avec les vecteurs $\vec{v}_{n_i}$ correspondants. Les vecteurs $\vec{v}_{n_i}$ sont des informations qui sont conservées dans une 1-chaîne de 2-sous-chaînes, car l'information est différente pour chacune des paires 2-cellule, 1-cellule adjacentes. Nous devons donc multiplier la 2-chaîne `chain2_va_norm` par la 1-chaîne de 2-sous-chaînes `chain12_mvouti`. Il s'agit de faire, comme nous l'avons vu à la sous-section 4.3.6 :

$$\texttt{chain2\_va\_norm.in}(-1,\texttt{false}) * \texttt{chain12\_mvouti};$$

où $\texttt{chain2\_va\_norm} = \texttt{chain2\_va.normalize}()$. Notez que nous utilisons ici la méthode `in` sans tenir compte de l'orientation relative.

Équation 5.7

Appliquons maintenant le premier principe : une masse de fluide à une 2-cellule se déplace dans la direction de $\vec{v}_a$ et une partie est transférée aux 2-cellules voisines.

[13] Cette chaîne n'est utilisée que pour définir la prochaine chaîne (*chain*12_*vni*).

[14] Cette chaîne n'est utilisée que pour définir la prochaine chaîne (*chain*12_*deltapi*) ;

Pour calculer m_{v_out} définie par l'équation 5.7 pour chacune des 2-cellules, il faut d'abord calculer les longueurs moyennes des géométries des 1-cellules pour chacune des 2-cellules. Celles-ci sont calculées dans une 2-chaîne `chain2_la` comme suit :

$$\texttt{chain2\_la} = \texttt{chain1\_L.dim\_inc}(+1, \texttt{average}, \texttt{false});$$

où $chain1\_L$ contient les longueurs des 1-cellules géométriques.

Ainsi, nous pouvons calculer la 2-chaîne `chain2_mvout`, qui contient m_{v_out} pour chacune des 2-cellules :

$$\texttt{chain2\_mvout} = \texttt{chain2\_va.norm}() * \texttt{Chain\_Real}(\texttt{delta\_t})/\texttt{chain2\_la} * \texttt{chain2\_m};$$

où `delta_t` est une constante réelle égale à Δt.

Équation 5.8

Afin de calculer $m_{v\_out_i}$ avec l'équation 5.8, nous avons besoin de 1-chaînes de 2-sous-chaînes.

Il faut d'abord calculer la 1-chaîne `chain1_vn` :

$$\texttt{chain1\_vn} = \texttt{Chain\_Real2}(-\texttt{chain1\_r\_norm.y}(), \texttt{chain1\_r\_norm.x}());$$

ou

$$\texttt{chain1\_vn} = \texttt{Chain\_Real2}(\texttt{chain1\_r\_norm.y}(), -\texttt{chain1\_r\_norm.x}());$$

selon l'orientation des 2-cellules[15]. La 1-chaîne `chain1_r_norm` sur R^2 est égale à `chain1_r.normalize()`. Les méthodes de chaînes sur R^2 (classe *Chain_Real2*) `x` et `y`

[15]Lorsque la mauvaise expression de chaînes est choisie, cela aura pour conséquence à la prochaine étape, que les vecteurs $\vec{v}_{n_i}$ pointeront vers l'intérieur plutôt que vers l'extérieur des 2-cellules.

retournent une chaîne sur R qui correspond respectivement aux composantes x et y de la chaîne sur R^2 qui invoque la méthode. Une façon de définir une instance de la classe *Chain_Real2* (que nous n'avions pas définie jusqu'à présent) est de se servir de $Chain\_Real2(chainp\_x, chainp\_y)$. Cette méthode prend en paramètre, deux p-chaînes sur R (*chainp_x* et *chainp_y* de la classe *Chain_Real*) qui correspondent aux composantes x et y des coefficients R^2 de la p-chaîne à définir.

Nous calculons ensuite la 1-chaîne de 2-sous-chaînes `chain12_vni` sur R^2 qui contient les vecteurs $\vec{v}_{n_i}$. Nous devons avoir les 2-cellules du complexe qui sont orientées de façon cohérente, comme à la sous-section 5.1.2. Nous calculons `chain12_vni` en tenant compte de l'orientation relative, afin que les vecteurs $\vec{v}_{n_i}$ pointent vers l'extérieur des 2-cellules :

$$\texttt{chain12\_vni} = \texttt{chain1\_vn.out}(+1, \texttt{true});$$

notez que ce calcul ne sera effectué qu'une fois lors de l'initialisation.

Nous définissons la 1-chaîne de 2-sous-chaînes sur R^2 suivante :

$$\texttt{chain12\_va\_norm} = \texttt{chain2\_va\_norm.in}(-1, \texttt{false}).$$

Nous avons maintenant ce qu'il faut pour calculer l'équation 5.8 avec des 1-chaînes de 2-sous-chaînes :

$$\texttt{chain12\_mvouti} = (\texttt{chain12\_va\_norm} * \texttt{chain12\_vni} * \texttt{chain12\_mvout}) \\ .\texttt{max}(\texttt{chain12\_zeros});$$

la méthode `max(r)` de la classe *Real* retourne le coefficient maximal entre le réel qui invoque la méthode et le réel `r` en paramètre. (La méthode étant définie sur les réels, l'est donc aussi sur une chaîne de réels et aussi sur une chaîne de sous-chaînes de réels. Voir la section 4.3.2.) La chaîne `chain12_zeros` est égale à `Chain2_Real(Chain_Real(0.0))` (tous les coefficients sont égaux à 0).

Après avoir calculé cette 1-chaîne de 2-sous-chaînes `chain12_mvouti`, il faut traiter les cas où une 2-cellule est en contact avec la frontière. Nous allons mettre

à 0 tous les coefficients de `chain12_mvouti` associés à chacune des paires 1-cellule, 2-cellule adjacentes, lorsque la 1-cellule fait partie de la frontière. Pour ce faire, nous allons utiliser le mécanisme de sélection :

`chain12_mvouti`[`vect1cells_one2cell`] = `chain12_zeros`;

avec `vect1cells_one2cell` étant une sélection qui donne les 1-cellules du complexe ayant une seule 2-cellule adjacente. Notez qu'il est facile de construire le vecteur de 1-cellules `vect1cells_one2cell`. Nous pouvons faire une boucle sur toutes les 1-cellules C_i^1 du complexe, et ajouter au vecteur, celles que $C_i^1.nb\_adj(+1) = 1$.

