N-g-cartes : structure, construction, parcours, géométrie

Thierry EXCOFFIER

30 septembre 1998

1 Introduction

Ce document présente une implémentation des n-g-cartes, quelles sont les structures de données, les fonctions de constructions ainsi que les fonctions de parcours.

La généralisation à une dimension quelconque permet une implémentation plus simple.

2 Rappel

Une g-carte est un graphe de brins vérifiant les propriétés topologique des cartes.

Si `a` est un brin alors `a.alpha[n]` est un brin qui est connecté à `a` par une liaison de dimension `n`. La première propriété est que :

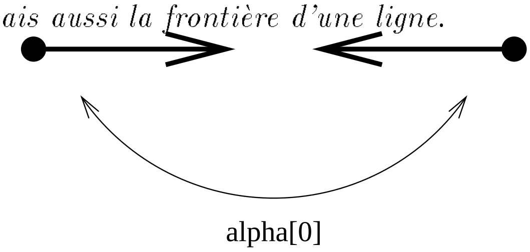
`a.alpha[n].alpha[n] = a`

2.1 (-1)-g-carte

C'est la g-carte pour laquelle les brins ne sont reliés à rien. *C'est l'ensemble vide mais aussi la frontière d'un point.*

2.2 0-g-carte

C'est l'ensemble des g-cartes formée d'une paire de brins `a` et `b` avec `a.alpha[0] = b` et `b.alpha[0] = a`. *C'est une paire de points mais aussi la frontière d'une ligne.*

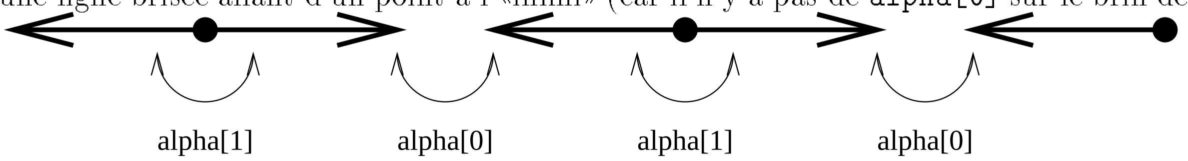


Si `a = b` alors la g-carte représente une boucle.

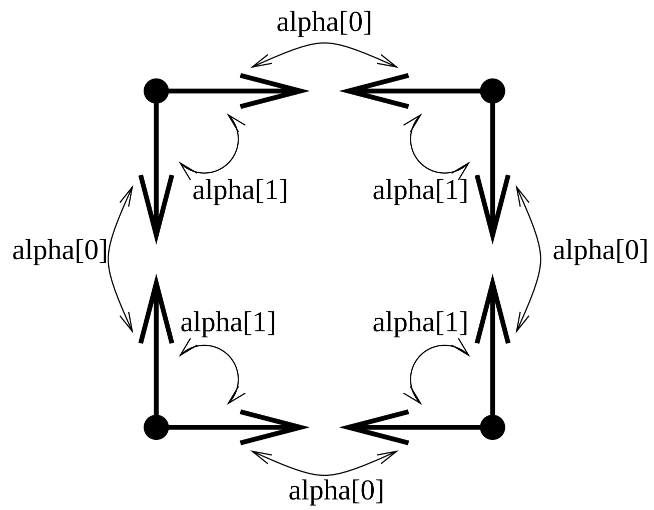
2.3 1-g-carte

Ce sont les lignes brisées ouvertes ou fermées mais aussi des frontières de face

Voici une ligne brisée allant d'un point à l'«infini» (car il n'y a pas de `alpha[0]` sur le brin de gauche).

