Généricité & topologie discrète en C++

Julien LAMY, lamy@unistra.fr
Laboratoire d'Imagerie et de Neurosciences Cognitives
Université de Strasbourg

INTRODUCTION

- Image : signal continu
- Ordinateur : mieux adapté aux signaux discrets
- Traitement d'images
  - Discrétisation d'un signal
  - Passage de $\mathbb{R}^n$ à $\mathbb{Z}^n$
- Géométrie : bonne maîtrise des notions d'angle et de distance dans $\mathbb{Z}^n$
- Topologie : 4- et 8-voisinages en 2D, 6-, 18, et 26-voisinages en 3D, connexités objet/fond, ...

INTRODUCTION

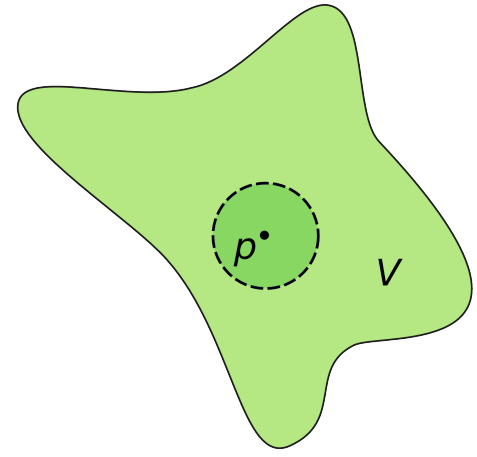
- But de l'exposé
  - Solution générique (en C++) pour l'intégration d'informations topologiques (discrètes) dans une bibliothèque de traitement d'images
- Plan
  - Notions de base de topologie
  - Équivalents en topologie discrète
  - Intégration dans une bibliothèque de traitement d'images

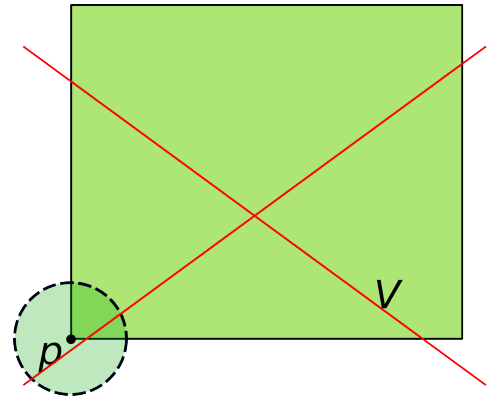
1. Bases de topologie

Voisinage
Courbes & surfaces
Squelette

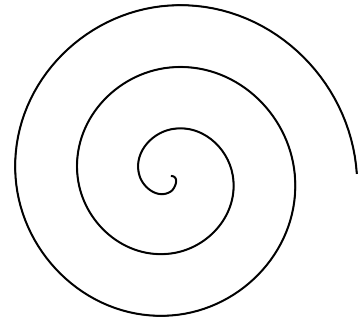
VOISINAGE

- Cas général
  - Voisinage de p : ensemble contenant un ouvert contenant p
- Espace euclidien (en fait tout espace métrique)
  - Voisinage de p : ensemble contenant une boule ouverte contenant p

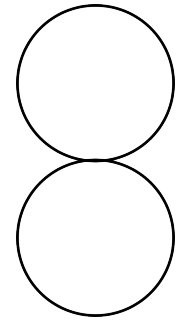


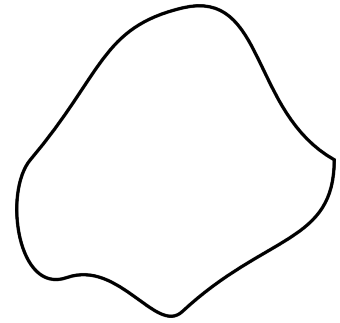


COURBE



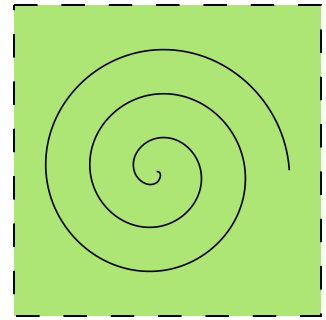
- Définition : application γ de $I \subset \mathbb{R} \rightarrow X$
- Courbe simple : application injective : $\gamma(x) = \gamma(y) \Rightarrow x = y$ (on ne repasse pas par le même point)
- Courbe fermée : $I = [a, b]$ et $\gamma(a) = \gamma(b)$

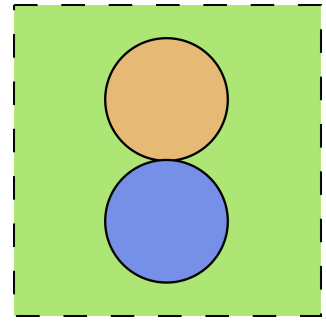


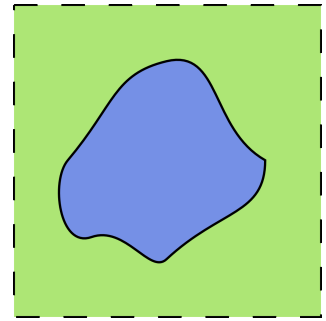


Théorème de Jordan

- Le complément de l'image d'une courbe fermée simple de $\mathbb{R}^2$ a deux composantes connexes :
  - L'une finie (l'intérieur)
  - L'autre infinie (l'extérieur)

