Synthèse d'images - modélisation géométrique

R. Raffin

romain.raffin@univ-amu.fr

CM/TD/TP dispos sur Ametice

- Plan de ce module 3CM 3TD 3TP + 2TPA
  - introduction
  - représentations des objets
  - courbes et surfaces
  - transformations & projections
  - Découpage
  -

Évaluations : 12/02 contrôle théorique (2 h) + ramassage de TP

Tp : C++/OpenGL

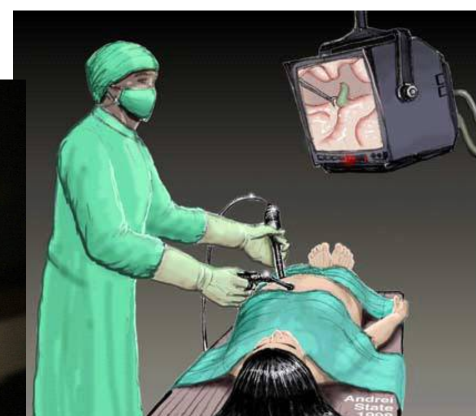
- suite = Archi IN
  - programmation OpenGL avancée
  - programmation du pipeline graphique (shaders)
  - tp C++/OpenGL/GLSL

- Plan du module
  - introduction
  - représentations des objets
  - courbes et surfaces
  - transformations & projections
  - découpage

Quels sont les besoins en Image de synthèse ?

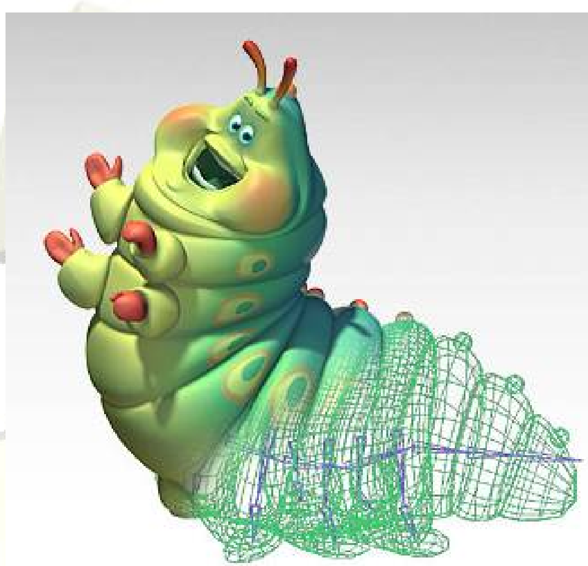
- il existe de nombreux domaines d'application et pour chacun d'eux, nous allons voir que les besoins sont spécifiques.

- quelques exemples :
  - cinéma, animation, effets spéciaux, ...,
  - imagerie médicale,
  - construction (CAO, DAO) : aéronautique, automobile, ...,
  - simulation de phénomènes physiques ou naturels,
  - réalité virtuelle,
  - ...



Les objectifs de la modélisation

- doit s'attacher aux contraintes (précision, qté de données, type de visualisation, ...)
- doit être intuitive et interactive, courbes 2D/3D, surfaces 3D ou volumes
- recherche une qualité visuelle
  - respect des proportions, d'un environnement « réaliste »
  - pour y parvenir, on introduira les notions de « continuité » :
    - de positions : C^0 (sommets des objets, textures)
    - de normales : C^1 (éclairage de base, détails et reliefs)
    - de courbure : C^2 (éclairage ++, reflets, modélisation ++)



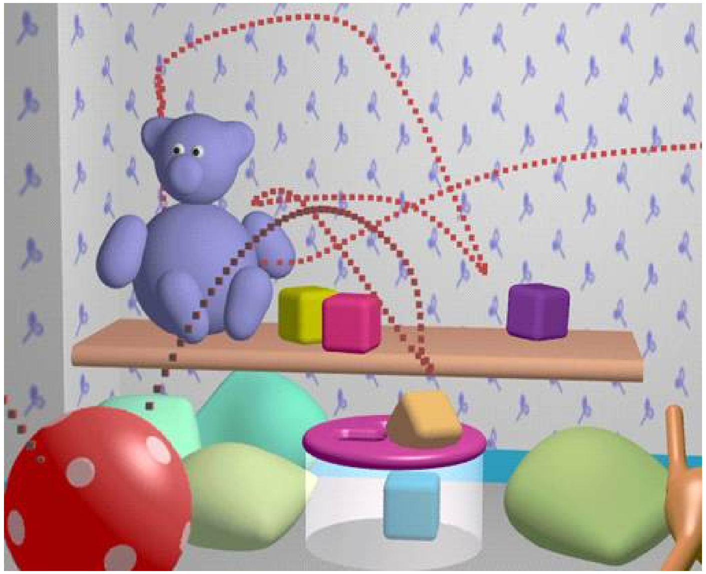
- modélisation de maquette : traduisible, découpable en morceaux, **adaptable**
- logiciels de constructions (CAO, métiers) : de la conception à la fabrication
  - représentation mathématique de la surface qui doit être précise et adaptée aux contraintes de fabrication et aux métiers (continuité, découpage, assemblage, câblage, publicité, …)
  - permet des tests pré-production (aérodynamique, résistance des matériaux, sécurité, confort, bruit, dangerosité, …)
- ...

- Plan du module
  - ~~introduction~~
  - représentations des objets
  - courbes et surfaces
  - transformations & projections
  - découpage

Donner une représentation géométrique des objets 3D

- **points** (sommets) -> fixent une dimension (x, y, z, ...)
- **lignes**, **polygones** -> fermés, ouverts, bordures, frontières
- **facettes** triangulaires ou non -> plans, normales
- **équations** mathématiques -> courbes, surfaces, dérivées
- **boules**, ellipsoïdes, **cubes** -> assemblage, volumes
- **représentations hybrides** -> parce qu'on ne sait pas tout faire
- …

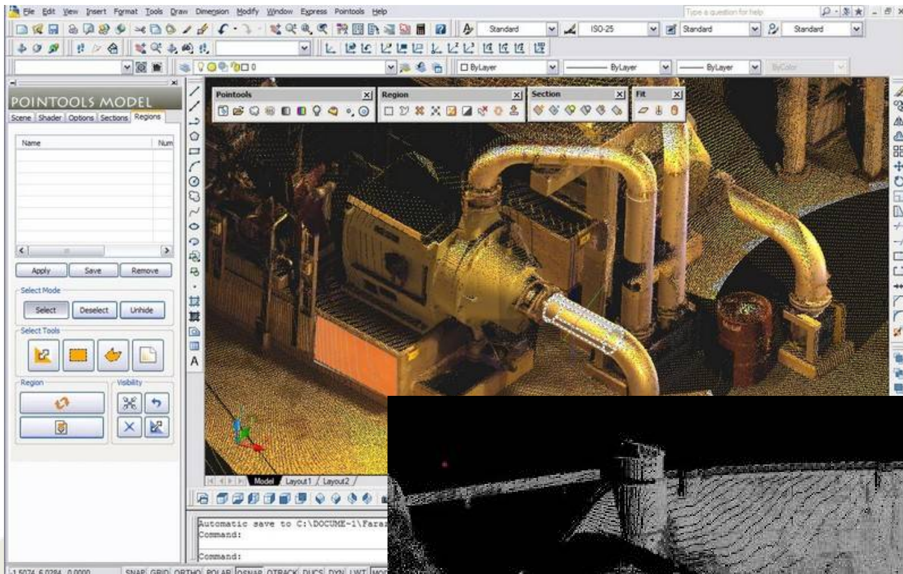
- Présentation des concepts fondamentaux
  - Modélisation
    - objets, scène, environnement,
    - comment représenter les objets ?
    - comment construire/choisir une représentation ?

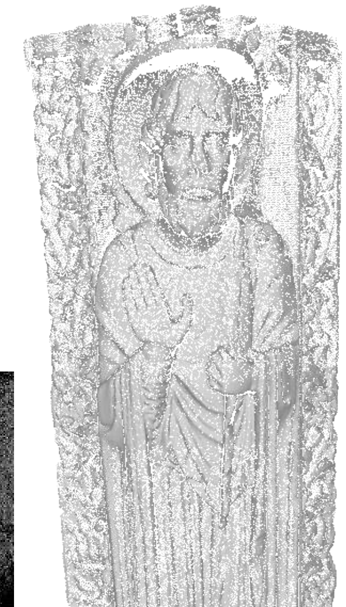


  - Animation
    - spécifier ou calculer positions, mouvements et déformations
  - Visualisation
    - objets, matières, environnement, éclairages, caméras, …

Représentations des objets / Les points

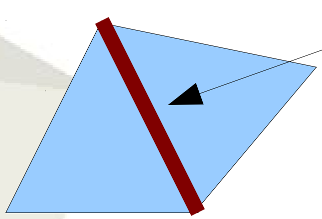
- C'est la primitive 3D la plus simple
- Un ensemble de points représente la <u>géométrie</u> de l'objet





Un ensemble de polygones (maillage) représente la géométrie + la <u>topologie</u> de l'objet