Équation 5.9

Nous avons à la figure 5.8 un complexe simple formé de quatre 2-cellules qui illustre des quantités de masse qui quittent les 2-cellules C_j^2 par les 1-cellules $C_j^2.adj(i,-1)$ (en raison des vitesses moyennes), soient $m_{v\_out_{j,i}}$. Ici les quantités $m_{v\_out_{j,i}}$ correspondent aux $m_{v\_out_i}$ qui proviennent de chacune des 2-cellules C_j^2 du complexe. La figure représente la $(1,2)$-chaîne *chain12_mvouti* et par le fait même sa chaîne duale. Remarquez que nous devrons transférer ces quantités pour chacune des 2-cellules C_j^2 vers les 2-cellules voisines qui partagent la 1-cellule $C_j^2.adj(i,-1)$. Remarquez aussi que les 2-cellules voisines n'existent pas nécessairement.

Nous introduisons la méthode *sym* valide dans les classes *Chain2_type* qui nous sera très utile par la suite :

- *sym()*. Méthode valide sur une $(p-1)$-chaîne de p-sous-chaînes. Pour chacun des coefficients g associé à la paire de cellules $(p-1)$-cellule c_i^{p-1}, p-cellule c_j^p adjacentes, la valeur du coefficient g est interchangée avec la valeur du coefficient associé à la paire de cellules $(p-1)$-cellule c_i^{p-1}, p-cellule c_k^p adjacentes, et c_k^p est différent de c_j^p. Si c_k^p n'existe pas, alors le coefficient aura la valeur $g - g$.

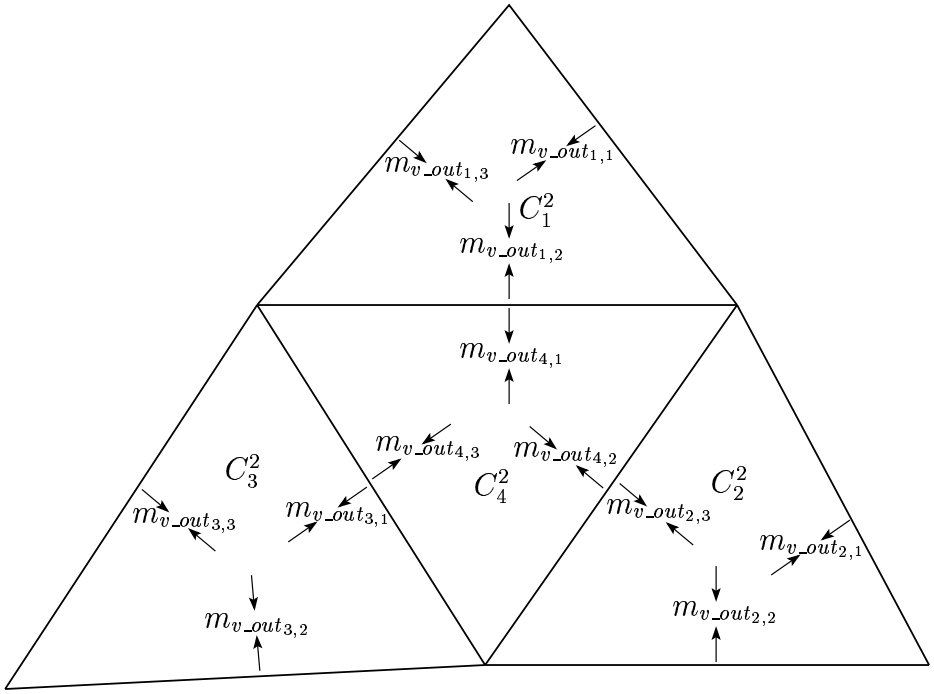

FIG. 5.8 – Des quantités de masses sont transférées des 2-cellules.

La figure 5.9 illustre l'application de la méthode *sym* sur une 1-chaîne de 2-sous-chaînes.

On implante facilement la méthode *sym* qui est appliquée sur la $(p-1)$-chaîne de p-sous-chaîne *chainqp_a*[16] et qui retourne la $(p-1)$-chaîne de p-sous-chaîne *chainqp_b* :

$chainq\_temp = chainqp\_a.sum();$
$chainqp\_b = chainq\_temp.out(+1, false)$;
$chainqp\_b = chainqp\_b - chainqp\_a$;

Les coefficients des chaînes *chainq_temp* et *chainqp_b* sont de même type que les coefficients de la chaîne *chainqp_a*. Nous supposons que la méthode *sum* et

[16]La chaîne est nommée *chainqp_a*, afin d'avoir un nom de chaîne plus simple, nous fixons que $q = p - 1$.

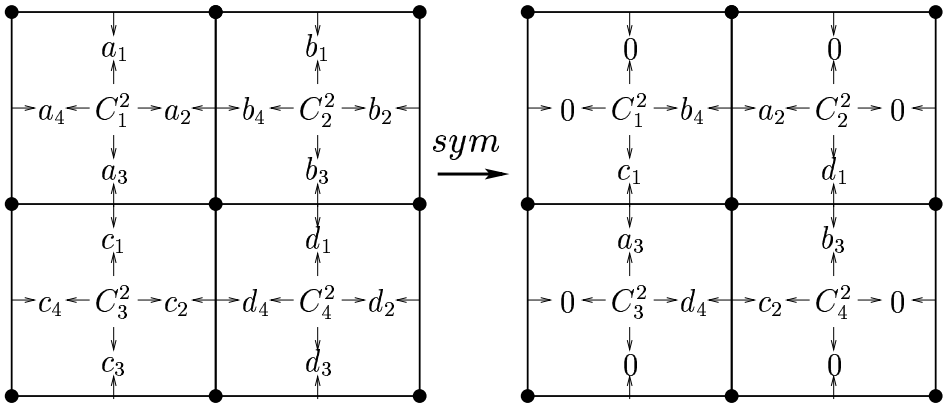

FIG. 5.9 – Application de la méthode *sym* sur une 1-chaîne de 2-sous-chaînes.

l'opération binaire "$-$" sont définies sur la classe *Chain2_Type*.