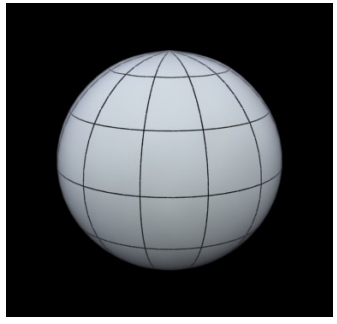






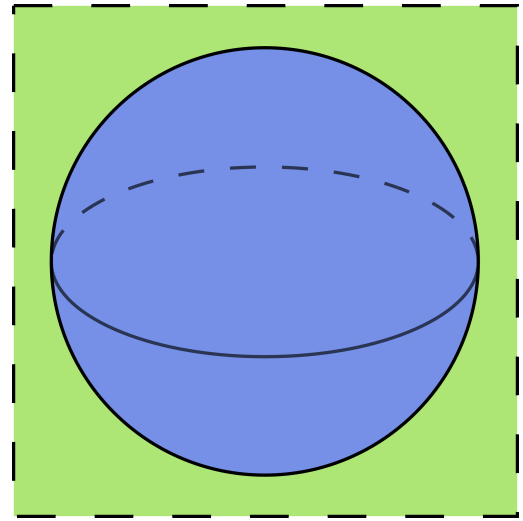
THÉORÈME DE JORDAN-BROUWER

- Extension du théorème de Jordan en n dimensions
- Courbe fermée simple de $\mathbb{R}^2$: application injective de S^1 dans $\mathbb{R}^2$
- Surface fermée simple de $\mathbb{R}^3$: application injective de S^2 dans $\mathbb{R}^3$



THÉORÈME DE JORDAN-BROUWER

- Une application continue injective de S^n dans $\mathbb{R}^{n+1}$ divise $\mathbb{R}^{n+1}$ en deux composantes connexes, l'une finie (l'intérieur), l'autre infinie (l'extérieur)

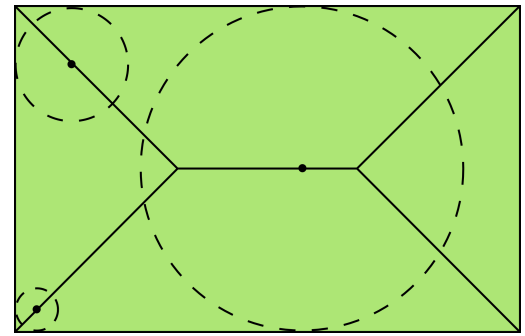


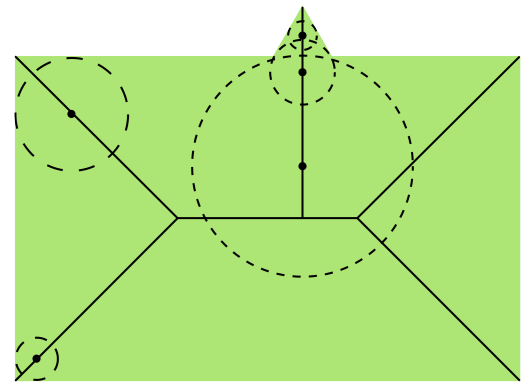
SQUELETTE

- Représentation simplifiée d'un objet
  - Fine
  - Équidistante des bords
  - Conserve la topologie
  - Conserve certaines propriétés géométriques (direction, élongation)
- Définition intuitive : feu de prairie (Blum, 1967)
  - En allumant un feu sur la frontière de l'objet, le squelette est constitué des points de rencontre des fronts de flamme

SQUELETTE

- Définition formelle : cercles bi-tangents
  - Ensemble des centres des sphères tangentes en au moins deux points de la frontière de l'objet





2. TOPOLOGIE DISCRÈTE

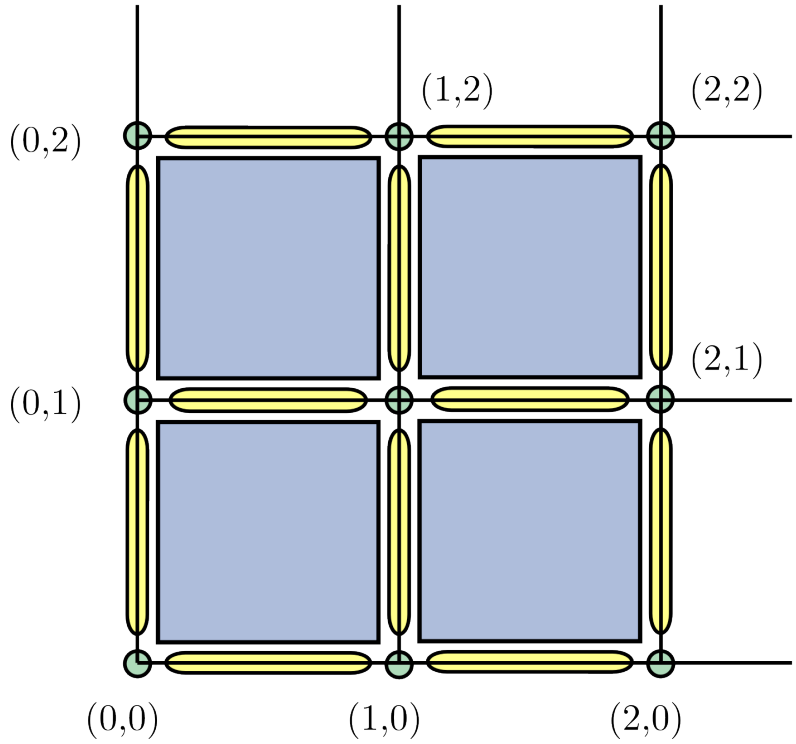
Décomposition cellulaire
Voisinage, connexité
Squelette

TOPOLOGIE DISCRÈTE

- Différences de voisinage : soit un voisinage V
  - Dans $\mathbb{R}^n$, il existe $V' \subsetneq V$ tel que V' est un voisinage de p
  - Dans $\mathbb{Z}^n$, pixels/voxels indivisibles
- Pour redéfinir les notions, utilisation d'un espace intermédiaire : décomposition cellulaire (Kovalevski)