+ de « finesse » (smooth surface)= + de polygones

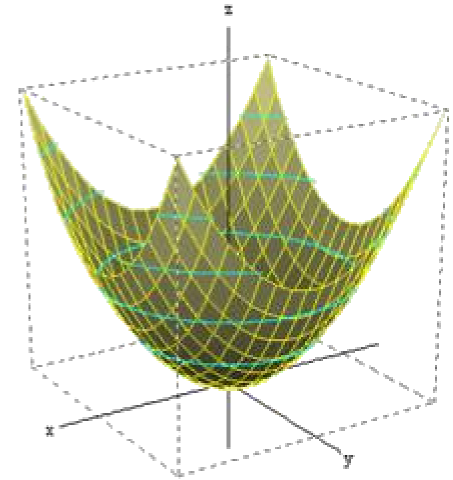


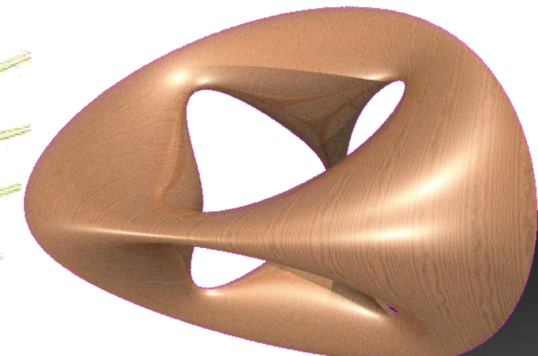
Topologie = Les arêtes et les sommets sont partagés par plusieurs faces

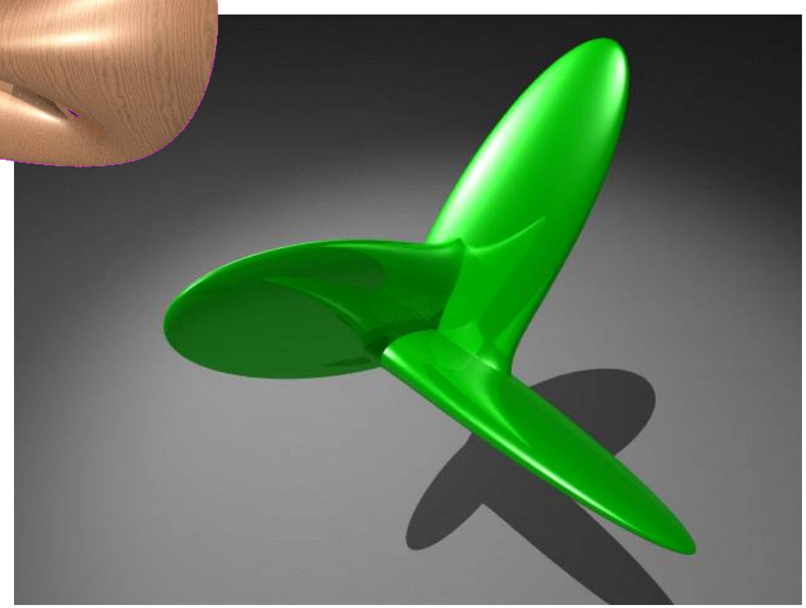
Les points connaissent leurs voisins

Les points et les voisinages sont donnés par une fonction mathématique : *f(a, b, c)* = ...

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 - z = 0$$

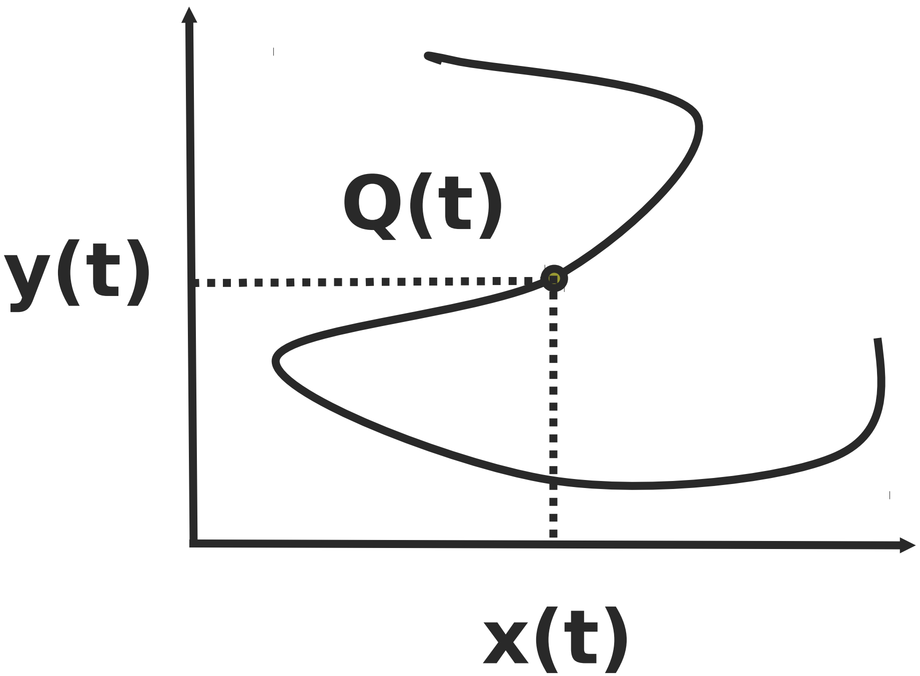






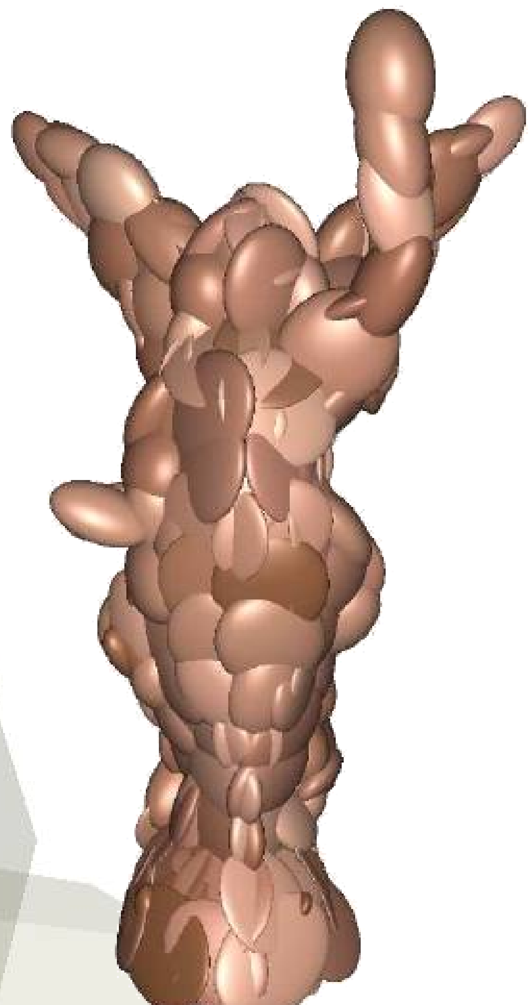
Les courbes paramétriques permettent de faire évoluer indépendamment x, y et z :

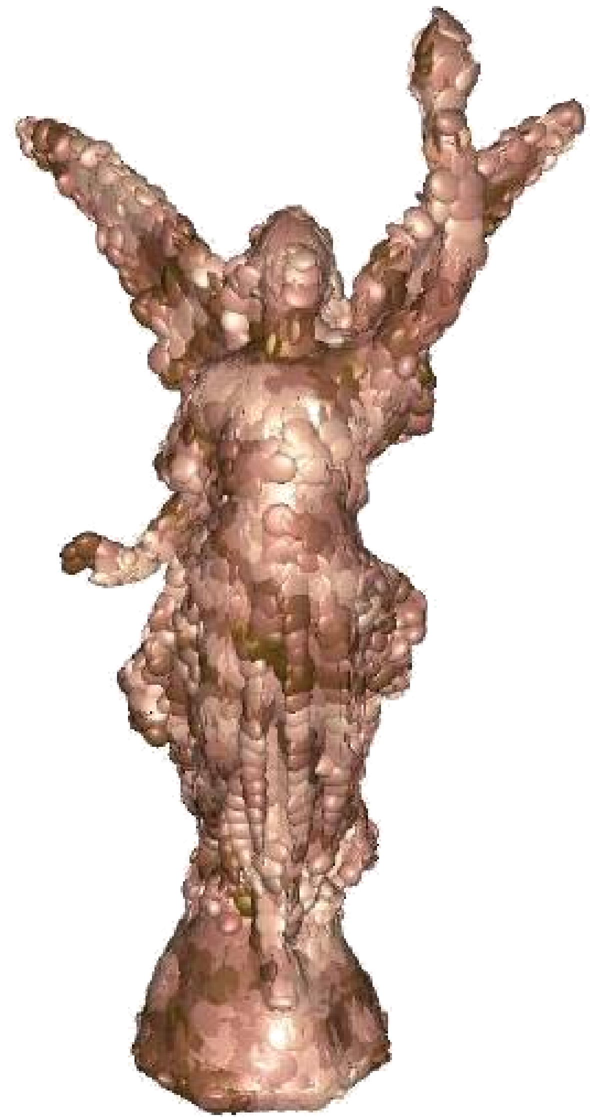
$$\begin{cases} Q : [a, b] \rightarrow \mathbb{R}^3 \\ t \mapsto Q(t) = (x(t), y(t), z(t)) \end{cases}$$



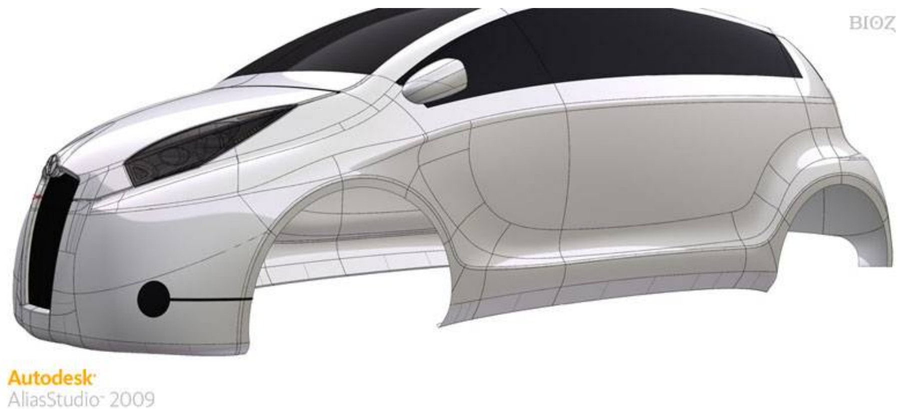
Elles sont également plus simples à « discrétiser » (en fonction du paramètre *t*).