Maintenant, concentrons-nous sur le dénominateur de l'équation 5.9. Il nous faut ajouter à m_i la valeur de $m_{v\_out_i}$. Cela veut dire qu'il faut ajouter à la masse à chacune des 2-cellules, les masses provenant des 2-cellules voisines. Dans notre exemple illustré à la figure 5.8, il faudra, entre autres, ajouter à la masse m associée à la cellule C_2^2, la masse $m_{v\_out_{4,2}}$. Nous remarquons que ces quantités ne sont pas associées à la même 2-cellule. C'est maintenant que la méthode *sym* nous est très utile. En effet, en appliquant cette méthode à la chaîne *chain12_mvouti*, on obtient la chaîne *chain12_mvouti_sym*, c'est-à-dire :

`chain12_mvouti_sym` = `chain12_mvouti.sym()`.

Cette 1-chaîne de 2-sous-chaînes est illustrée à la figure 5.10 (la méthode *sym* a été appliquée sur la chaîne *chain12_mvouti* à la figure 5.8). Maintenant, nous avons d'associé à chacune des 2-cellules, les masses provenant des 2-cellules adjacentes. Il est maintenant simple de calculer le dénominateur de l'équation 5.9 pour toute les 2-cellules, soit :

`chain2_m` + `chain12_mvouti_sym.transpose().sum()`.

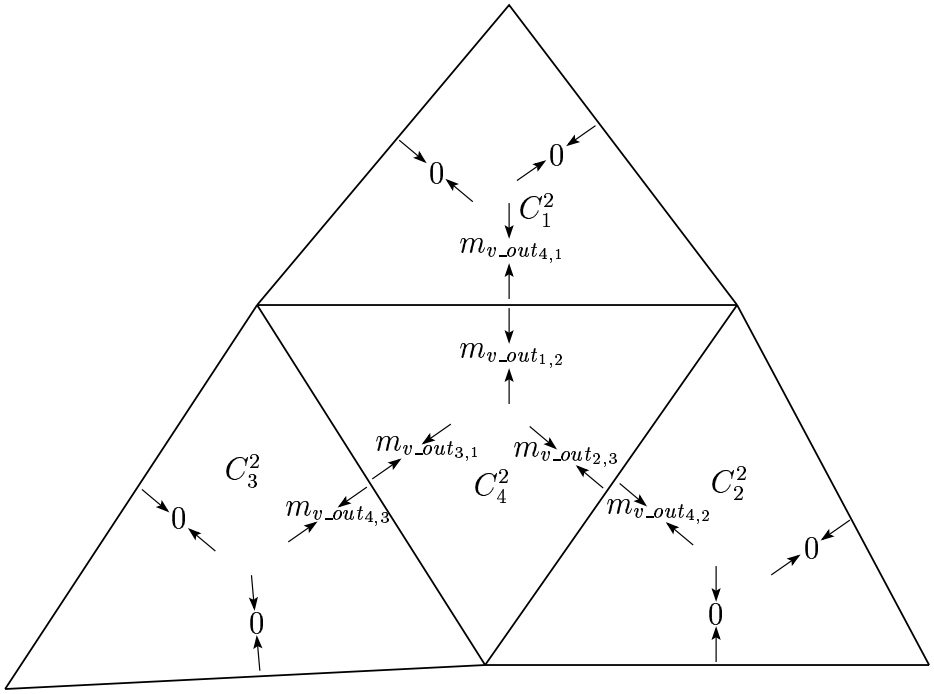

FIG. 5.10 – Résultat de l'application de la méthode *sym* sur la chaîne *chain*12_*mvouti*.

En appliquant la même logique, nous pouvons calculer complètement l'équation 5.9, soit :

$$\texttt{chain12\_va\_sym} = \texttt{chain2\_va.in}(+1, \texttt{false}).\texttt{sym}();$$

et

$$\texttt{chain2\_va} = (\texttt{chain2\_m} * \texttt{chain2\_va} + (\texttt{chain12\_mvouti\_sym} * \texttt{chain12\_va\_sym}).\texttt{transpose}().\texttt{sum}()) / (\texttt{chain2\_m} + \texttt{chain12\_mvouti\_sym.transpose}().\texttt{sum}()).$$

Équation 5.10 et 5.11

Avec :

$$\texttt{chain21\_mvouti} = \texttt{chain12\_mvouti.transpose}();$$

les équations 5.10 et 5.11 indiquant les mises à jour des masses pour le premier principe sont calculées dans tout le complexe comme suit :

```
chain2_m = chain2_m − chain21_mvouti.sum() +
                          chain12_mvouti_sym.transpose().sum().
```

Le calcul `chain21_mvouti.sum()` provient de l'équation 5.10 et le calcul `chain12_mvouti_sym.transpose().sum()` provient de l'équation 5.11. Ici, pour chacune des 2-cellules, nous soustrayons la masse qui sort de la 2-cellule, et nous ajoutons les masses qui proviennent des 2-cellules voisines. Remarquez le "`sym`" pour ce qui provient des 2-cellules voisines.

Équation 5.12

Afin de calculer l'équation 5.12 qui définit la quantité de masse qui quitte une 2-cellule vers ses 2-cellules voisines (en raison d'une différence de pression), il faut d'abord calculer les différences de pression Δp_i dans la 1-chaîne `chain1_deltapi`, comme suit :

```
chain2_p = chain2_m / chain2_s;
chain1_deltapi = chain2_p.dim_inc(−1, plus, true);
```

avec `chain2_s` donnant l'aire de chacune des 2-cellules géométriques (triangles). La méthode `dim_inc` tient compte de l'orientation relative afin que l'opération Ω (indiquée par le deuxième paramètre de `dim_inc`, soit `plus`) qui est l'addition, soustrait l'une de l'autre les deux pressions p aux deux 2-cellules qui partagent chacune des 1-cellules.

Nous supposons la différence de pression, avec l'extérieur et avec la frontière (1-cellules) d'un objet, égale à 0. Nous utilisons le mécanisme de sélection afin de mettre à zéro les coefficients de la chaîne `chain1_deltapi` dont les 1-cellules ont une seule 2-cellule adjacente, comme suit :

```
chain1_deltapi[vect1cells_one2cell] = Chain_Real(0.0);
```

avec la sélection `vect1cells_one2cell` contenant les 1-cellules du complexe ayant une seule 2-cellule adjacente.