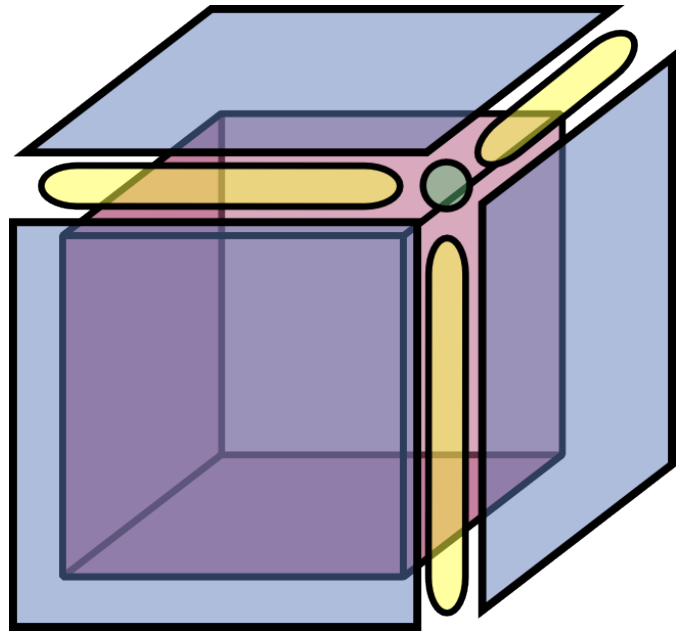
DÉCOMPOSITION CELLULAIRE

- Généralise les notion de sommets, arêtes, faces
- k-cellule de $\mathbb{R}^n$: $c = I_1 \times \ldots \times I_n$
  - k intervalles de la forme $I_i =]z_i, z_i + 1[$
  - $n - k$ intervalles de la forme $I_i = \{z_i\}$



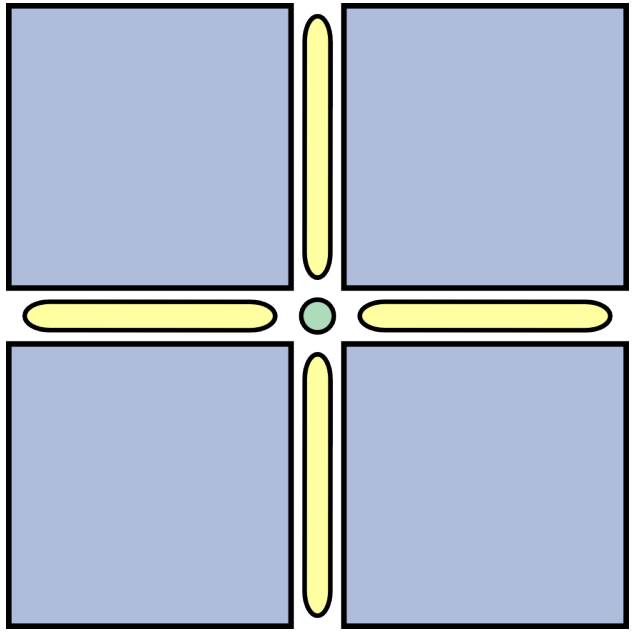
DÉCOMPOSITION CELLULAIRE

- *n*-cellule : volume unitaire, pixel/voxel
- 0-cellule : sommet d'un pixel/voxel