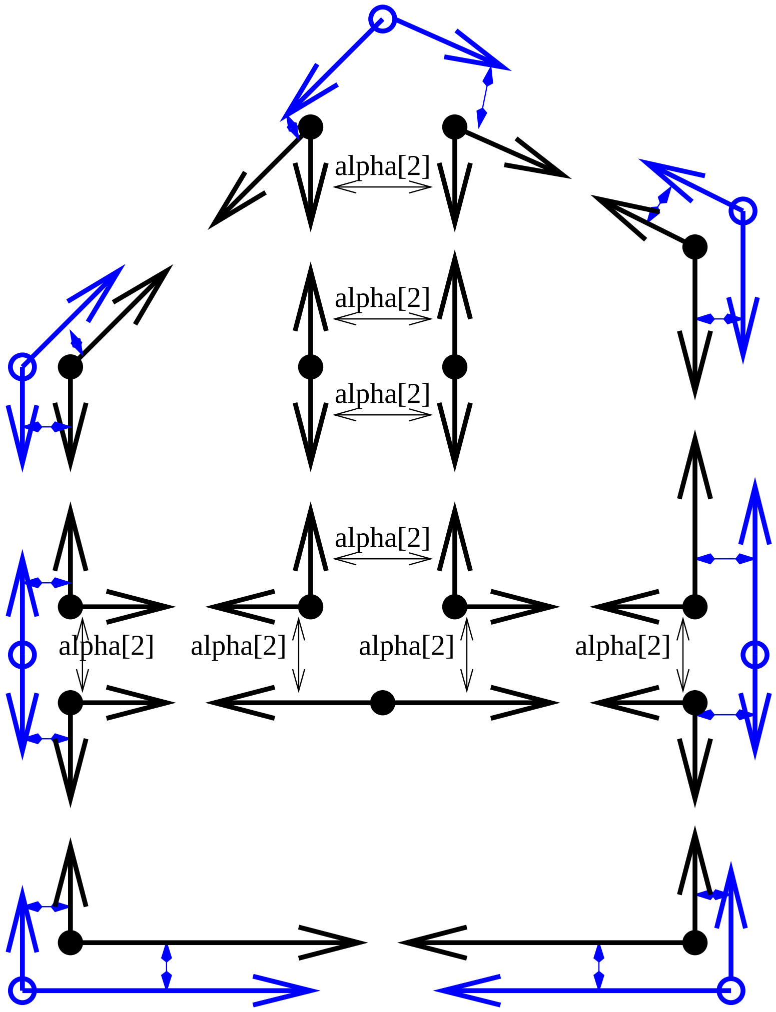


Un cas plus intéressant est la ligne brisée fermée qui est la base de construction des faces non-dégénérées.



2.4 2-g-carte

Ce sont des graphes de faces mais aussi des frontières de polyèdres
La figure ne montre plus les `alpha[0]` et `alpha[1]` afin d'être plus lisible.



2.5 Quelques remarques importantes

- Les `alpha[n]` non définis sont considérés comme étant l'identité.
- Si `alpha[0]` n'existe pas, la correspondance géométrique est une arête allant à l'infini.

- Il y a énormément de contraintes à respecter pour que la g-carte soit valide.
- Les dessins ci-dessus n'exposent que des cas non dégénérés. Une carte avec seulement quelques brins peut définir des topologies très complexes.

3 Construction des n-g-cartes

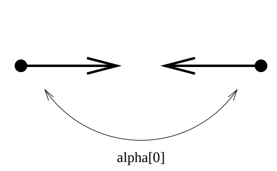
Les n-g-cartes, quel que soient leur complexité peuvent-être générées avec seulement deux opérations :

- La création d'un brin de (-1)-g-carte : `Brin()`
- Collage de deux brins par un `alpha[n]` : `Coller(a,b,n)`

  Cette opération met la n-g-carte à jour de manière à garantir la cohérence (si c'est possible). L'algorithme réalisé par cette fonction sera défini par la suite. La fonction `Coller` retourne `a`.

- Création d'une arète :

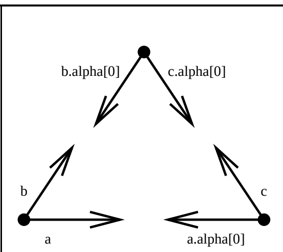
```
Arète()
{
a := Brin() ;
b := Brin() ;
Coller(a, b, 0) ;
return a ;
}
```




```
a.alpha[0] := b ;
b.alpha[0] := a ;
```

- La création d'un triangle est un peu plus complexe car on a besoin de faire un bouclage. On ne détaille pas les 6 affectations faites par la fonction `Arète` (les `alpha[0]`).

```
Triangle()
{
a := Arète() ;
b := Arète() ;
c := Arète() ;
Coller(a, b, 1) ;
Coller(a.alpha[0], c, 1) ;
Coller(b.alpha[0], c.alpha[0], 1) ;
return a ;
}
```