Nous travaillons avec des 1-chaînes de 2-sous-chaînes dans le calcul de l'équation 5.12, alors nous intégrons la 1-chaîne `chain1_deltapi` dans une 1-chaîne de 2-sous-chaînes :

$$\texttt{chain12\_deltapi} = \texttt{chain1\_deltapi.out}(+1, \texttt{true}).$$

Nous avons ainsi emmagasiné, pour chacune des 2-cellules, les Δp_i la concernant dans `chain12_deltapi`[17]. Nous tenons compte de l'orientation relative afin qu'une différence de pression positive dans `chain12_deltapi` indique qu'à la paire de 1-cellule, 2-cellule concernée, la masse à la 2-cellule quittera par la 1-cellule.

Ainsi, nous calculons l'équation 5.12 comme suit :

$$\begin{aligned}\texttt{chain12\_mpouti} = (&\texttt{chain12\_deltapi/chain2\_p.in}(-1, \texttt{false}) * \\ &\texttt{Chain2\_Real}(\texttt{Chain\_Real}(\texttt{delta\_t/kv})) * \texttt{chain2\_m.in}(-1, \texttt{false})) \\ &\texttt{.max}(\texttt{chain12\_zeros});\end{aligned}$$

où `kv` est une constante réelle égale à k_v et où `delta_t` est une constante réelle égale à Δt.

Équation 5.13

L'équation 5.13 est calculée comme suit :

$$\begin{aligned}\texttt{chain12\_vpi} = &\texttt{chain2\_va.in}(-1, \texttt{false}) + \\ &\texttt{Chain2\_Real}(\texttt{Chain\_Real}(\texttt{kp})) * \texttt{chain12\_vni};\end{aligned}$$

où `kp` est une constante réelle égale à k_p.

[17] Dans la chaîne duale, il est plus facile de se représenter les Δp_i.

Équation 5.14

Notez la similarité entre l'équation 5.9 et l'équation 5.14. Comme ce fut le cas dans l'équation 5.9, nous utilisons la méthode `sym` afin de combiner des valeurs à chacune des 2-cellules avec ses 2-cellules voisines :

```
chain2_va = (chain2_m * chain2_va+
    (chain12_mpouti.sym() * chain12_vpi.sym()).transpose().sum()) /
                (chain2_m + chain12_mpouti.sym().transpose().sum()).
```

Équation 5.15

Nous calculons d'abord le terme $\sum_i m_{p\_out_i}$ de l'équation 5.15 :

```
chain2_som2 = chain12_mpouti.transpose().sum();
```

dans une 2-chaîne sur R. Notez ici que nous n'avons pas à appliquer la méthode `sym` car dans l'équation 5.15, nous calculons les $\vec{v}_a$ et non les $\vec{v}_{a_i}$ à chacune des 2-cellules.

Nous calculons aussi le terme $\sum_i m_{p\_out_i}\vec{v}_{p_i}$:

```
chain2_som1 = (chain12_mpouti * chain12_vpi).transpose().sum();
```

dans une 2-chaîne sur R^2.

Et enfin, l'équation 5.15 :

```
chain2_va = (chain2_m * chain2_va + chain2_som1) /
                                        (chain2_m + chain2_som2).
```

Équations 5.16 et 5.17

Notez la similarité des équations 5.16 et 5.17 et des équations 5.10 et 5.11. Les équations 5.16 et 5.17 indiquant les mises à jour des masses pour le deuxième

principe sont calculées dans tout le complexe comme suit :

$$\texttt{chain2\_m} = \texttt{chain2\_m} - \texttt{chain21\_mpouti.sum}()+ \\ \texttt{chain12\_mpouti.sym}().\texttt{transpose}().\texttt{sum}().$$

Le calcul `chain21_mpouti.sum()` provient de l'équation 5.16 et le calcul `chain12_mpouti.sym().transpose().sum()` provient de l'équation 5.17.

Équation 5.18

Nous abordons maintenant le troisième principe, qui concerne la friction. L'équation 5.18, qui est pour la friction interne dans une 2-cellule, est facilement calculée avec :

$$\texttt{chain2\_va} = \texttt{Chain\_Real}(1.0 - \texttt{kf} * \texttt{delta\_t}) * \texttt{chain2\_va};$$

où `kf` est une constante réelle égale à k_f.

Équation 5.19

Il nous reste à traiter la friction d'une masse de fluide avec les masses aux 2-cellules voisines. Nous utilisons la méthode `dim_inc` sur une 2-chaîne avec δp égal à 0^- (précisé avec l'entier -100 comme premier paramètre). Cette relation d'adjacence indique que nous utilisons les coefficients des 2-cellules voisines à une 2-cellule. L'opération $op =$ `average`, précise que Ω (section 2.5) est égale à $\frac{1}{m_i}\sum_{k=1}^{m_i}$, et ce, sans tenir compte de l'orientation relative. Ainsi, pour l'équation 5.19, nous avons :

$$\texttt{chain2\_va} = \texttt{Chain\_Real}(1.0 - \texttt{kn} * \texttt{delta\_t}) * \texttt{chain2\_va} + \\ \texttt{Chain\_Real}(\texttt{kn} * \texttt{delta\_t}) * \texttt{chain2\_va.dim\_inc}(-100, \texttt{average}, \texttt{false});$$

où `kn` est une constante réelle égale à k_n.

Équations 5.20 et 5.21

Pour le quatrième principe, nous n'avons pas réussi, avec les outils vus dans cette thèse, à exprimer cela avec des expressions de chaînes. Il est par contre, relativement facile d'appliquer les équations 5.20 et 5.21 avec les méthodes auxquelles nous avons accès avec l'interface de programmation et avec une boucle sur toutes les 1-cellules faisant partie de la frontière de l'objet[18].

Nous avons expérimenté avec des outils, que nous allons aborder très sommairement dans la conclusion de la thèse, nous permettant d'exprimer ces formules avec une expression de chaînes.

Cela complète la partie qui simule des masses de fluide se déplaçant simultanément de 2-cellules à d'autres.

5.2.4 Modélisation des particules dans le fluide

Afin de visualiser le mouvement des masses de fluide, nous avons utilisé des particules de masse nulle [17]. À chacune des particules est associée l'information suivante :

- la vitesse $\vec{v_p}$;
- la position $\vec{p_p}$;
- la 2-cellule géométrique C_i^2 à l'intérieur de laquelle se situe la particule.

Des valeurs aléatoires sont affectées aux positions[19] $\vec{p_p}$ au début de la simulation $(t = 0)$.