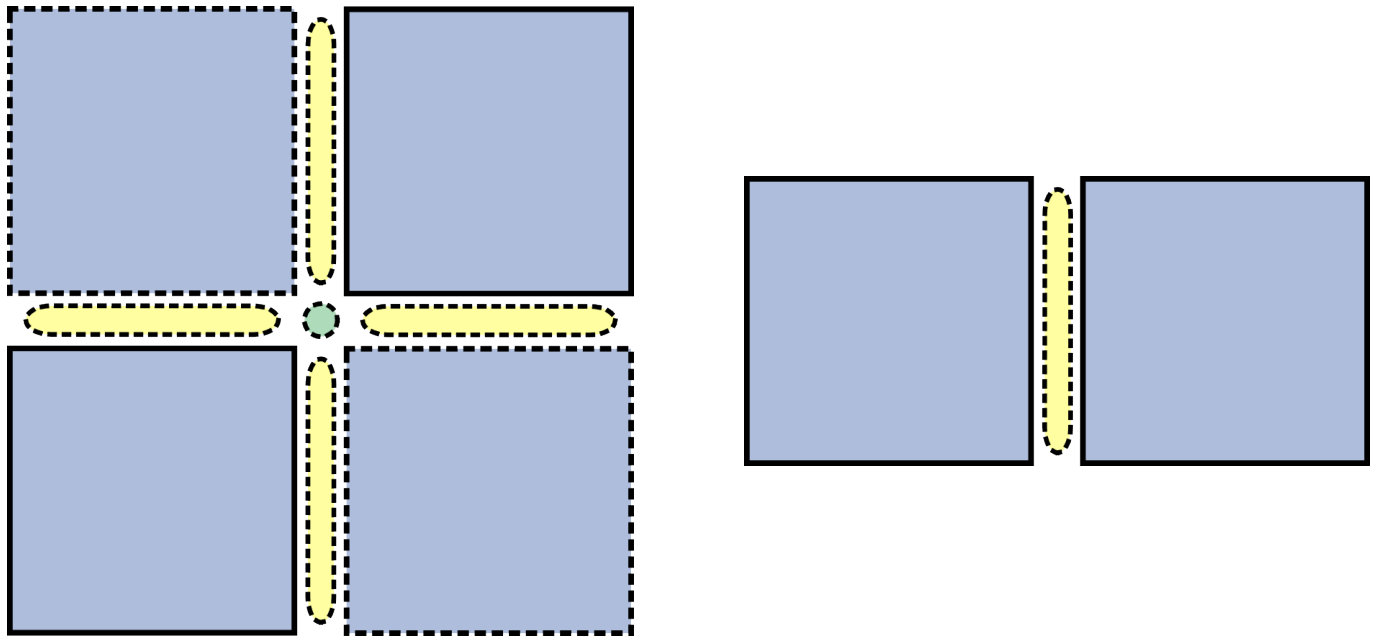
DÉCOMPOSITION CELLULAIRE

- Étoile ouverte $St(c) = J_1 \times \ldots \times J_n$
  - $J_i = I_i$ si $I_i =]z_i, z_i + 1[$
  - $J_i =]z_i - 1, z_i + 1[$ si $I_i = \{z_i\}$
- $St(c)$ est l'union de c et des cellules adjacentes de dimension supérieure
- Étoile d'une 0-cellule de R2 :



VOISINAGE

- Soit une image de dimension D
- Deux D-cellules c_1, c_2 sont (D, k)-voisines s'il existe une k-cellule c ($k < d$) telle que $c_1 \in St(c)$ et $c_2 \in St(c)$
- (2,0) et (2,1)-voisinages :

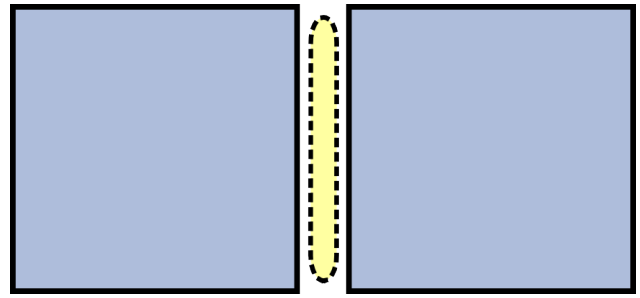


VOISINAGE

- Nombre de voisins : deux (D, k)-voisins ont
  - k coordonnées égales
  - $D - k$ coordonnées différant de ± 1
- Un (D, k)-voisinage contient $\begin{pmatrix} D \\ k \end{pmatrix} 2^{D-k}$ éléments
- Notation facilitant le dénombrement des voisins

CONNEXITÉ

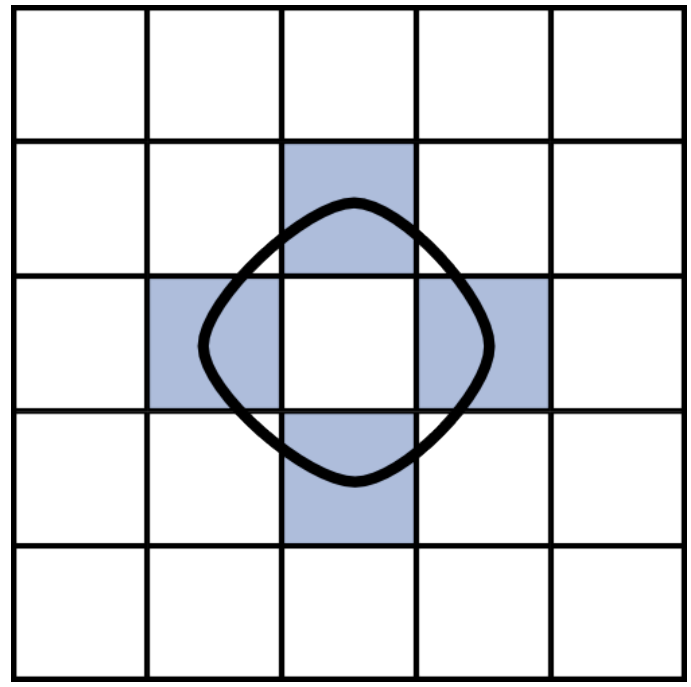
- Deux D-cellules c_1 et c_2 sont (D, k)-connexes s'il existe $j \in [k, D-1]$ tel que c_1 et c_2 sont (D, j)-voisines
- Cellules 0-connexes et 1-connexes car 1-voisines :



- En 2D
  - 4-connexité : (2,1)-connexité
  - 8-connexité : (2,0)-connexité
- En 3D
  - 6-connexité : (3,2)-connexité
  - 18-connexité : (3,1)-connexité
  - 26-connexité : (3,0)-connexité

THÉORÈME DE JORDAN

- (2,0)-connexité pour l'objet et le fond : une seule composante connexe du fond
- (2,1)-connexité pour l'objet et le fond : deux composantes connexes du fond, mais pas de courbe fermée simple

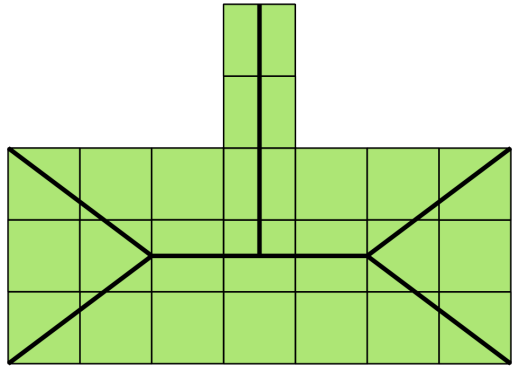


COUPLES DE CONNEXITÉS

- Seuls certains couples de connexités permettent de respecter le théorème de Jordan:
  - la $(D, D-1)$-connexité doit être présente (Malandain)
  - en général, l'autre connexité est la $(D, 0)$-connexité
- En 3D
  - (3,0), (3,2) $\Leftrightarrow$ 26-, 6-connexité
  - (3,1), (3,2) $\Leftrightarrow$ 18-, 6-connexité
  - (3,2), (3,0) $\Leftrightarrow$ 6-, 26-connexité
  - (3,2), (3,1) $\Leftrightarrow$ 6-, 18-connexité