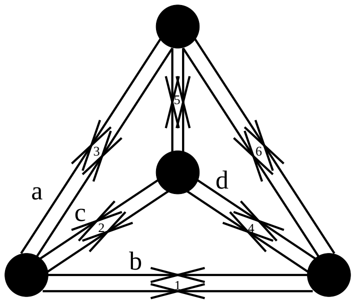



```
a.alpha[1] := b ;
b.alpha[1] := a ;
a.alpha[0].alpha[1] := c ;
c.alpha[1] := a.alpha[0].alpha[1] ;
b.alpha[0].alpha[1] := c.alpha[0] ;
c.alpha[0].alpha[1] := b.alpha[0] ;
```

- Et voici la fonction de création d'un tétraèdre. On ne détaille pas les 4*6 affectations faites par la fonction `Triangle` (les `alpha[1]`) et les 4*6 affectations indirectement faites par la fonction `Arète` (les `alpha[0]`).

  Les chiffres indiqués sur le graphique correspondent à l'ordre des opérations de collage.

```
Tétraèdre()
{
a := Triangle() ;
b := Triangle() ;
c := Triangle() ;
d := Triangle() ;
Coller(a.alpha[1], b, 2) ;
Coller(b.alpha[1], c, 2) ;
Coller(c.alpha[1], a, 2) ;
Coller(d,
       b.alpha[1].alpha[0].alpha[1],
       2) ;
Coller(d.alpha[1],
       c.alpha[0].alpha[1],
       2) ;
Coller(d.alpha[0].alpha[1],
       a.alpha[1].alpha[0].alpha[1],
       2) ;
return a ;
}
```




```
a.alpha[1].alpha[2]  := b ;
b.alpha[2]  := a.alpha[1] ;
a.alpha[1].alpha[0].alpha[2]  := b.alpha[0] ;
b.alpha[0].alpha[2]  := a.alpha[1].alpha[0].alpha[2] ;

b.alpha[1].alpha[2]  := c ;
c.alpha[2]  := b.alpha[1] ;
b.alpha[1].alpha[0].alpha[2]  := c.alpha[0] ;
c.alpha[0].alpha[2]  := b.alpha[1].alpha[0].alpha[2] ;

c.alpha[1].alpha[2]  := a ;
a.alpha[2]  := c.alpha[1] ;
c.alpha[1].alpha[0].alpha[2]  := a.alpha[0] ;
a.alpha[0].alpha[2]  := c.alpha[1].alpha[0].alpha[2] ;

d.alpha[2]  := b.alpha[1].alpha[0].alpha[1] ;
b.alpha[1].alpha[0].alpha[1].alpha[2]  := d ;
d.alpha[0].alpha[2]  :=
    b.alpha[1].alpha[0].alpha[1].alpha[0] ;
b.alpha[1].alpha[0].alpha[1].alpha[0].alpha[2]  :=
    d.alpha[0] ;

d.alpha[1].alpha[2]  := c.alpha[0].alpha[1] ;
c.alpha[0].alpha[1].alpha[2]  := d.alpha[1] ;
d.alpha[1].alpha[0].alpha[2]  :=
    c.alpha[0].alpha[1].alpha[0] ;
c.alpha[0].alpha[1].alpha[0].alpha[2]  :=
    d.alpha[1].alpha[0] ;

d.alpha[0].alpha[1].alpha[2]  :=
    a.alpha[1].alpha[0].alpha[1] ;
a.alpha[1].alpha[0].alpha[1].alpha[2]  :=
    d.alpha[0].alpha[1] ;
d.alpha[0].alpha[1].alpha[0].alpha[2]  :=
    a.alpha[1].alpha[0].alpha[1].alpha[0] ;
a.alpha[1].alpha[0].alpha[1].alpha[0].alpha[2]  :=
    d.alpha[0].alpha[1].alpha[0] ;
```