La vitesse de chacune des particules sera calculée d'après la position de la particule dans un champ de vitesse du fluide sur le complexe géométrique. Nous avons une discontinuité de la vitesse de la masse de fluide à chacune des 1-cellules car

[18] Le vecteur de 1-cellules `vect1cells_one2cell` que nous avons vu ci-dessus, contient les 1-cellules faisant partie de la frontière.

[19] Ces positions devront cependant être à l'intérieur de l'une des 2-cellules géométriques.

nous avons une vitesse $\vec{v}_a$ qui peut être différente d'une 2-cellule à l'autre. Afin que les vitesses soient continues dans l'espace en deux dimensions, nous commençons par calculer une 0-chaîne sur R^2 de vitesses $chain0\_va$ qui contient la moyenne des vitesses $\vec{v}_a$ aux 2-cellules adjacentes[20]. Ensuite la position $\vec{p}_p$ de chacune des particules est convertie en coordonnées barycentriques en fonction des positions des trois 0-cellules adjacentes à la 2-cellule C_i^2. C'est-à-dire que nous trouvons les réels u, v et w, tel que :

$$\vec{p}_p = u\vec{p}_1 + v\vec{p}_2 + w\vec{p}_3,$$

avec

$$u + v + w = 1 \text{ et } 0 \leq u, v, w \leq 1,$$

où $\vec{p}_1$, $\vec{p}_2$ et $\vec{p}_3$ sont respectivement les positions des 0-cellules $C_i^2.adj(1,-2)$, $C_i^2.adj(2,-2)$ et $C_i^2.adj(3,-2)$.

Ensuite, nous avons la vitesse au temps t de la particule comme étant :

$$\vec{v}_p = u\vec{v}_1 + v\vec{v}_2 + w\vec{v}_3,$$

où $\vec{v}_1$, $\vec{v}_2$ et $\vec{v}_3$ sont respectivement les vitesses dans la 0-chaîne $chain0\_va$ aux 0-cellules $C_i^2.adj(1,-2)$, $C_i^2.adj(2,-2)$ et $C_i^2.adj(3,-2)$.

Nous pouvons alors mettre à jour la position $\vec{p}_p$ de chacune des particules au temps $t + \Delta t$, comme étant :

$$\vec{p}_p + \vec{v}_p \Delta t.$$

Cette équation a cette forme très simple car nous le rappelons, on suppose que les particules ont une masse nulle.

Il peut arriver que des particules, en suivant le champ de vitesse, ne se retrouvent à l'intérieur d'aucune 2-cellule géométrique. Nous devons régler ce

[20] Ceci est effectué avec $chain0\_va = chain2\_va.dim\_inc(-2, average, false)$.

problème. Pour simplifier celui-ci, nous avons construit la frontière du complexe géométrique *complex_fluid* alignée sur les axes canoniques, et la frontière dans son ensemble est de forme rectangulaire. Nous notons les valeurs X_{min}, X_{max}, Y_{min} et Y_{max} qui délimitent la frontière du complexe. Alors si la composante x est plus petite que la valeur X_{min}, nous affectons la valeur X_{min} à celle-ci. Si la composante x est plus grande que X_{max}, alors nous affectons la valeur X_{max} à la composante x. Nous faisons de même pour la composante y.

5.2.5 Résultats

Le but de cette section était de montrer que l'on peut effectuer des simulations basées sur des règles relativement complexes avec l'aide de chaînes de sous-chaînes, et ce, tout en restant haut niveau. Il faut aussi mentionner que beaucoup de sous-problèmes ont été résolus avec les chaînes de sous-chaînes dans cette section. Cela inclut, par exemple, le calcul de tous les vecteurs normaux pointant vers l'extérieur de la 2-cellule, ou encore, l'utilisation de la méthode *sym* pour accéder aux coefficients associés avec les 2-cellules voisines pour chacune des 2-cellules.

Comme nous l'avons déjà mentionné, nous avons obtenu des résultats très satisfaisants. Nous avons trouvé certaines valeurs des différentes constantes qui donnaient des résultats que nous supposons réalistes. Nous avons constaté, pour certaines valeurs des différentes constantes, que la simulation donnait des résultats non réalistes, mais qui pourraient être intéressants pour explorer différents effets spéciaux. Nous n'avons pas fait, par contre, de recherche dans ce sens.

Ajoutons aussi que nous avons beaucoup de constantes, cela a comme avantage qu'il y a plus de diversité dans les résultats, c'est-à-dire, plus d'effets possibles. Par contre, cela peut prendre plus de temps pour ajuster les différentes constantes. Une analyse pourrait être effectuée afin de fusionner différentes constantes en une seule, par exemple, celles qui seraient en relation avec la viscosité du fluide.

Nous avons utilisé un complexe de 2-cellules triangulaires, car il est plus facile

de calculer un champ de vitesse continue avec des triangles (ou avec des rectangles) que pour des formes arbitraires. Nous avons préféré utiliser les triangles étant donnée la flexibilité qu'ils permettent afin de découper l'espace. Nous croyons que les heuristiques que nous avons employées peuvent être utilisées pour des rectangles, et probablement pour d'autres formes, et ce, avec des modifications mineures aux heuristiques.

Mentionnons, que le rendu des particules est réduit à sa plus simple expression, c'est-à-dire que seulement un point est affiché par particule. Nous pourrions alternativement afficher un petit "nuage" autour de la position de chacune des particules.

Il semble clair que la simulation du fluide avec des chaînes de sous-chaînes peut être appliquée à d'autre dimensions, en particulier pour des dimensions plus grandes que trois.

Chapitre 6

Conclusion

Dans cette thèse, nous avons décrit un cadre de travail basé sur les chaînes pour la spécification et la manipulation de modèles de système. Le cadre de travail est plus général que les autres approches, et nous avons montré des exemples illustrant son utilité et sa simplicité. Un problème qui, à la base, est difficile, demeurera difficile, mais ce cadre de travail permet d'exprimer les sous-problèmes beaucoup plus facilement et sans ambiguïté : nous avons alors une vision globale du problème nous permettant de négliger les détails de bas niveau.

La généralité de notre approche provient au moins de trois sources : nous distinguons le complexe géométrique du complexe topologique, nous utilisons une version généralisée (*dim_inc*) des opérateurs de chaînes *boundary* et *coboundary*, et nous permettons une très grande généralité pour les coefficients, en particulier des cellules et surtout, des sous-chaînes. Nous avons aussi démontré comment il est possible de traiter le problème de l'orientation d'une cellule, pour des cellules de dimension arbitraire, en utilisant seulement les orientations relatives.