… souvent utilisés sur des nuages de points, permettent une interprétation volumique

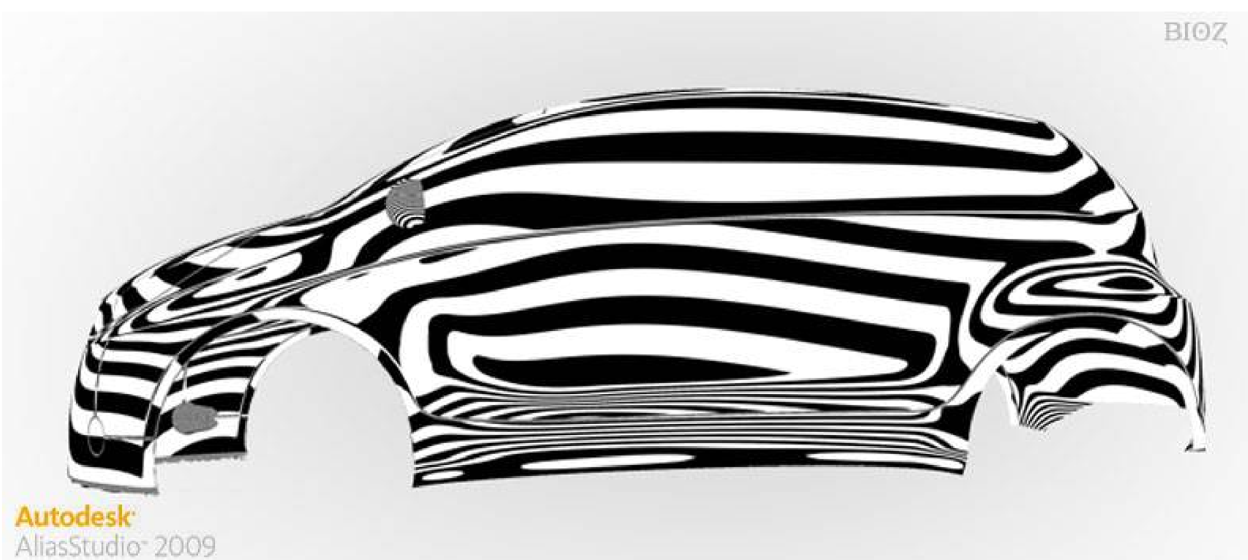




La continuité des positions, tangentes, normales, courbures est utile pour la modélisation, la visualisation, le rendu réaliste, ...

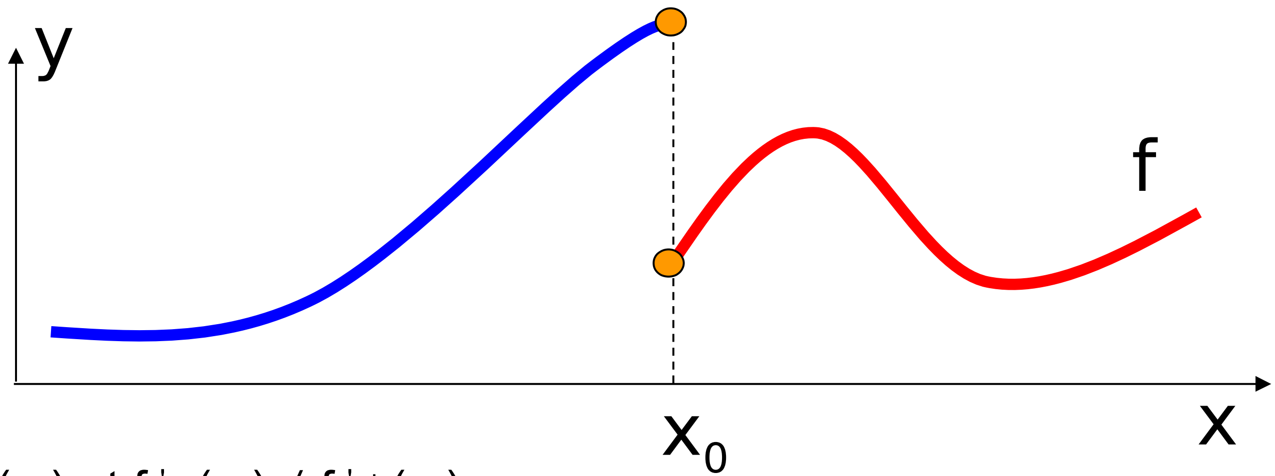


Thomas Battut
http://www.st-biozdesign.com/

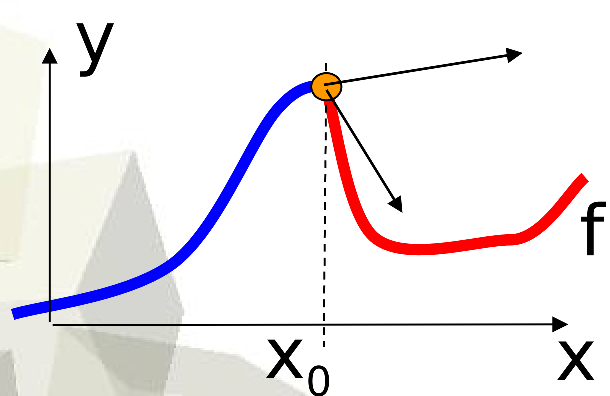


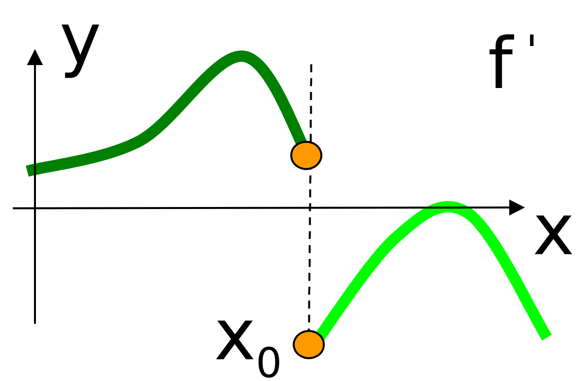
- Soit une fonction f : R→ R
  - Si en $x=x_0$ $f^-(x_0) \neq f^+(x_0)$ la courbe est discontinue en x_0

C^{-1}



  - Si $f^-(x_0) = f^+(x_0)$ et $f'^-(x_0) \neq f'^+(x_0)$



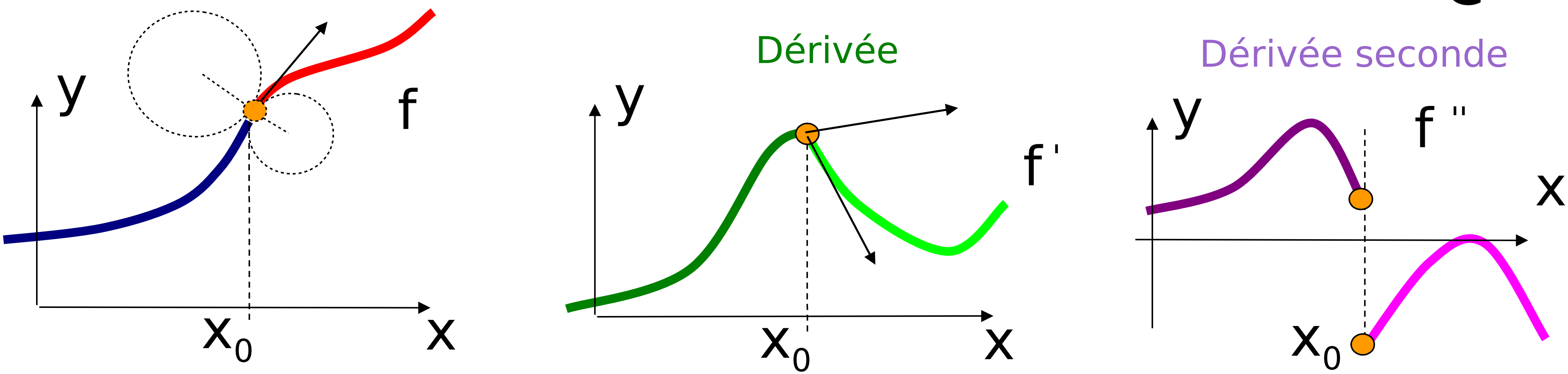


Dérivée

C^0

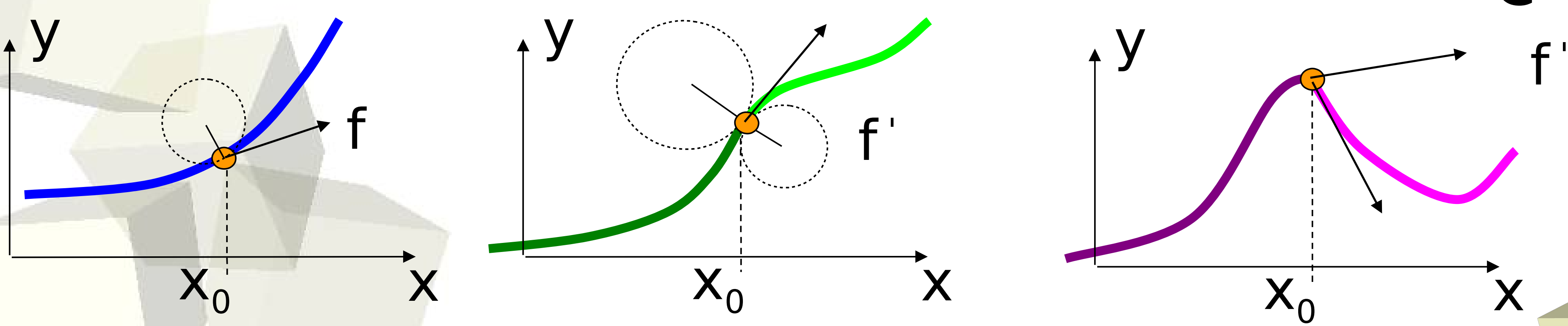
- Si $f^-(x_0) = f^+(x_0)$, $f'^-(x_0) = f'^+(x_0)$ et $\mathbf{f''^-(x_0) \neq f''^+(x_0)}$