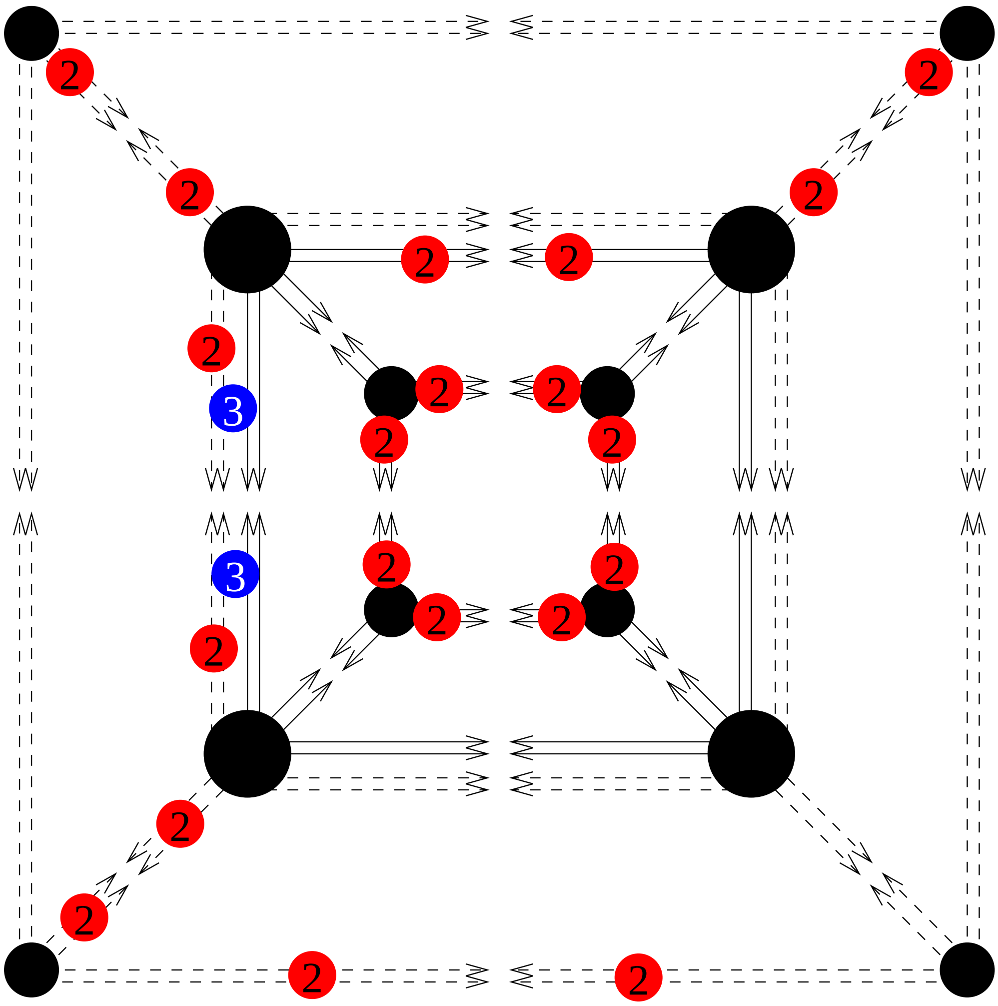
4 Le concept de porte

Une structure intéressante pour le parcour des n-g-cartes est l'arbre des cellules. En effet avec cet arbre de cellules, on peut parcourir toutes les entités d'une n-g-carte sans utiliser ni une fonction récursive, ni marqueur, ni tableau. L'algorithme sera détaillé par la suite.

Comment coder l'arbre dans la n-g-carte ? Tous simplement en notant pour chaque dimension `n` possible si les deux brins reliées suivant cette dimension sont connexes dans l'arbre des cellules. Si oui, on a une porte de dimension `n`.

`a.porte[n]` est vrai si `a` et `a.alpha[n]` sont connexes dans l'arbre des cellules

Voici un arbre de cellule (parmi beaucoup d'autres) pour deux cubes collés face contre face. En fait c'est un arbre composé de deux volumes et chaque volume est un arbre de six faces. Les cercles gris numéroté 2 ou 3 indiquent que les deux brins qu'ils encerclent sont un lieu de passage d'un cellule de dimension 2 ou 3 à une autre cellule de même dimension.