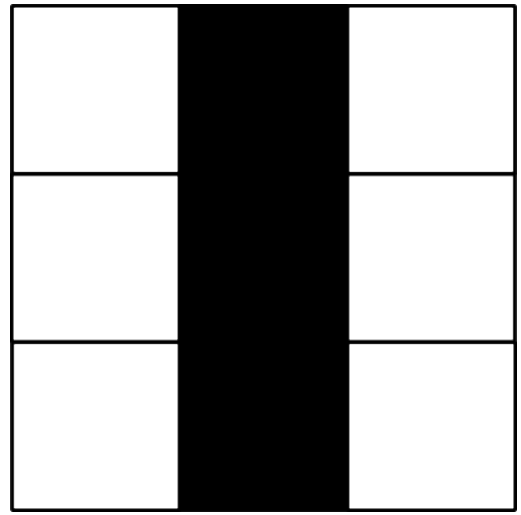
SQUELETTE

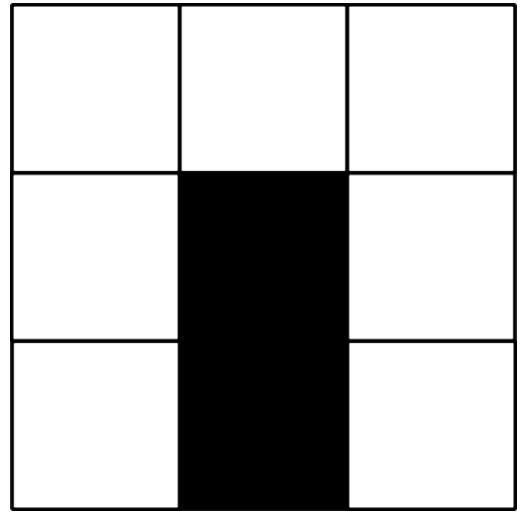
- Définition continue inutilisable
  - Centres des cercles non-situés sur des voxels
- Utilisation d'une définition algorithmique
  - Suppression (parallèle ou séquentielle) de points
    - Sans altérer la topologie de l'objet (points simples)
    - En conservant certaines caractéristiques géométriques



NOMBRES TOPOLOGIQUES

- Caractérisation locale de la topologie (Bertrand, Malandain)
- Compte des composantes connexes
  - Dans un voisinage
  - De l'objet et du fond
- $$T_{(3,0)}(p) = \sharp C^{p}_{(3,0)}\left(N^{*}_{(3,0)}(p)\right)$$
  $$T_{(3,2)}(p) = \sharp C^{p}_{(3,2)}\left(N^{*}_{(3,1)}(p)\right)$$
  - $\sharp C$: nombre de composantes connexes
  - N^{*} : voisinage strict (p exclus)



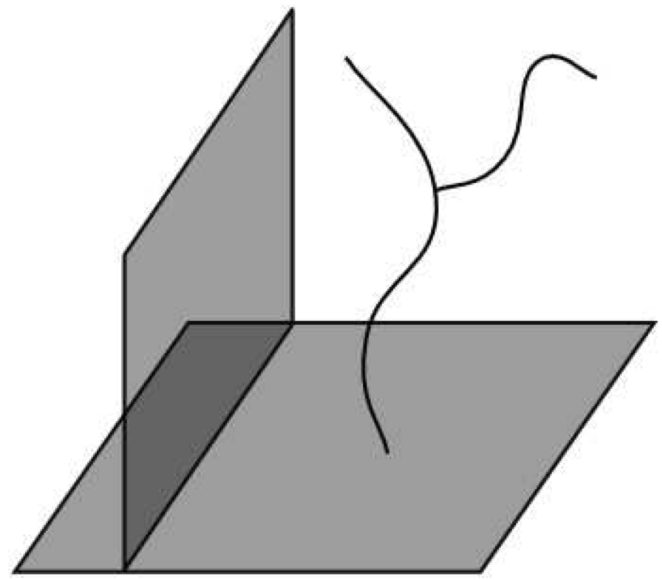


NOMBRES TOPOLOGIQUES

- Définition similaire pour le cas 2D
- $T_{(3,2)}(p) = \sharp C^p_{(3,2)}\left(N^*_{(3,1)}(p)\right)$ utilise les composantes du $(3,1)$-voisinage
  - Le (3,2)-voisinage strict n'est pas (3,2)-connexe
  - Idem en 2D pour $T_{(2,1)}$
  - Notion d'une connexité associée pour le calcul des nombres topologiques, similaire à la connexité du fond

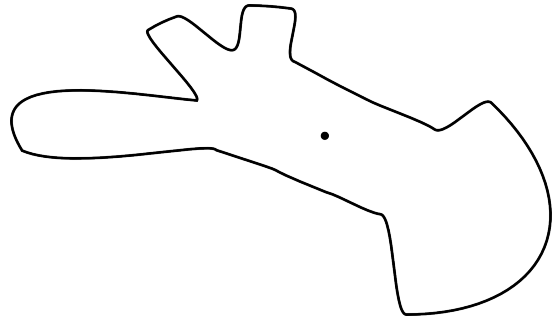
NOMBRES TOPOLOGIQUES

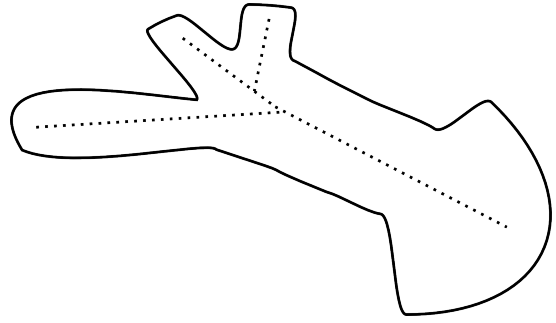
- Caractérisation de la topologie par $(T, \overline{T})$
- $\overline{T}$: nombre topologique par rapport au fond
  - Intérieur : $(T, \overline{T}) = (x, 0)$
  - Isolé : $(T, \overline{T}) = (0, x)$
  - Courbe : $(T, \overline{T}) = (2, 1)$
  - Surface : $(T, \overline{T}) = (1, 2)$
  - Simple : $(T, \overline{T}) = (1, 1)$