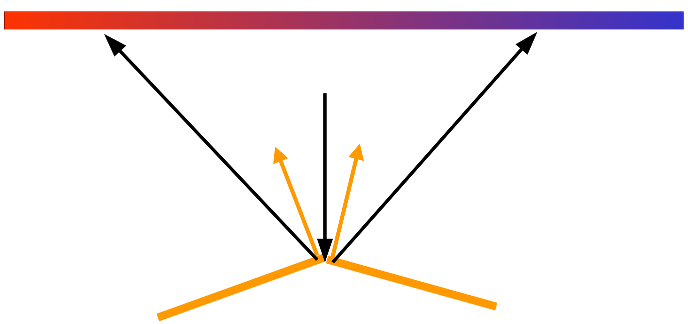
C¹



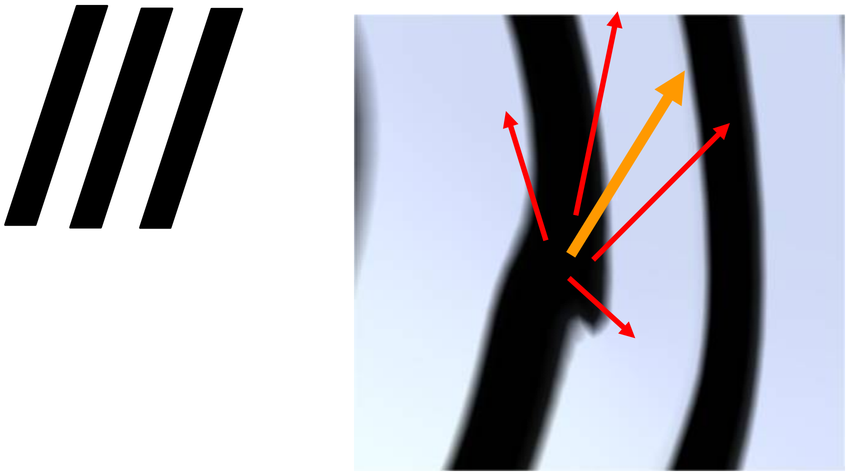
- Si $f^-(x_0) = f^+(x_0)$, $f'^-(x_0) = f'^+(x_0)$, $f''^-(x_0) = f''^+(x_0)$ et $\mathbf{f'''^-(x_0) \neq f'''^+(x_0)}$

C²





Surface C^0, reflet discontinu (C^1)



Surface C^1, reflet C^0: la courbure varie dans les différentes directions autour du point central

La continuité des reflets est égale à celle de la surface moins 1 (surface de continuité C^2 => reflets de continuité C^1).

- Pour cette raison, en animation, on souhaite produire des surfaces de continuité C^2 en tous points : pour que les reflets soient C^1

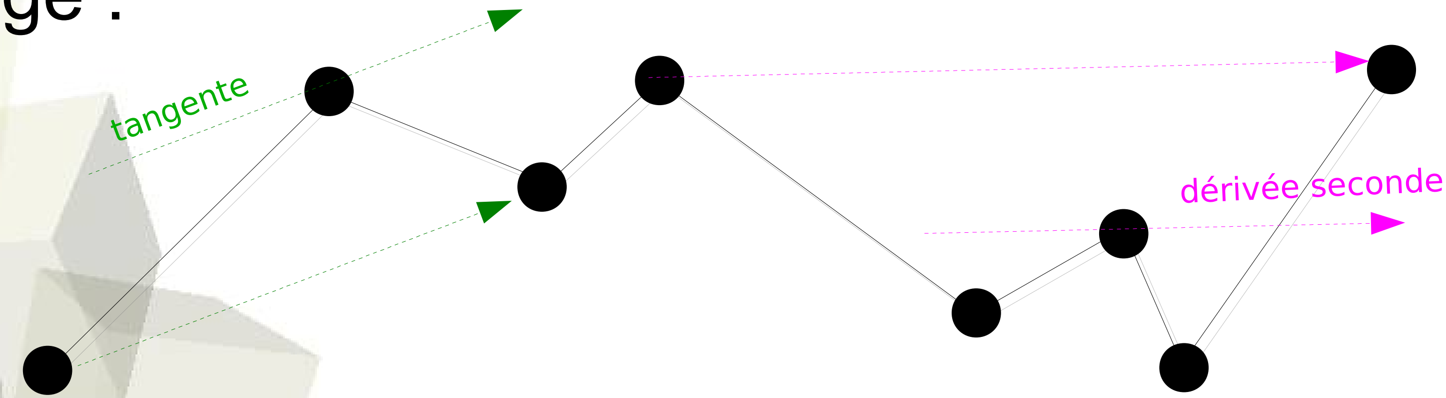

New Front
BIOZ
Autodesk
AliasStudio 2009

- les dérivées premières définissent les tangentes,
- les dérivées secondes les courbures

Ces définitions concernent surtout les courbes et surfaces de définition paramétrique.

En effet, comment calculer la dérivée d'un polygone ?

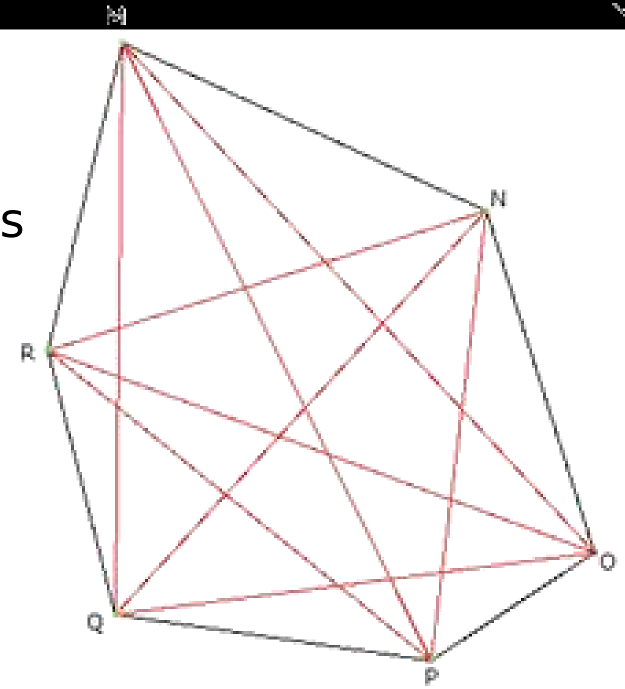
-> un calcul (très approximatif) souvent utilisé, à partir du voisinage :



Rappel, définition mathématique de la dérivée :
(f(x) - f(x0)) / (x - x0), quand x -> x0

Polygone convexe

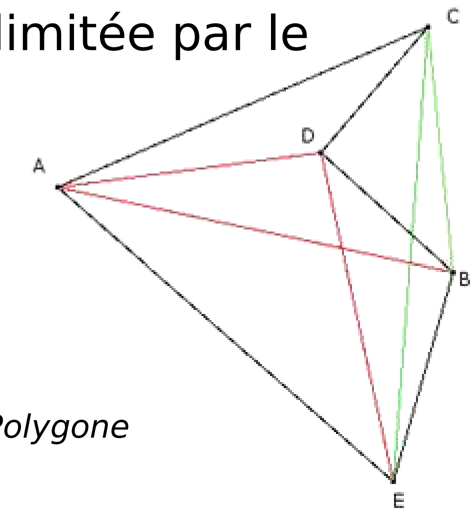
Un polygone est dit convexe s'il n'est pas croisé et si toutes ses diagonales sont entièrement à l'intérieur de la surface délimitée par le polygone. Ainsi, l'hexagone MNOPQR ci-contre (à droite) est dit convexe.



Polygone concave

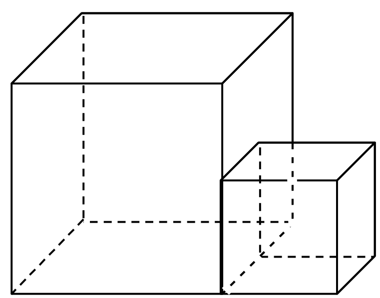
Un polygone est dit concave s'il n'est pas croisé et si l'une de ses diagonales n'est pas entièrement à l'intérieur de la surface délimitée par le polygone.