Comment déterminer l'arbre des cellules? Tout simplement, lors du collage de dimension `n`, il y a une porte si les deux brins que l'on colle appartiennent à des n-g-cartes différentes. La création d'une porte revient à souder deux n-g-cartes pour n'en former qu'une seule.

Pour simplifier l'algorithme de parcour du graphe, on considérera que tous les passages dimension 1 sont des portes bien que cela ne soient pas toujours le cas (La fermeture d'un polygone ne devrait pas faire une porte de dimension 1)

5 Structures de données

Si l'on met de coté les données globales à la cartes ainsi que les données géométriques, la structure de données restante est celle du brin.

La structure minimum codant un brin est une table de pointeurs sur brins (`alpha`) et une table de booléens (`porte`) l'indice de ces tables est la dimension suivant laquelle les deux brins sont liées.

6 Brin précédent

C'est la base de tous les algorithmes, en effet cette fonction permet de parcourir la moitié (l'autre moitié est accessible via `alpha[0]`) des brins d'une n-g-carte (avec portes).

Il est aussi pratique de limiter l'exploration de la n-g-carte à une dimension fixée. Par exemple pour limiter l'exploration d'un cube (cellule de dimension 3) à seulement celle d'une face du cube (cellule de dimension 2). On notera `N` cette dimension maximum de parcour.

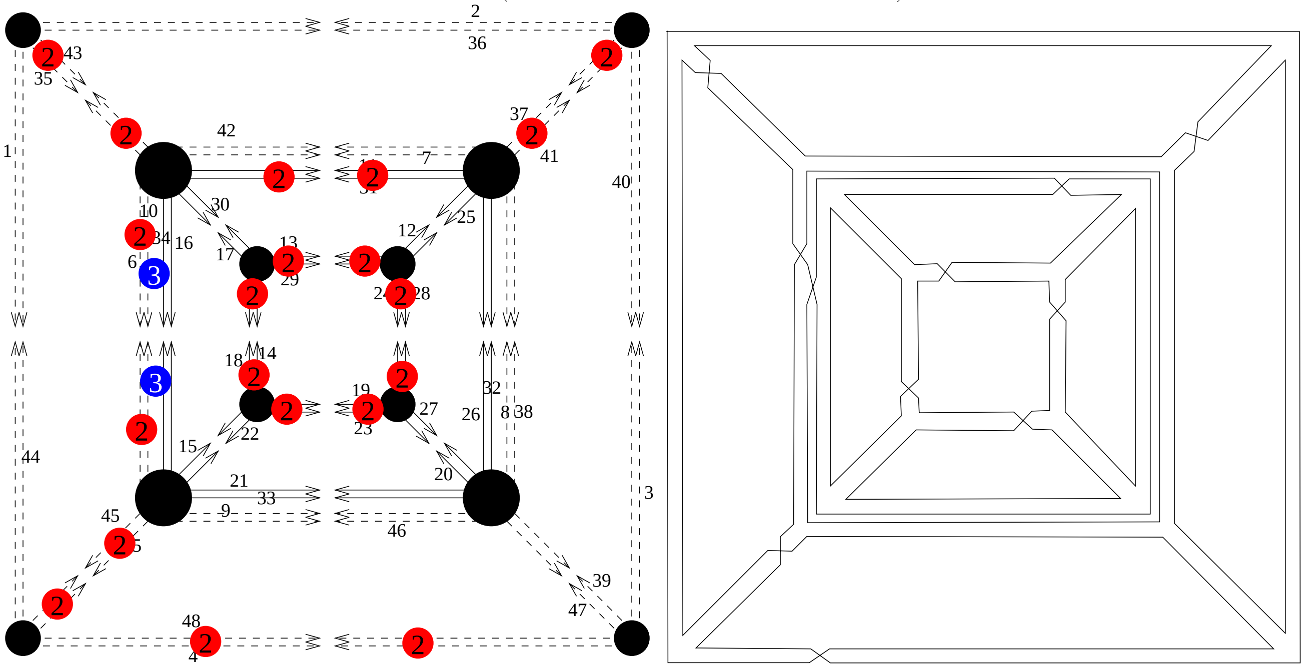
```
Précédent(brin, N)
   {
```

```
   for( i from N to 0 by -1 )
     if ( brin.porte[i] )       /* Il y a une porte ? */
       brin := brin.alpha[i] ;
     end if ;
   end for ;
   return brin ;
  }
```

On notera `A(b,N)` l'ensemble des brins parcourus en partant de `b` (`b` compris) en utilisant `Précédent(b,N)`.