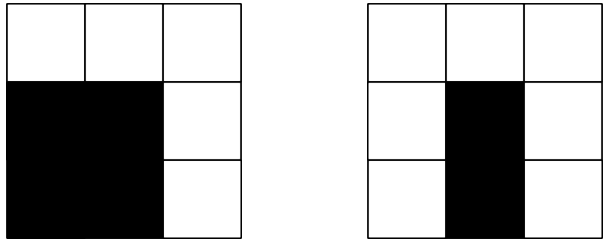


NOYAU HOMOTOPIQUE & SQUELETTE

- Noyau homotopique
  - Suppression de tous les points simples
  - Ne conserve que la topologie
- Squelette
  - Suppression des points simples non-extrémités
  - Point extrémité : un seul voisin dans l'objet

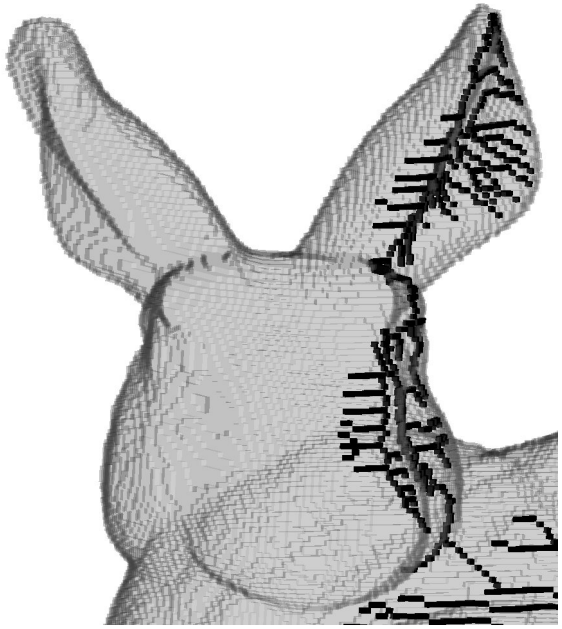


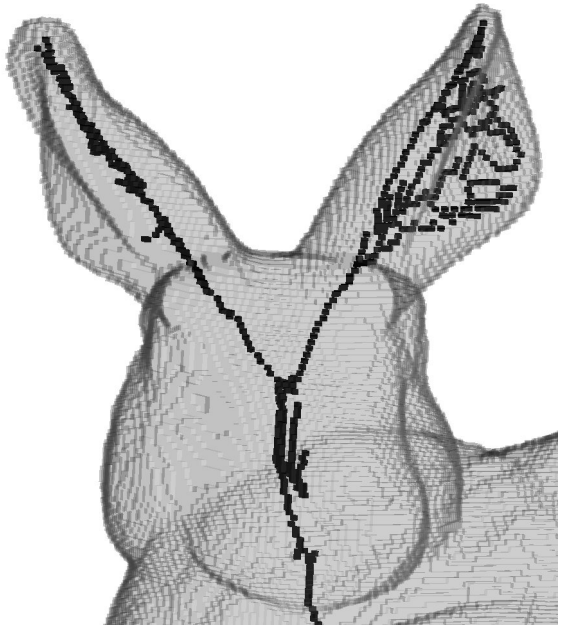




ORDRE DE SUPPRESSION DES POINTS

- Position du squelette : dépendant de l'ordre de suppression des points
  - Ordre de balayage : squelette non-centré
  - Transformée de distance : squelette centré





3. Implémentation

Connexité
Nombres topologiques
Squelettisation

Implémentation

- Voisinage, connexité
  - Pas d'accès aux pixels/voxels
  - Indépendant de la bibliothèque
- Nombres topologiques, squelettisation
  - Dépendant de la structure d'image
  - Sémantique d'ITK

CONNEXITÉ

- (D,k)-connexité
  - Générique selon D et k
  - Constant et sans initialisation
  - Instance unique
- Implémentation
  - Template
  - Singleton

CONNEXITÉ

```
template<unsigned int D, unsigned int k>
class Connectivity {
public :
    static Connectivity<D, k> const & GetInstance();

    bool AreNeighbors(Point const & p1,
                      Point const & p2) const;
    bool IsInNeighborhood(Point const & p) const;
    Point const * const GetNeighborsPoints() const;

private :
    Point * const m_NeighborsPoints;

    Connectivity();
    static int computeNumberOfNeighbors();
};
```

CONNEXITÉ DU FOND

- Permet d'automatiser le choix de la connexité du fond
- Paramètre par défaut des algorithmes
- La (D,D-1)-connexité doit être utilisée soit pour l'objet soit pour le fond
  - Si ce n'est pas la connexité de l'objet : (D,D-1) pour le fond
  - Sinon : choix arbitraire de (D,0) pour le fond

CONNEXITÉ DU FOND

- Résultat obtenu par spécialisation partielle
- Utilisation :

```
typedef Connectivity<3,2> FGC;
typedef BackgroundConnectivity<FGC>::Type BGC;
```