Par exemple, le pentagone ACDBE ci-contre (à droite) est dit concave car les diagonales [BC] et [CE] sont à l'extérieur de la surface délimitée par le polygone.

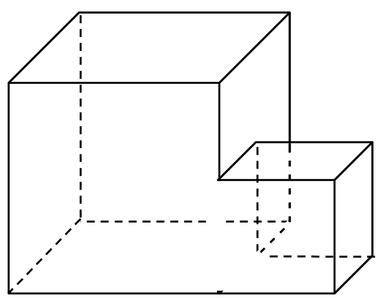


http://fr.wikipedia.org/wiki/Polygone

- Boundary-Representation
  - le modèle est représenté par ses bords
  - pas de notion de volume
  - on peut représenter des solides

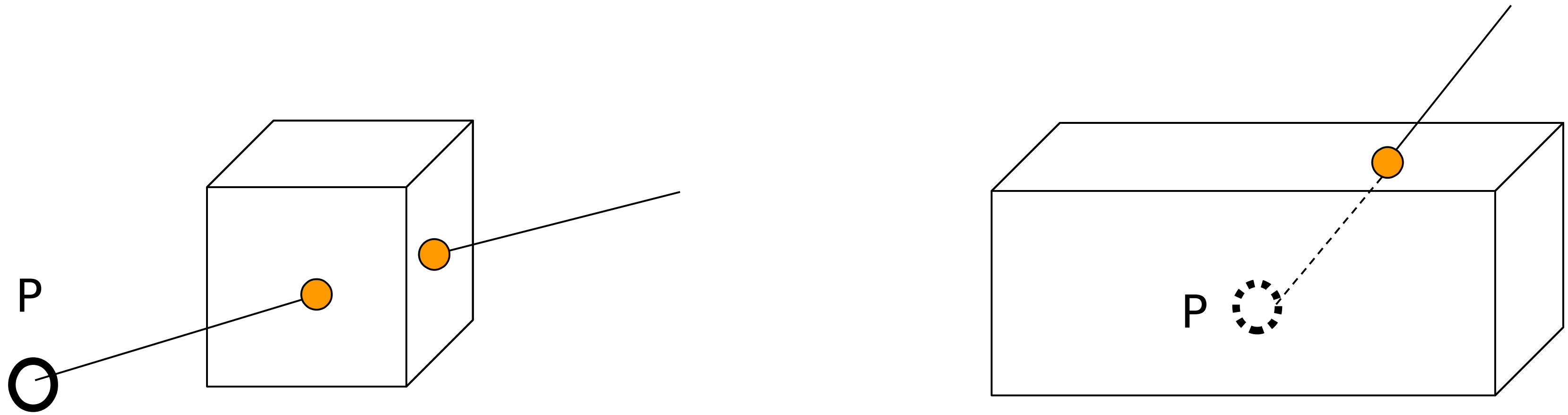


B-Reps quelconque



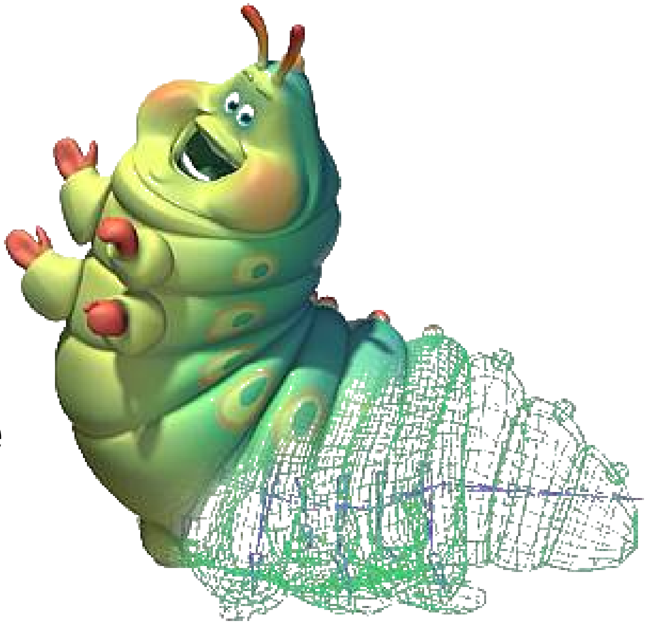
B-Rep solide

- Test d'intériorité d'un point dans un solide



- nombre **pair** d'intersections : le point P est à **l'extérieur** du solide
- nombre **impair** d'intersections : le point P est à **l'intérieur** du solide

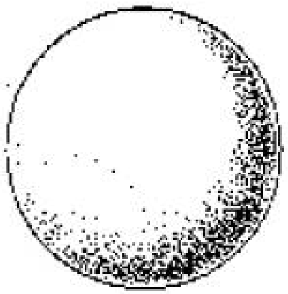
- c'est un complexe linéaire par morceaux : les surfaces sont représentées avec des polygones. Le face la plus simple est le triangle (3 points, plan),
- sa continuité globale est C^0 (discontinuité des tangentes aux sommets, des normales aux arêtes),
- il définit la géométrie tout en donnant une topologie de la surface (voisinages),
- c'est actuellement une structure standard pour afficher des scènes complexes en 3D,
- leur visualisation et leur manipulation sont optimisées par la grande majorité des cartes graphiques actuelles,



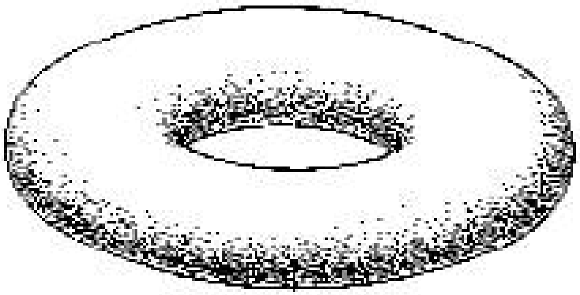
Formule d'Euler (1752)
Cette formule donne la correspondance entre le nombre de composant de chaque entité du maillage (sommets, arêtes, faces)

$$S - A + F = 2\,(1\text{-}g)$$

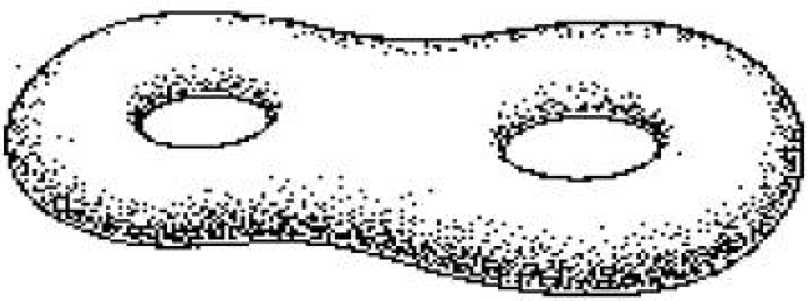
Où S est le nombre de sommets, A est le nombre d'arêtes et F est le nombre de faces. g est le genre (*genus*) de l'objet = le nombre de trous (poignées) dans un objet fermé



genre 0



genre 1

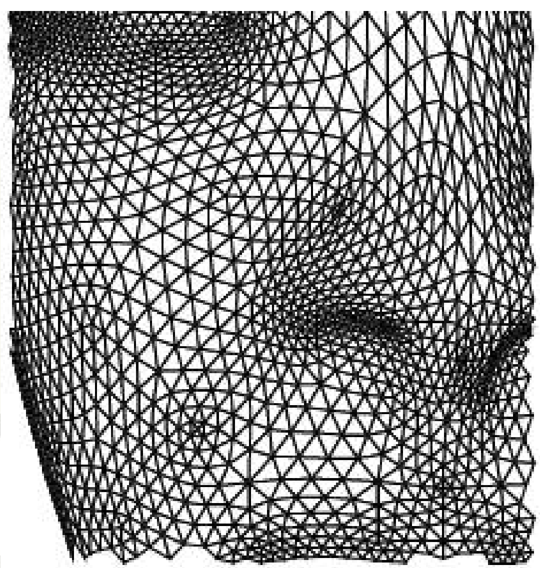


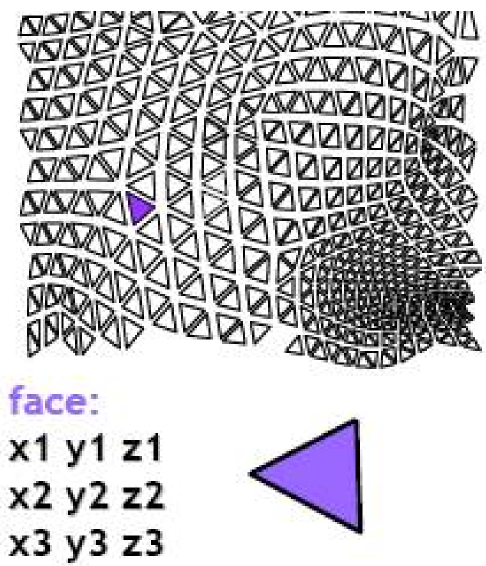
genre 2

Choisir une représentation adaptée aux opérations que l'on souhaite effectuer sur le maillage :

- Une représentation compacte (taille mémoire)
- Une représentation optimisée pour le parcours

- pour chaque triangle, on donne la liste de sommets
  - c'est l'approche la plus naïve
  - 4 octets par coordonnée (un flottant)
  - 9 coordonnées par faces
  - 2 fois plus de faces que de sommets
- pour un maillage composé de S sommets, on a donc besoin de 4*9*2*S = 72*S octets.