En plus de son approche générale, notre cadre de travail permet d'unifier la façon de décrire une très grande variété d'applications. Il permet aussi d'unifier les aspects géométriques et non géométriques d'un système.

6.1 Cadre de travail de base et exemples

Le but était d'avoir des bases solides sur lesquelles nous pouvions ensuite élaborer. Nous ne voulions pas de contraintes sur les types de coefficients d'une chaîne et nous voulions aussi des relations d'adjacences générales. Ensuite, en ayant traité efficacement le problème de l'orientation, nous avons ainsi défini la méthode *dim_inc* qui permet d'avoir des relations entre des chaînes de différentes dimensions, avec un choix d'opérations Ω.

L'exemple du ressort a montré que l'on pouvait décrire un système relativement complexe très simplement. Le modèle est décrit clairement avec seulement une dizaine d'expressions de chaînes et la simulation peut s'appliquer sur un réseau de masses-ressorts arbitraire. Nous avons vu aussi qu'après avoir ajouté des 2-cellules, cela nous a permis de faire une simulation d'un drapeau flottant au vent en ajoutant seulement quelques expressions de chaînes. En ayant les mêmes conventions, il est facile de bâtir sur un modèle existant. Nous avons aussi montré que nous pouvions utiliser le modèle de chaîne pour un exemple totalement différent, c'est-à-dire la subdivision de surface de Catmull-Clark. Dans ces deux derniers exemples nous avons montré qu'il est utile que la relation entre des chaînes soit autre chose que $\delta p = +1$ et $\delta p = -1$, et que Ω pouvait être différent de Σ.

Nous avions considéré au départ les relations d'adjacences 0^+ et 0^- dans un but de complétude. Nous avons quand même trouvé un cas où cela nous a été utile : dans l'expression de chaînes qui considère la friction d'une masse de fluide à une 2-cellule avec les masses aux 2-cellules voisines.

6.2 Extensions au cadre de travail et exemples

Comme nous l'avons vu, il y a des coefficients qui peuvent être associés à deux cellules, de là, les chaînes de sous-chaînes. En permettant des sous-chaînes pour les coefficients d'une chaîne, nous avons développé un formalisme pour manipuler

celle-ci. Ainsi seulement quatre[1] opérations de base sont nécessaires (*transpose*, *out*, *in*, *dim_inc*) en plus de l'extension des méthodes et opérations binaires sur des chaînes primitives à des chaînes de sous-chaînes.

Nous avons l'exemple des moments de force illustrant les possibilités des chaînes de sous-chaînes. Nous avons résolu ce problème, tout de même complexe, en considérant que simplement pour un cube, il y a douze forces provenant des moments de force à être cumulées à chacun des sommets. Tout cela est fait avec une dizaine d'expressions de chaînes qui traitent efficacement l'accumulation de toutes ces forces. Il faut noter aussi l'utilisation appropriée de l'orientation relative.

Les chaînes de sous-chaînes ont aussi permis de résoudre des problèmes très difficiles à exprimer seulement avec des chaînes primitives. Nous avons procédé en convertissant les chaînes primitives en chaînes de sous-chaînes, pour ensuite faire les traitements voulus, et finalement, en convertissant les résultats dans des chaînes primitives. L'exemple du calcul de l'aire des cellules triangulaires dans le complexe géométrique est un cas simple qui démontre l'utilisation de chaînes de sous-chaînes pour effectuer les calculs.

C'est ce que nous avons fait aussi dans l'exemple des fluides. Cet exemple est composé de quinze heuristiques relativement complexes. Nous avons calculé celles-ci avec une trentaine d'expressions de chaînes. Il faut considérer que des sous-problèmes ont dû être résolus avant même d'appliquer ces heuristiques, par exemple, d'avoir les vecteurs normaux externes à chacune des 2-cellules. Mais les résolutions de ces sous-problèmes, peuvent ensuite être utilisées telles quelles, pour des applications qui peuvent être très différentes de la simulation de fluide, car nous utilisons toujours les mêmes conventions. Avec de très légères modifications, plusieurs résolutions de ces sous-problèmes pourraient aussi être utilisées pour d'autres dimensions. Mentionnons que le traitement du fluide à la frontière est fait

[1]Nous pouvons même dire trois en considérant que la méthode *in* est définie avec la méthode *transpose* et la méthode *out*.

de façon simple avec le mécanisme de sélection.

Nous nous rendons compte qu'il y a un certain temps d'apprentissage pour utiliser les chaînes et les chaînes de sous-chaînes, mais une fois cette étape franchie, il est très intéressant de constater toutes les possibilités qu'offre le cadre de travail. Il n'y a pas de limites sur les dimensions des cellules rattachées aux problèmes. Nous constatons aussi que beaucoup de problèmes peuvent être résolus plus facilement, de façon concise et surtout, de façon élégante.

6.3 Travaux futurs

Nous voudrions exploiter les possibilités qu'offre le cadre de travail considérant que nous n'avons pas de limites sur la dimension des cellules dans un complexe. Nous voudrions commencer par la simulation des ressorts dans un espace à quatre dimensions. Nous pensons aussi pouvoir faire la simulation des fluides en trois et même quatre dimensions (par exemple, il pourrait y avoir un courant d'air à l'intérieur de la bouteille de Klein).

Nous voulons étudier le sens que pourrait avoir des chaînes de sous-chaînes de sous-chaînes... c'est-à-dire un complexe avec des coefficients associés à plus de deux cellules (nous pourrions aussi exprimer cela comme une $(p, q, r, ...)$-chaîne). Il y a peut-être des problèmes qui pourraient être exprimés avec ces chaînes. Bien que notre implantation ne le permette pas pour l'instant, nous pensons que les opérations de base peuvent facilement être adaptées à ces chaînes.

Concernant la mise à jour du maillage dans la dernière partie de l'exemple de Catmull-Clark, nous avons utilisé, entre autres, deux 1-chaînes de 1-cellules. Nous pensons pouvoir améliorer cette partie avec une chaîne de sous-chaînes. Les deux 1-chaînes seraient remplacées par une $(1, 0)$-chaîne. Car en fait, la première 1-chaîne est associée à la première 0-cellule adjacente de chacune des 1-cellules et la deuxième 1-chaîne est associée à la deuxième 0-cellule.