- FGC : foreground connectivity, BGC : background connectivity

CONNEXITÉ DU FOND

- Connexité $\neq$ (D,D-1) :

```
template<typename FGC>
struct BackgroundConnectivity
{
  typedef Connectivity<FGC::D, FGC::D-1> Type;
};
```

CONNEXITÉ DU FOND

- Connexité=(D,D-1) : on voudrait écrire

```
template<unsigned int D>
struct BackgroundConnectivity<
    Connectivity<D, D-1> >
{
    typedef Connectivity<D, 0> Type;
};
```

- Limitation du langage : interdit d'avoir un argument template (D-1) dépendant d'un argument de spécialisation (D)

CONNEXITÉ DU FOND

- Déclaration explicite pour chaque dimension

```
template<>
struct BackgroundConnectivity<Connectivity<2,1> >
{
  typedef Connectivity<2, 0> Type;
};

template<>
struct BackgroundConnectivity<Connectivity<3,2> >
{
  typedef Connectivity<3, 0> Type;
};
```

- Possibilité d'utiliser le pré-processeur (e.g. Boost.Preprocessor)

NOMBRES TOPOLOGIQUES

- Utilisation de itk::ImageFunction
  - Objet-fonction intégré au reste du pipeline
  - Classe template paramétrée par le type d'image et le type du résultat de la fonction
  - Fonction membre EvaluateAtIndex(Index)
- Classe paramétrée par la connexité de l'objet et du fond
- Compte les composantes connexes
  - Dans un voisinage dépendant de la connexité
  - Par un algorithme de propagation
- Déclaration :

```
template<typename Image, typename FGC,
    typename BGC=BackgroundConnectivity<FGC>::Type>
class TopologicalNumbers :
public itk::ImageFunction<Image, std::pair<int,int> >;
```

NOMBRES TOPOLOGIQUES

- Voisinage spécifique à utiliser selon la connexité

```
template<typename Connectivity>
struct NeighborhoodConnectivity
{
  typedef Connectivity Type;
};
```

- Spécialisations explicites pour les (D,D-1)-connexités
  - (2,0)-connexité pour la (2,1)-connexité
  - (3,1)-connexité pour la (3,2)-connexité

```
template<>
struct NeighborhoodConnectivity<Connectivity<3,2> >
{
  typedef Connectivity<3,2> Type;
};
```

NOMBRES TOPOLOGIQUES

- Calcul des composantes connexes par propagation
  - Pré-calcul des paires de points (p_1, p_2) telles que :
    - p_1 et p_2 sont adjacents (connexité « de base »)
    - p_1 est adjacent (connexité du voisinage) à l'origine
    - $p_1 + p_2$ est contenu dans un cube $3 \times 3 \times 3$ centré à l'origine
  - Propagation dans un cube $3 \times 3 \times 3$ centré à l'origine
- Classe paramétrée par la connexité « de base » et celle du voisinage

```
template<typename C,
  typename NC =
    typename NeighborhoodConnectivity<C>::Type>
class UnitCubeNeighbors
{
public :
  bool operator()(Point p1, Point p2) const;
};
```

NOMBRES TOPOLOGIQUES

- Calcul des composantes connexes par propagation
  - Pré-calcul des paires de points (p_1, p_2) telles que :
    - p_1 et p_2 sont adjacents (connexité « de base »)
    - p_1 est adjacent (connexité du voisinage) à l'origine
    - $p_1 + p_2$ est contenu dans un cube $3 \times 3 \times 3$ centré à l'origine
  - Propagation dans un cube $3 \times 3 \times 3$ centré à l'origine
- Classe paramétrée par la connexité « de base » et celle du voisinage

```
template<typename C,
  typename NC =
    typename NeighborhoodConnectivity<C>::Type>
class UnitCubeCCCounter
{
public :
  unsigned int operator()() const;
};
```

NOMBRES TOPOLOGIQUES

```
template<typename TImage, typename FGC,
  typename BGC =
    typename BackgroundConnectivity<BGC>::Type>
class TopologicalNumbers :
  public itk::ImageFunction<TImage,
    std::pair<unsigned int, unsigned int> >
{
public :
  std::pair<unsigned int, unsigned int>
  EvaluateAtIndex(IndexType const & index) const;
private :
  static UnitCubeCCCounter<FGC> foregroundCCCounter;
  static UnitCubeCCCounter<BGC> backgroundCCCounter;
};
```

SIMPLICITÉ

- Hérite de itk::ImageFunction
- Paramétrée par les connexités de l'objet et du fond
- Déclaration :

```
template<typename Image, typename FGC,
  typename BGC=typename
    BackgroundConnectivity<FGC>::Type> >
class Simplicity :
   public itk::ImageFunction<Image, bool>
{
public :
  bool EvaluateAtIndex(Index const & index) const;
private :
  typename TopologicalNumbers<TImage, FGC> tnCounter;
};
```

NOYAU HOMOTOPIQUE

- Filtre paramétré par le type d'image et la connexité
- Accepte d'autres critères de simplicité que les nombres topologiques
- Déclaration :

```
template<typename Image, typename FGC>
class HomotopicKernel :
  public InPlaceImageFilter<Image>
{
public :
  typedef ImageFunction<Image, bool> SimplicityCriterion;

  SimplicityCriterion GetSimplicityCriterion() const;
  void SetSimplicityCriterion(SimplicityCriterion);
private :
  typename SimplicityCriterion simplicityCriterion;
};
```

NOYAU HOMOTOPIQUE

```
while(!q.IsEmpty()) {
  Index const p = q.GetFront();

  if(simplicityCriterion->EvaluateAtIndex(p)) {
    outputImage->SetPixel(p, 0);

    // Add neighbors that are not already in the queue
    for(int i=0; i<fgc.GetNumberOfNeighbors(); ++i) {
      Index n = p+fgc.GetNeighborsPoints()[i];

      if(/* neighbor is in image */
         outputImage->GetPixel(n) != 0 &&
         /*and has not 0 priority*/
         orderingImage->GetPixel(n) != 0 ) {
        q.Push(orderingImage->GetPixel(n), n);
} } } }
```