F0 : x_0,y_0,z_0 x_1,y_1,z_1 x_2,y_2,z_2

F1 : x_5,y_5,z_5 x_1,y_1,z_1 x_{12},y_{12},z_{12}

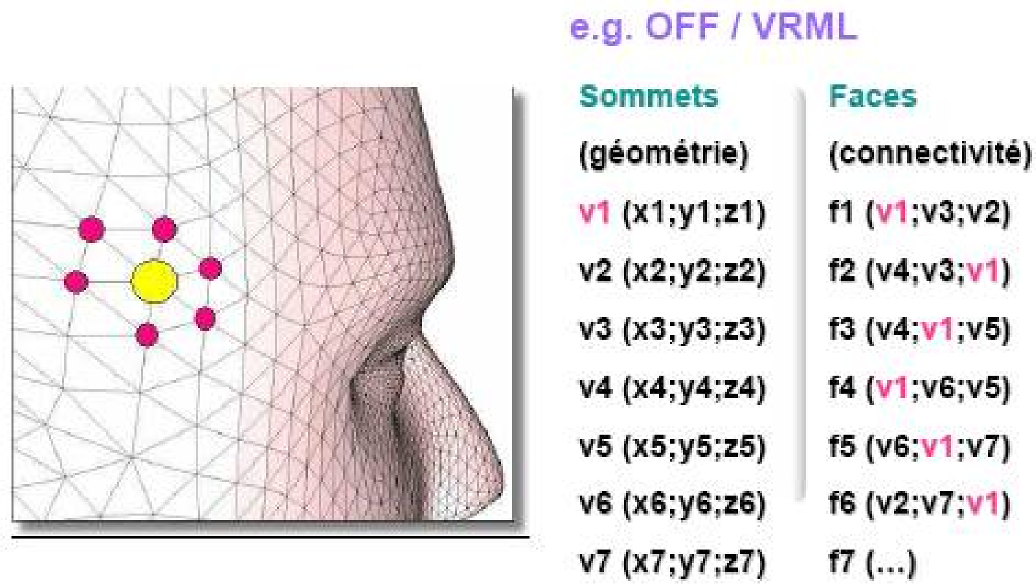
F2 : x_9,y_9,z_9 x_0,y_0,z_0 $x_{1025},y_{1025},z_{1025}$

∴ **Duplication des sommets**

- tout d'abord on stocke la liste des sommets
- puis les facettes sont définies en donnant les index des sommets qui la compose

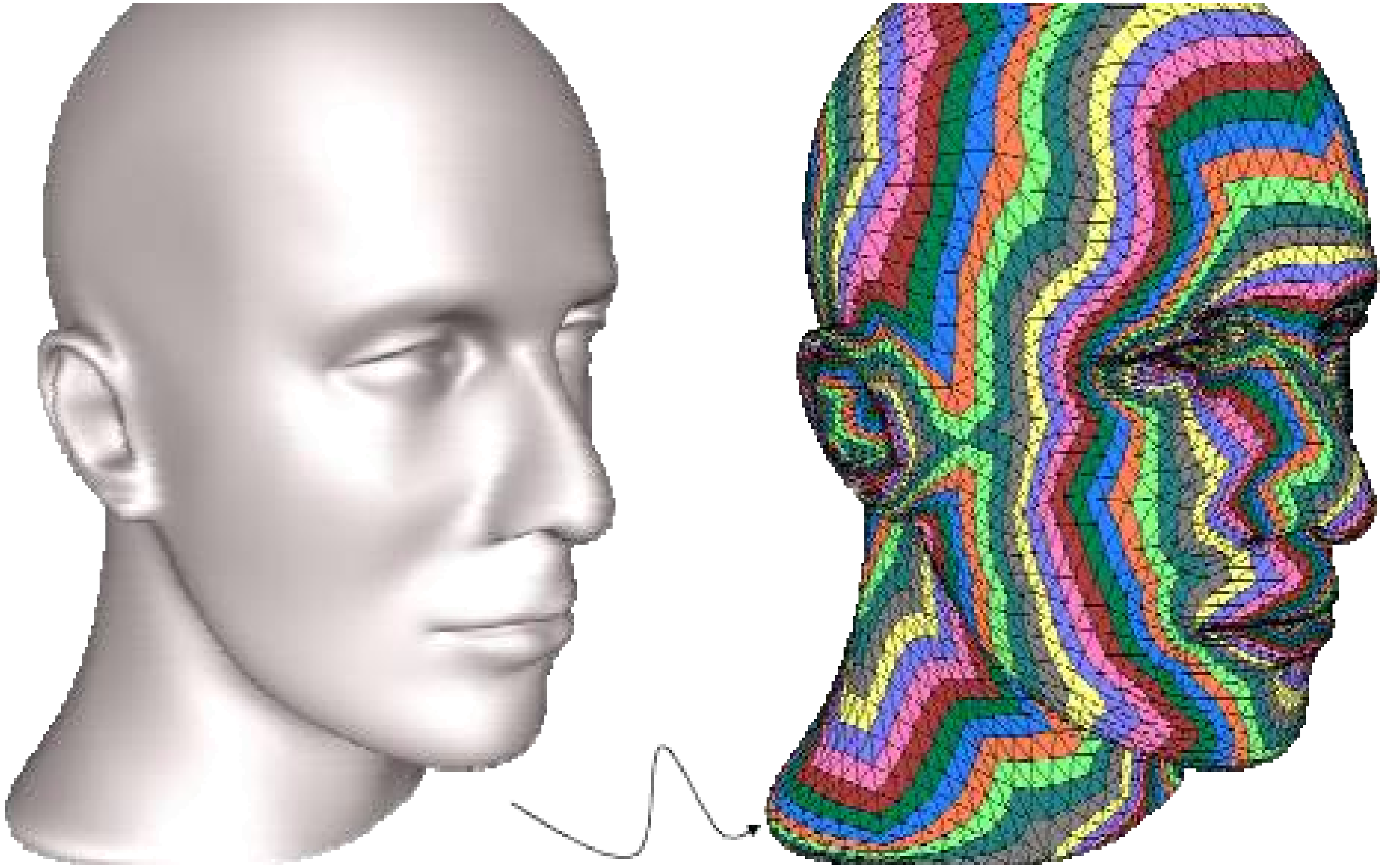
Liste des sommets	Liste des facettes
x_0,y_0,z_0	0 1 2
x_1,y_1,z_1	5 1 12
x_2,y_2,z_2	9 0 1025
…	…

Combien d'octets par sommet? (36)



en général, le fichier commence par le nombre de sommets et le nombre de facettes

La topologie est codée de façon implicite dans l'ordre des sommets



-> accélération matérielle !!

Construction de l'objet

- sa liste de sommets GLfloat sommets[] = {x_0, y_0, z_0,

  …

  x_n, y_n, z_n};

- ses attributs, par exemples les normales

  GLfloat normales[] = {n_{x0}, n_{y0}, n_{z0},

  …

  n_{xn}, n_{yn}, n_{zn}};