Nous voulons formaliser des outils expérimentaux dérivés du mécanisme de sélection. Nous avons défini la sélection comme étant un vecteur de p-cellules, ce qui est très proche d'une chaîne de p-cellules. La suite logique est d'utiliser les méthodes définies sur une cellule (*nb_adj* et *adj*), et d'utiliser celles-ci sur une chaîne de cellules. Il y a des cas particuliers qui peuvent être résolus avec ce principe (nous avons mentionné un cas dans la thèse concernant le quatrième principe lors de la modélisation de fluide).

Lorsque la résolution d'un problème est formulée avec des chaînes, nous avons une forme qui, nous pensons, pourrait être calculé en parallèle. Par exemple, lorsque nous faisons la racine carrée d'une chaîne, l'ordre des racines carrées des coefficients n'a aucune importance. Voilà pourquoi nous pensons qu'une implantation de l'interface pourrait être faite dans ce sens.

Il serait intéressant de développer des outils facilitant le changement de topologie d'un complexe. En particulier, un concept de cellule hiérarchique pourrait être élaboré. Nous voulons aussi tenter d'assouplir les conditions imposées sur les cellules d'un complexe. À quelques endroits dans la thèse, nous n'avons pas réussi à exprimer un calcul avec une expression de chaînes (par exemple pour les vecteurs normaux dans la simulation des moments de force). Nous voudrions caractériser plus précisément ce qui peut être calculé et ce qui ne peut l'être.

Il serait évidemment intéressant de définir d'autres classes de complexe géométrique avec, entre autres, une méthode permettant l'intersection d'un axe avec celui-ci. Nous voulons aussi développer de nouvelles applications ou adapter des applications déjà existantes en utilisant notre cadre de travail.

Bibliographie

[1] Palmer, R. S. et Shapiro, V. Chain models of physical behavior for engineering analysis and design. *Research in Engineering Design* (5), 161-184, 1993.

[2] Palmer, R. S. Chain models and finite element analysis : an executable *Chains* formulation of plane stress. *Computer Aided Geometric Design* (12), 733-770, 1995.

[3] Provot, X. Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Graphics Interface '95,* 147-154, 1995.

[4] Catmull, E. et Clark, J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design* (10), No. 6, 350-355, 1978.

[5] Requicha, A. A. G. Mathematical models of rigid solid objects. Technical Memo TM-28, University of Rochester, 1977.

[6] Hocking, J. G. et Young, G. S. Topology. Addison-Wesley, 1961.

[7] Tonti, E. The reason for analogies between physical theories. *Applied Mathematical Modeling* (1), No. 1, Polytec di Milano, 37-50, June 1976.

[8] Branin, F. H. Jr. The network concept as a unifying principle in engineering and the physical sciences. In : Problem Analysis in Science and Engineering, F. H. Branin Jr., editor, 41-101. Academic Press, 1977.

[9] Bamberg, P., et Sternberg, S. A Course in Mathematics for Students of Physics, Volumes 1 et 2. Cambridge University Press, 1990.

[10] Chard, J. A. et Shapiro, V. A Multivector Data Structure for Differential Forms and Equations. À paraître dans *IMACS Journal, Mathematics and Computers in Simulation.*

[11] Bernardini, F., Ferrucci, V. et Paoluzzi, A. Working with dimension-independent polyhedra. Report RAP.07.91, Universitá Degli Studi di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica, 1991.

[12] Rossignac, J. R., et O'Connor, M. A. SGC : a dimension-independent model for pointsets with internal structures et incomplete boundaries. In Geometric Modeling for Product Engineering, Wozny, M. *et al*, editors. IFIP, 1990.

[13] Elter, H. et Lienhardt, P. Extension of the notion of map for the representation of the topology of cellular complexes. *Proc. 4th Canadian Conference on Computational Geometry,* 65-70, 1992.

[14] Weiler, K. The radial edge structure : a topological representation for non-manifold geometric boundary modeling. In Geometric Modeling for CAD Applications, Wozny, M. *et al*, editeurs. IFIP, 1988.

[15] MacCracken, R. et Joy, K. I. Free-form deformations with lattices of arbitrary topology. *Computer Graphics, Annual Conference Series,* 181-188, 1996.

[16] Foster, N. et Metaxas, D. Realistic animation of liquids. *Graphics Interface '96,* 204-212, 1996.

[17] Foster, N. et Metaxas, D. Modeling the motion of a hot, turbulent gas. *Computer Graphics, Annual Conference Series,* 181-188, 1997.

[18] Essa, I. A., Sclaroff, S., et Pentland, A. A unified approach for physical and geometric modeling for graphics and animation. *Eurographics '92*, A. Kilgour et L. Kjelldahl, éditeurs invités. Blackwell Publishers, Eurographics Association, 1992.

[19] Hoppe, H. Progressive meshes. *Computer Graphics, Annual Conference Series,* 99-108, 1996.

[20] Paoluzzi, A. Pascucci, V. et Vicentino, M. Geometric programming : A programming approach to geometric design. *ACM Transactions on Graphics* (14), No. 3, 266-306, 1995.

[21] McKenna, M. et Zelter, D. Dynamic simulation of autonomous legged locomotion. *Computer Graphics, Annual Conference Series,* 29-38, 1990.

[22] Donikian, S. et Cozot, R. General animation and simulation platform. *Computer Animation and Simulation '95.* D. Terzopoulos et D. Thalmann, éditeurs. Springer-Verlag, 197-209, 1995.

[23] Zeleznik, R. C. *et al.* An object-oriented framework for the integration of interactive animation techniques. *Computer Graphics, Annual Conference Series,* 105-111, 1991.

[24] Barzel, R. et Barr, A. H. A modeling system based on dynamic constraints. *Computer Graphics, Annual Conference Series,* 179-187, 1992.

[25] Lienhardt, P. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer-Aided Design* (23), No. 1, 59-82, 1991.

[26] Berge, C. Graphs and Hypergraphs. Elsevier, 1976.

[27] De Floriani, L., Maulik, A. et Nagy, G. Manipulating a modular boundary model with a face-based graph structure. In Geometric Modeling for Product Engineering, Wozny, M. *et al*, editors. IFIP, 1990.