Application pratique : avec des perles (représentant les arètes) et un seul fil, on peut réaliser n'importe quel polyèdre ou décomposition cellulaire de l'espace (quelque soit sa dimension).

Voici le parcour de la paire de cube (avec l'arbre de cellule donné) :



7 Collage

Pour être possible, le collage doit respecter les conditions suivantes :

- On ne colle pas par la dimension `n` des cellules de dimension strictement supérieures à `n`
- On ne colle qu'une seule fois un brin suivant une dimension. (on n'écrase pas les pointeurs `alpha[]` avec une nouvelle valeur)
- On colle seulement des structures identiques. Ce test est extremement difficile à réaliser car des arbres de cellules différents peuvent coder des structures identiques. L'implémentation actuelle ne le fait pas.

La fonction suivante permet de parcourir une cellule `a` en entrainant un parcour identique sur la cellule `b`. Ceci est obligatoire si l'on veut coller des cellules de dimension supérieur ou égale à 3 (via `alpha[4]` ou plus), en effet les arbres de sous-cellule peuvent être différents.

```
Paire-Précédente(a, b, N) /* Si N<0 cela retourne la paire sans changement */
  {
   for( i from N to 0 by -1 )
     if ( a.porte[i] )
       a := a.alpha[i] ;
```

```
      b := b.alpha[i] ; /* Les cartes de a et de b sont différentes */
                        /* si b.alpha[i] n'est pas défini */
                        /* Dans ce cas le collage est impossible */
    end if ;
  end for ;
  return a, b ;
 }
```

Le principe du collage est simple, si l'on colle deux brin suivant la dimension `n` on doit lier tous les brins des deux cellules de dimension `n-1`.

Pour parcourir tous les brins d'une cellule de dimension `n-1` il suffit d'utiliser les `alpha[k]` avec `k`≤`n`-2. Par exemple pour lier deux faces (`n`=2) on utilise les `alpha[k]` avec `k`≤0, on ne relie donc que deux paires de brins ensemble.

Pour simplifier l'algorithme, les vérifications de validité de collage ne sont pas montrées.

```
Coller(a,b,n)
 {
  /* Creation de la porte */

  if ( n < 2 | Non-connexe?(a,b) )
     a.porte[n] := Vrai ;
     b.porte[n] := Vrai ;
     if ( n > 1 )                      /* Porte dans la direction opposee */
        a.alpha[0].porte[n] := Vrai ;
        b.alpha[0].porte[n] := Vrai ;
     end if ;
  end if ;

  /* Création de tous les alpha[n] nécessaires */

  (ia , ib) := (a, b) ;
  FAIRE
     ia.alpha[n] := ib ;
     ib.alpha[n] := ia ;
     (ia, ib) := Paire-Précédente(a,b, n - 2) ;
  TANT QUE ia != a ;
 end if ;
 return a ;
```

8 P(b,n,N) : Ensemble des brins "parallèles"

C'est l'ensemble des brins que l'on peut atteindre à partir d'un brin en utilisant que des `alpha[i]` avec `n`≤ `i` ≤ `N`

$$\mathtt{P}(\mathtt{b},\mathtt{n},\mathtt{N}) = b \cup \bigcup_{\mathtt{n}\leq\mathtt{i}\leq\mathtt{N}} \mathtt{P}(\mathtt{b}[\mathtt{i}],\mathtt{n},\mathtt{N})$$

Si l'on prend N la dimension de la carte et un brin de départ de parcour, on obtient suivant `n` :

n	Orbite
0	Tous les brins (rien de commun)
1	Les brins qui ont un sommet commun
2	Les brins qui ont un sommet/arête commun
3	Les brins qui ont un sommet/arête/face commun

Malheureusement pour n>0, le parcour de l'orbite nécessite marqueurs et tables car l'utilisation des portes n'est plus possible. En effet, en enlevant une partie de la n-g-carte, l'arbre des cellules peut perdre sa connexité.