- sa liste d'index GLuint index[] = {0, 1, 2,

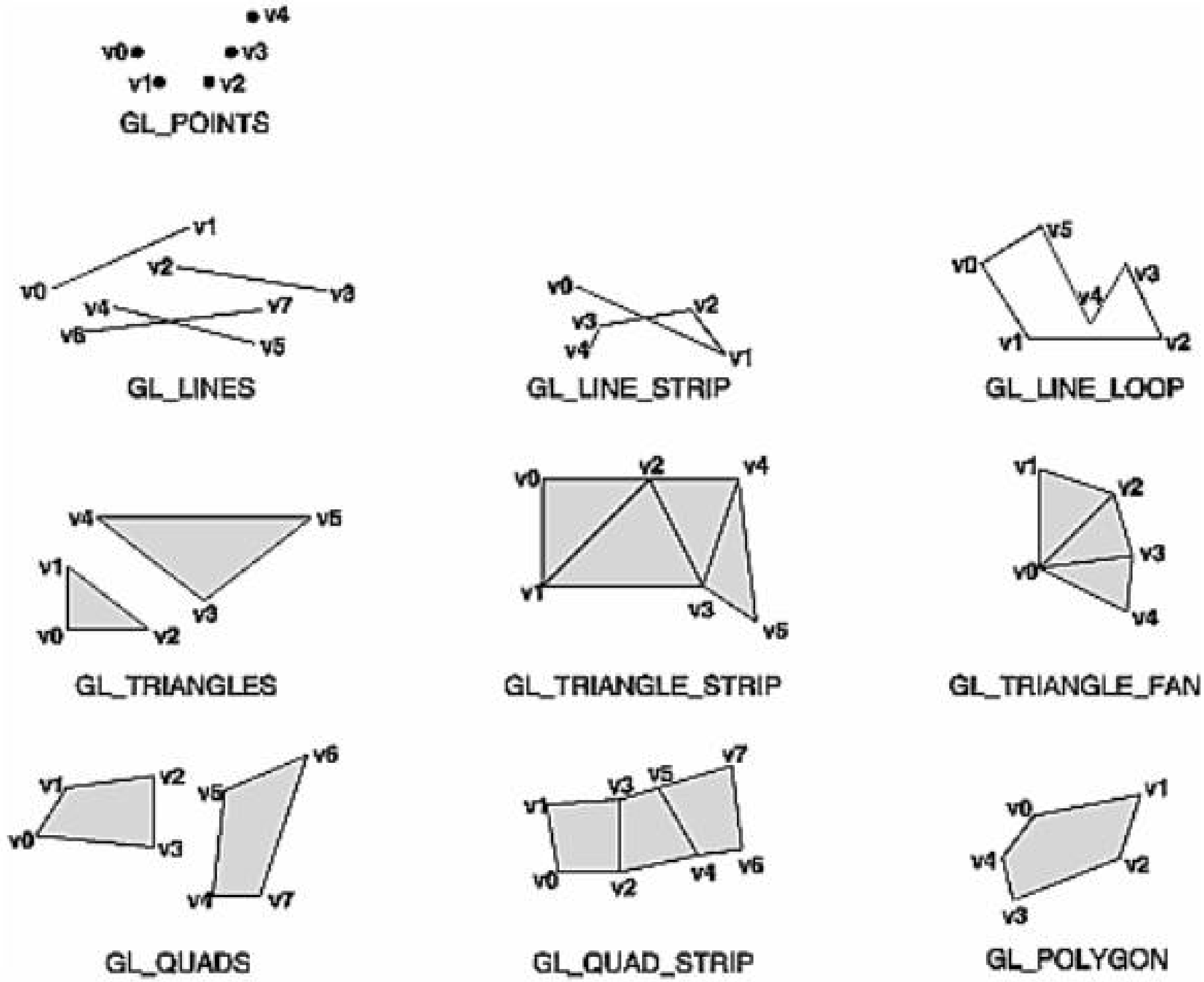
  5, 0, 4,

  . . . };

Types de représentation pour OpenGL :

- GL_TRIANGLES : il faut donner les index des facettes triangulaires,
- GL_QUADS : il faut donner les index des quadrilatères,
- GL_TRIANGLE_STRIP : la liste ordonnée des index pointant sur les sommets des bandes de triangles. Dans le tableau d’index, il est possible d’utiliser un séparateur pour séparer les différentes bandes de triangles,
- GL_TRIANGLE_FAN, GL_QUAD_STRIP,
- …

Représentations des objets / Visualisation OpenGL d'un maillage



A la main, face par face (déprécié)

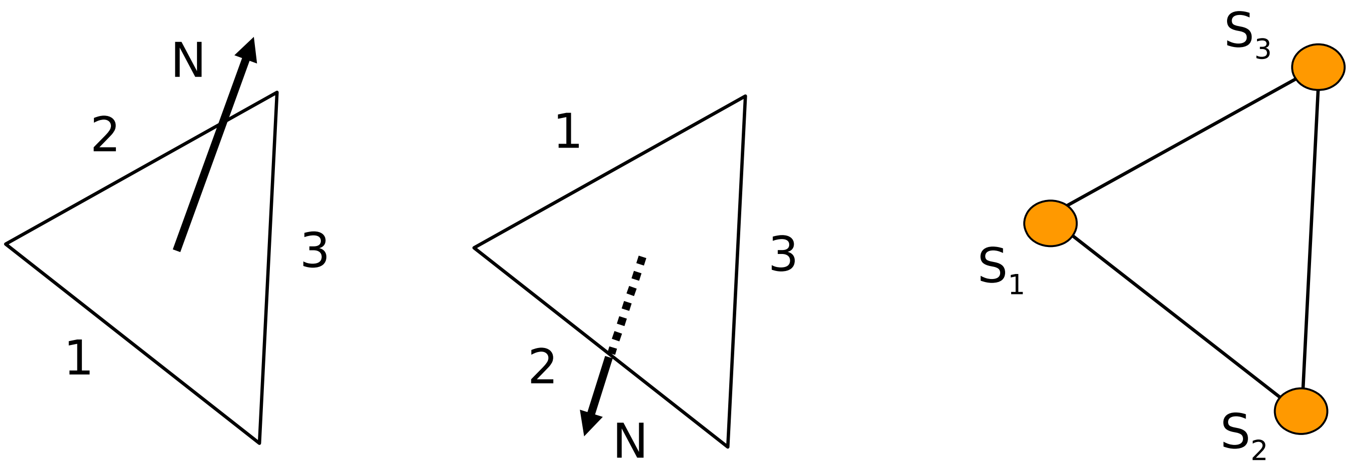
```
glBegin(GL_TRIANGLES)
        glVertex3f(0.0, 0.0, 0.0);
        glVertex3f(0.0, 0.0, 0.0);
        glVertex3f(0.0, 0.0, 0.0);
glEnd();
```

+ compliqué, sur un tableau de données

```
// Activation du mode de tracé par tableaux de sommets
glEnableClientState (GL_VERTEX_ARRAY);
glEnableClientState (GL_NORMAL_ARRAY);  // si on utilise les normales
glEnableClientState (...);  // si on utilise d'autres attributs (couleur, coordonnées texture, ...)

// Affectation des différents tableaux
  glVertexPointer (3, GL_FLOAT, 0, sommets);
glNormalPointer (GL_FLOAT, 0, normales);  // si on utilise les normales
// … + autres tableaux s'il y a (couleur,…)
// Tracé du maillage triangulaire
  glDrawElements (GL_TRIANGLES, nb_index, GL_UNSIGNED_INT, index);
//  Désactivation du mode tracé
glDisableClientState ( ………..);  // un pour chaque glEnableClientState
```

la normale a une facette est obtenu en faisant le produit vectoriel de sa première arête avec la deuxième.



$$N = \frac{(S_2 - S_1) \wedge (S_3 - S_1)}{\left\| (S_2 - S_1) \wedge (S_3 - S_1) \right\|}$$

attention à l'orientation des normales quand on stocke un maillage => les sommets des faces doivent toujours être stockés dans le même sens.

les normales donnent l'orientation vers l'extérieur d'un objet fermé. Elles sont aussi fondamentales pour l'éclairage.

- approche naïve : somme normée des normales unitaires des faces adjacentes

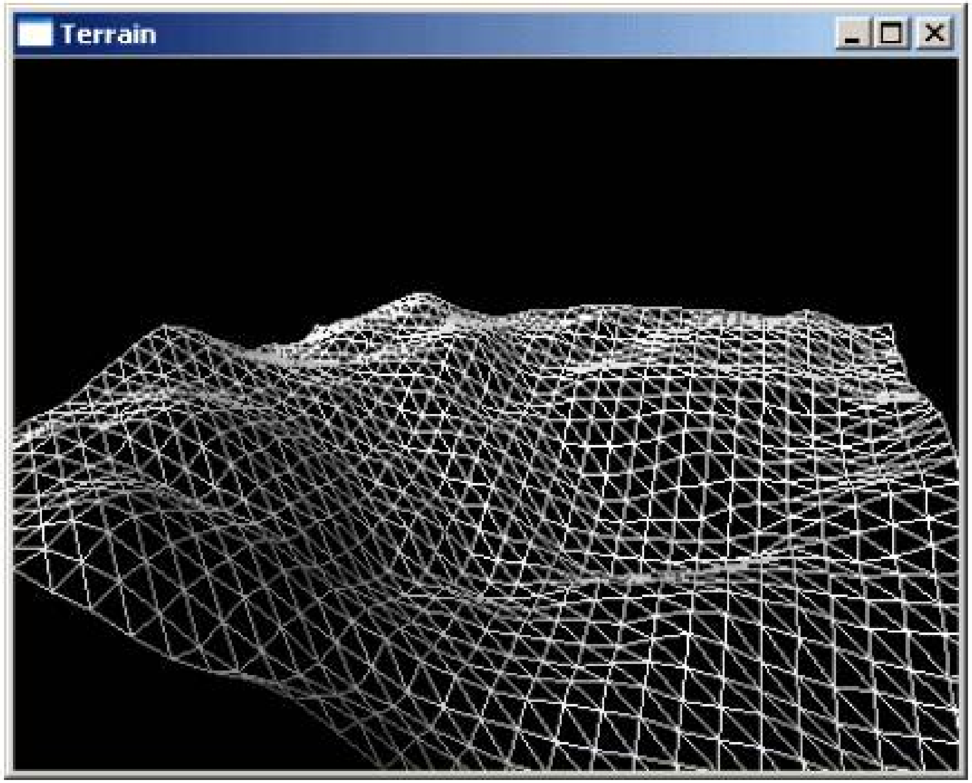
$$N^s = \frac{\sum N^f_i}{\|\sum N^f_i\|}$$