[28] Lear, D. A. Practical applications of algebraic topology to geometric design. Working paper, August 1992 IFIP WG 5.2 Workshop notes. Fourth IFIP WG 5.2 Workshop on Geometric Modeling and Computer-Aided Design, 27 septembre - 1 octobre, Rensselaerville, NY, 1992.

[29] Egli, R., Petit, C., et Stewart, N. F. Moving coordinate frames for representation and visualization in four dimensions. *Computers and Graphics* (20), No. 6, 905-919, 1996.

[30] Egli, R. http ://www.iro.umontreal.ca/~egli.

[31] Brière, N. et Poulin, P. Hierarchical view-dependent structures for interactive scene manipulation. *Computer Graphics, Annual Conference Series,* 83-90, 1996.

[32] Lambardo, J.-C. et Puech C. Oriented particules : a tool for shape memory objects modeling. *Graphics Interface '95,* 255-262, 1995.

[33] Sokolnikoff, I. S. et Redheffer, R. M. Mathematics of Physics and Modern Engineering. McGraw-Hill, 1958.

Annexe A

Glossaire

Ce glossaire donne la définition pour certains mots qui ne sont pas définis explicitement dans le présent document. Comme la correspondance entre les termes anglais et français peut parfois porter à confusion, les termes traduits se retrouvent aussi ici.

attribut mot utilisé en infographie qui est équivalent au mot "coefficient" dans le formalisme du modèle de chaînes.

interface de programmation *API, Application Programming Interface.*

cadre de travail *framework.*

chaîne avec des coefficients primitifs chaîne dont les coefficients ne sont pas des sous-chaînes. Par exemples, des réels, des vecteurs dans R^3, etc.

modèle de chaînes *chain-model.*

modélisation solide *solid modelling.*

coefficient mot utilisé dans le formalisme du modèle de chaînes qui est équivalent au mot "attribut" en infographie.

groupe un groupe G est défini comme étant un ensemble muni d'une opération binaire interne, soit $* : G \times G \to G$, qui est associative, qui admet un élément

neutre $e \in G$ et pour laquelle tout élément $x \in G$ possède un inverse unique, c'est-à-dire un élément de G, noté x^{-1}, tel que

$$x * x^{-1} = x^{-1} * x = e.$$

simplexe cas particulier d'un complexe. Il y a exactement $p + 1$ 0-cellules dans une p-cellule d'un simplexe.

variété *manifold.*

Annexe B

Hiérarchie des classes pour les chaînes.

La hiérarchie des classes est indiquée à la figure B.1. La classe de base est *Chain*. Elle contient les méthodes communes à toutes les classes représentant une chaîne. Par exemple, la méthode *complex* qui indique sur quel complexe la chaîne est définie. De la classe *Chain* dérivent deux classes, la classe *Template_Chain_Type* et la classe *Template_Chain2_Type*. La classe *Template_Chain_Type* est une classe générique (*template*) dont dérivent les classes *Chain_Type*, c'est-à-dire toutes les classes de chaînes de type primitif (qui ne sont pas des chaînes de sous-chaînes). Nous avons la classe *Template_Chain2_Type* qui est aussi une classe générique dont dérivent les classes *Chain2_Type*, c'est-à-dire toutes les classes de chaînes de sous-chaînes.

À la figure B.1, nous avons indiqué seulement les classes *Chain_Real* et *Chain_Real3* qui dérivent de la classe *Template_Chain_Type*. Mais en fait, toutes les classes de chaînes de type primitif héritent de la classe *Template_Chain_Type* et le programmeur pourra facilement en ajouter d'autres. Les classes de chaînes de sous-chaînes (*Chain2_Real* et *Chain2_Real3*), quant à elles, dérivent de la classe

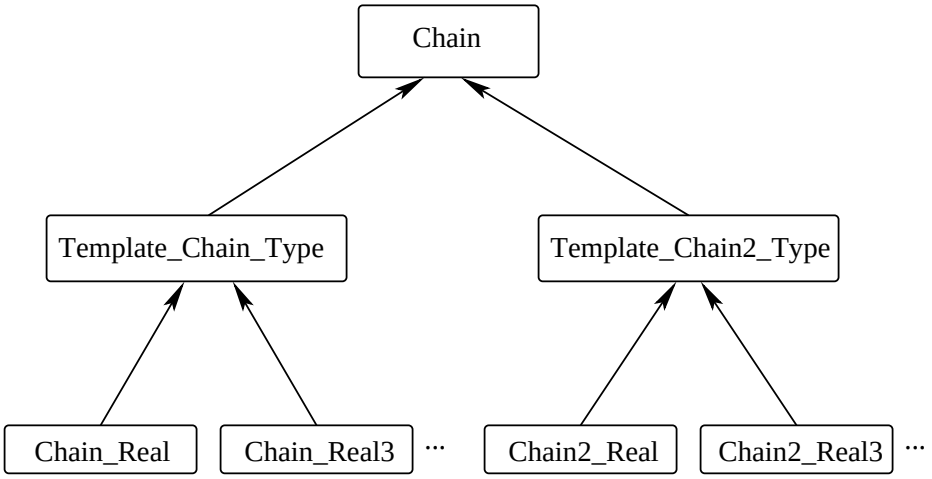

FIG. B.1 – Hiérarchie des classes.

Template_Chain2_Type. Le programmeur pourra aussi ajouter à volonté d'autres classes de chaînes de sous-chaînes.

Remarquez que nous avons fait deux classes génériques dans le but d'éviter au programmeur de redéfinir les méthodes qui sont communes aux classes dérivées mais qui sont spécifiques aux chaînes de sous-chaînes ou aux chaînes de type primitif. Par exemple, la méthode *coefficient(Cell cp, Cell cq)* (voir section 4.3) qui est applicable sur une chaîne de sous-chaînes *Chain2_Type*. Remarquez que cette méthode retourne un coefficient de type *Type*. Voilà pourquoi nous avons des classes génériques. Pour être plus précis, les classes génériques définissent une classe pour chacune des classes *Chain_Type* et *Chain2_Type*. Elles doivent être génériques pour savoir qu'elles sont les classes dérivées de celles-ci.

Les objets qui sont des sous-chaînes ou des sélections sur des chaînes sont intégrés à la classe *Chain_Type*. Cela permet au programmeur de définir une seule fois les opérations binaires et les méthodes qui sont appliquées, cellule par cellule. Nous aurions pu aussi utiliser l'héritage multiple, mais en C++, cela est plutôt complexe.