9 Classe d'équivalence de brins

Soit :

- d la dimension topologique pour laquelle les brins sont équivalents.
- N la dimension topologique de la cellule contenante.
- b un représentant de la classe d'équivalence.

On peut définir la classe d'équivalence comme :

$$\mathtt{C}(\mathtt{b},\mathtt{d},\mathtt{N}) = b \cup \bigcup_{0 \leq \mathtt{i} \leq \mathtt{N} \text{ et } \mathtt{i} \neq \mathtt{d}} \mathtt{C}(\mathtt{b}[\mathtt{i}],\mathtt{d},\mathtt{N})$$

Le cardinal de la classe d'équivalence est constant pour certaine valeurs, pour d'autre il dépend de la n-g-carte. L'exemple suivant donne le cardinal dans le cas général des polyèdres, le cas du cube et celui de deux cubes accolés par une face.

d	N	L'ensemble des brins :	Général	Cube	*Bicube*
0	1	partant du même sommet que b et dans la même face	2	2	2
0	2	partant du même sommet que b et dans le même volume	?	6	6
0	3	partant du même sommet que b et dans le même hyper-volume	?	6	6 ou 12
1	1	de même arète que b et dans la même face	2	2	2
1	2	de même arète que b et dans le même volume	4	4	4
1	3	de même arète que b et dans le même hyper-volume	?	4	4 ou 8
2	2	de même face que b et dans le même volume	?	4	4
2	3	de même face que b et dans le même hyper-volume	?	4	4 ou 8

Comment parcour-t-on ces classes d'équivalences ? Si $\mathtt{d} \geq \mathtt{N}$ cela revient à faire un parcour total de la cellule avec `Précédent`, sinon c'est un peu plus compliqué, pour un brin de départ `b` :

```
POUR tous les brins c de A(b, d - 1)
   POUR tous les brins d de P(c, d + 1, N)
      Affiche(d) ;
   end for ;
end for ;
```

10 Géométrie et autres attributs

Les intérêts d'une séparation totale de la géométrie et de la topologie sont nombreux :

- Les structures de données topologiques sont les même quelque soit les données stockées.
- On peut associer plusieurs instances géométriques à une instance topologique.
- Seule la place réellement utiles est occupée.

L'information géométrique est une entité de dimension géométrique `d` donnée : 0 (point), 1 (droite), 2 (plan), ...

L'information graphique est lié à une information géométrique de dimension d mais elle peut aussi est locale à une cellule de dimension `N`. Par exemple une même arête peut avoir une couleur différente suivant la face à laquelle elle appartient.

Pour accéder à toutes ces informations, en évitant la redondance, on utilise une fonction et une table à trois dimensions contenant les informations géométriques.

Première étape : trouver le brins contenant l'information géométrique :

```
trouve-brin-avec-géométrie(b,d,N)
{
   POUR tous c dans C(b, d, N)
       SI table-géométrie[c,d,N] contient une valeur
           RETOURNE c
   RETOURNE Faux
}
```

Récupération de l'information géométrique :

```
geometrie(b,d,N)
{
   bg :=  trouve-brin-avec-géométrie(b,d,N) ;
   SI ( bg = Faux )
       RETOURNE Faux ;
   SINON
       RETOURNE table-géométrie[bg,d,N] ;
}
```

Création/Modification d'une information géométrique :

```
change-geometrie(b,d,N, g)
{
   bg :=  trouve-brin-avec-géométrie(b,d,N) ;
   SI ( bg = Faux )
       table-géométrie[b,d,N] := g ;
   SINON
       table-géométrie[bg,d,N] := g ;
}
```

Le premier indice de `table-géométrie` n'est pas un entier mais un brin, on utilise le *hash-coding* pour cet indice. L'utilisation du *hash-coding* permet de réduire la taille du tableau, en effet «peu» de brins contiennent des informations géométrique.

Ce codage est minimal en place car seul les informations existantes prennent de la place (il n'y a pas d'informations redondantes). Par contre son temps d'accès est lent car il nécessite de parcourir toute la classe d'équivalence pour trouver le brin contenant l'information.