TERMINALITÉ & SQUELETTISATION

- Principe similaire à la simplicité et au noyau homotopique
- Terminalité

```
template<typename Image, typename FGC,
  typename BGC=typename
    BackgroundConnectivity<FGC>::Type> >
class Terminality;
```

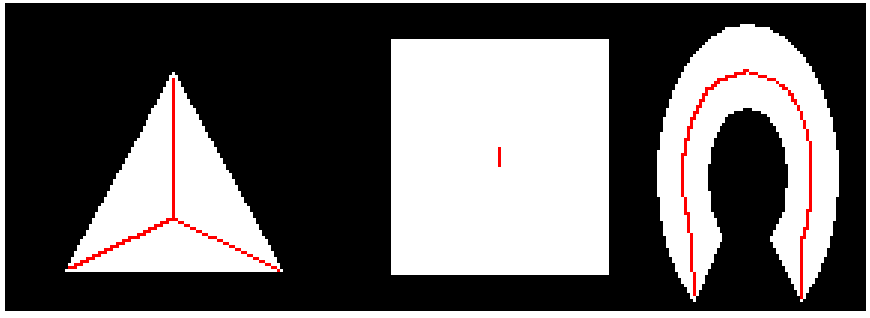
- Squelettisation
  - Similaire au filtre de noyau homotopique
  - Ajoute un critère de terminalité

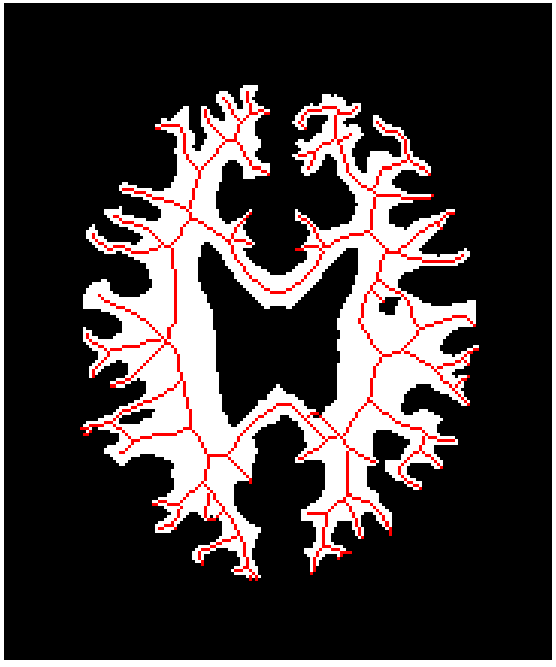
TERMINALITÉ & SQUELETTISATION

```
while(!q.IsEmpty()) {
  Index const p = q.GetFront();

  if(simplicityCriterion->EvaluateAtIndex(p) &&
     terminalityCriterion->EvaluateAtIndex(p)) {
    outputImage->SetPixel(p, 0);

    // Add neighbors that are not already in the queue
    for(int i=0; i<fgc.GetNumberOfNeighbors(); ++i) {
      Index n = p+fgc.GetNeighborsPoints()[i];

      if(/* neighbor is in image */
         outputImage->GetPixel(n) != 0 &&
         /*and has not 0 priority*/
         orderingImage->GetPixel(n) != 0 ) {
        q.Push(orderingImage->GetPixel(n), n);
} } } }
```

RÉSULTATS

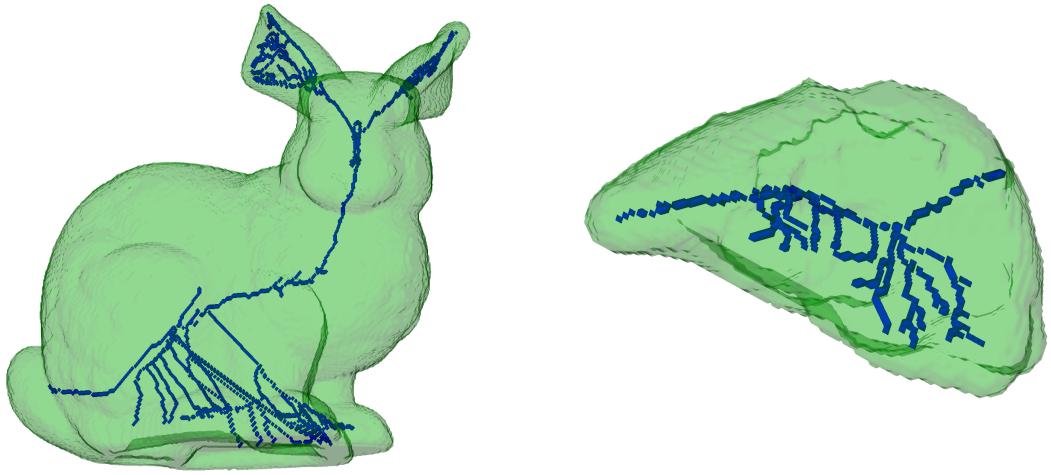
- 2D :





RÉSULTATS

- 3D :



CONCLUSION

- Il est possible d'intégrer les notions fondamentales de topologie discrète à une bibliothèque de traitement d'images
  - De façon générique par rapport à la dimension de l'image (modulo quelques instanciations implicites)
  - De façon non-intrusive (pas de modification de la structure d'image)
  - De façon quasi-indépendante de la bibliothèque (adapter en fonction de la sémantique d'accès)

Merci de votre attention !