On peut bien sur avoir plusieur tableau d'information géométrique dont les indices sont des brins de la même carte.

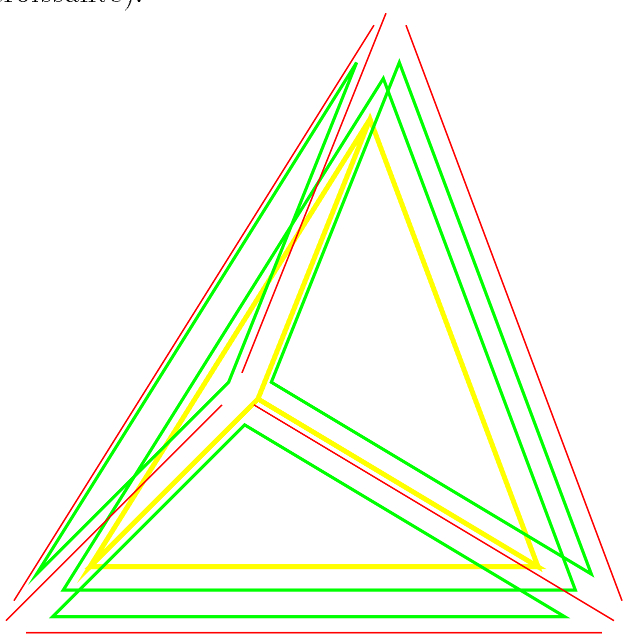
11 Affichage

On peut tracer à l'aide de lignes brisées toutes les entités d'une n-g-carte dont on donne un brin `t`.

On suppose l'accès aux coordonnées des points est transparent à travers `coord` et que tous les représentants canoniques sont connus.

```
for( d from N to 0 by -1 )            /* Les dimensions possibles de cellule */
   POUR tous les brins b de A(t,N)         /* Recherche toutes les cellules */
      if ( Représentant-canonique(b,d,d) ) /* Trace une fois chaque cellule */

        // Calcul centre de la cellule de dimension "d"

        p := point() ;
        nb := 0 ;
        POUR tous les brins v de A(b,d)          /* Les brins de la cellule */
           p := p + coord[v] ;
           nb := nb + 1 ;
        end for ;
        p := p / nb ;

        // Affiche la ligne brisée passant par tous les sommets de la cellule

        Nouvelle-ligne-brisée() ;
        POUR tous les brins v de A(b,d)          /* Les brins de la cellule */
           Ajoute-point-à-la-ligne-brisée( T[v] * 0.9  +  p * 0.1 ) ;
        end for ;
        Fin-ligne-brisée() ;
      end if ;
   end for ;
end for ;
```

Pour un tétraèdre (les cellules tracées sont celles de dimensions 3, 2, 1 avec une épaisseur de trait décroissante).



12 Questions en suspend

L'opération de collage fonctionne parfaitement si on lui donne des cellules collables. Dans le cas contraire, elle ne détecte pas forcément l'incohérence.

En effet cette opération de vérification est complexe. Il faudrait prouver que deux brins a et b permettent d'accéder à deux cartes identiques. Le test suivant peu “manquer” certaines information : *Une fois le parcours de b guidé par a terminé, on considére que les deux cartes sont identiques si tous les brins de la carte b ont été parcourus une seule et unique fois.*

Le collage par dimension inférieur. Que ce passe-t-il si l'on colle des `alpha[n<m]` après avoir collé des `alpha[m]` ? Obtient-on une n-g-carte valide ?

L'association du `alpha[n]` à une entité topologique de dimension `n` n'est pas forcément une bonne approche. Il serait interessant d'associer le `alpha[2n]` à la dimension topolgique `n` et d'utiliser les `alpha` restants pour coder des cellules “pendantes”.

13 Implémentation

Elle fonctionne en DYLAN :

La bibliothèque en cours de développement est dans :
`/home/exco/LOGICIELS/MINDY/GEOMETRIE/n-g-carte.dylan`

Un exemple d'utilisation dans :
`/home/exco/LOGICIELS/MINDY/GEOMETRIE/test6.dylan`

Les fichiers sont malheureusement peu commentés. De plus, pour des raisons d'implémentation le 0 est souvent utilisé pour indiquer la dimension topologique maximum.