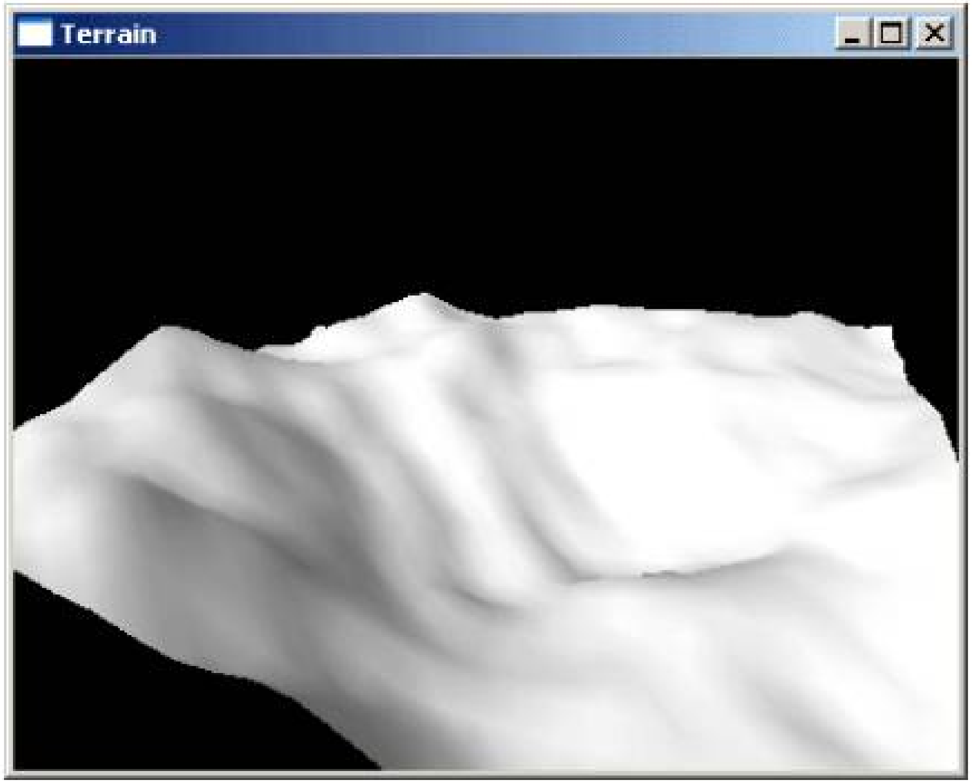
- approches plus avancées : les normales sont pondérées

$$N^s = \frac{\sum \alpha_i N^f_i}{\|\sum \alpha i N^f_i\|}$$

  - si on suppose que le calcul de la normale est extrêmement local, les α_i sont les angles que forment chaque facette au point
  - on peut aussi prendre l'aire des facettes adjacentes pour α_i

- dans tous les cas, on obtient une approximation plus ou moins bonne de la normale à la surface.


Terrain


Terrain

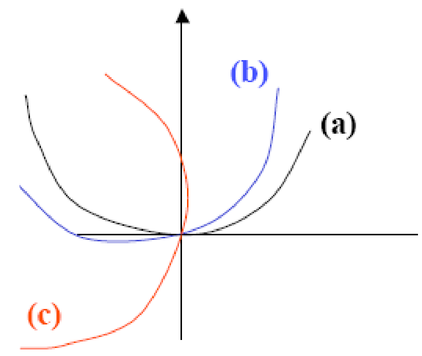
- liste des points et des facettes

- nombre de facettes à créer
  - objet de base : 12,
  - tasse : 100,
  - Personnage : 8 000,
  - forêt : 5 000 000

Impossible de les définir et modifier une à une!

- Plan du module
  - ~~introduction~~
  - ~~représentations des objets~~
  - courbes et surfaces
  - interpolation vs approximation
  - transformations & projections
  - découpage

- utiliser une représentation mathématique :
  $y=f(x)$
  $z=g(x)$



- avantages :
  - en général la simplicité
  - l'accès direct à y et z connaissant x
- inconvénients (majeurs) :
  - tangente verticale : il faut changer de référentiel
  - une rotation altère la définition de la courbe :
  - modification du domaine de variation (b)
  - (c) non représentable par le même type d'équation

- Utiliser une représentation paramétrique

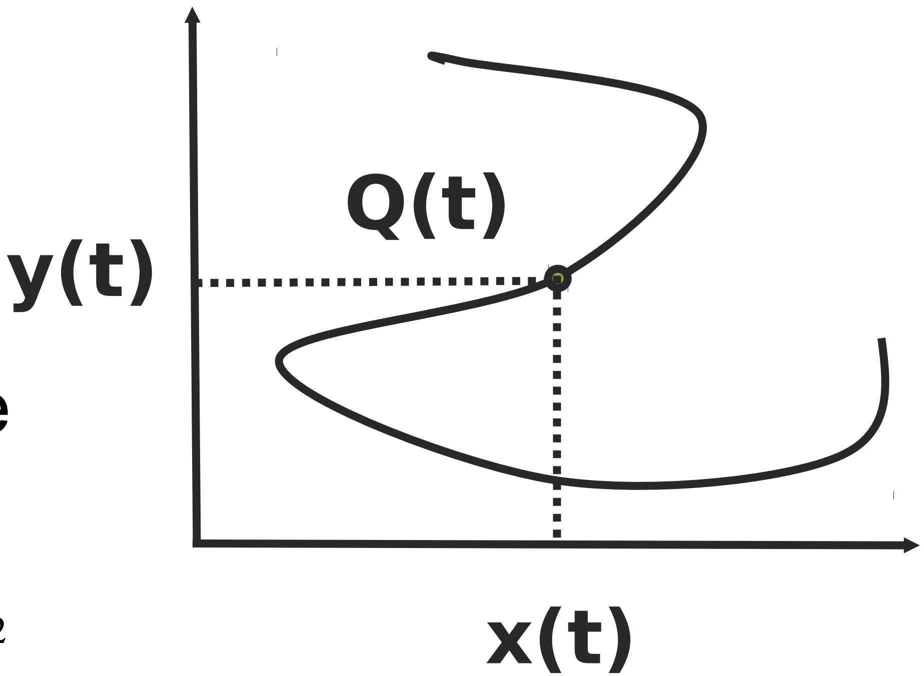
$$X = f(t)$$
$$Y = g(t)$$
$$Z = h(t)$$

- *f*, *g* et *h* sont des polynômes en *t*
- Exemple : $h(t) = a\,t^3 + b\,t^2 + c\,t + d$
- Une courbe peut-être approximée par une partie de courbe polynômiale

Définition des courbes paramétrées

On appelle courbe paramétrée dans l'espace toute application continue :

$$\begin{cases} Q:[a,b] \to IR^3 \\ t \mapsto Q(t) = (x(t), y(t), z(t)) \end{cases}$$



La courbe paramétrée *Q* est dite fermée si *Q(a)=Q(b)*

Si Q est à valeur dans le plan $\mathbb{R}^2$ au lieu de l'espace $\mathbb{R}^3$, la courbe Q est appelée <u>courbe plane</u>

Dérivée d'une courbe paramétrée

Soit *Q:[a,b]→IR²* une courbe paramétrée

- la courbe Q est dite ***dérivable*** en un point *t Є [a,b]* si chacune des fonctions sont dérivables au point t.

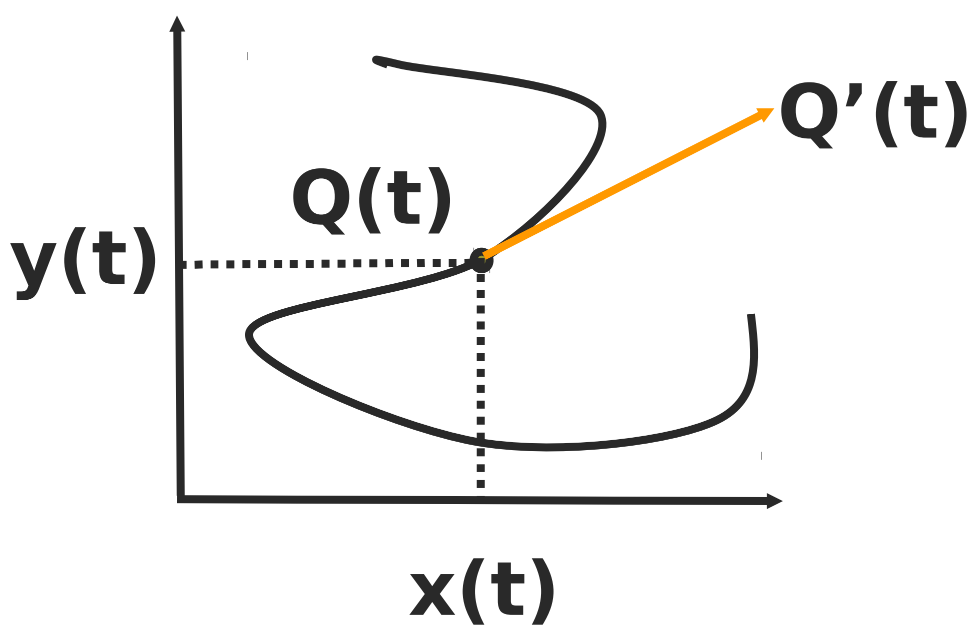
On note le vecteur dérivé dont les coordonnées sont les dérivées au point t

$$Q^{'}(t) = \frac{dQ}{dt} = (x'(t), y'(t), z'(t))$$

- la courbe est dite ***régulière*** si pour tout t Є [a,b], la courbe est dérivable au point t et la dérivée Q'(t) de Q est *non nulle* (c'est-à-dire qu'une au moins des coordonnées de ce vecteur est non nulle).

Nous utiliserons dans la suite des courbes paramétrées régulières

En un point t d’une courbe régulière, le vecteur Q’(t) est un vecteur tangent à la courbe Q au point Q(t)



Raccordement de courbes paramétrées

Soient *Q1:[a,b]→IR3* et <u>Q2:[b,c]→IR3</u> deux courbes paramétrées de classe Ck (dérivables k fois), avec <u>k Є IN</u>

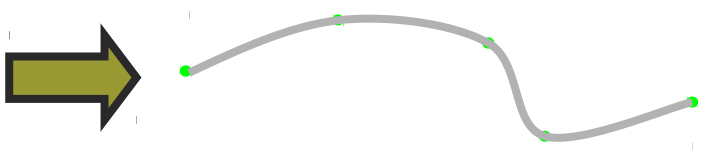
Les courbes Q1 et Q2 se raccordent Ck pour *t=b* si les dérivées de Q1 pour *t=b* coïncident jusqu'à l'ordre k avec les dérivées de Q2 jusqu'à l'ordre k pour *t=b*

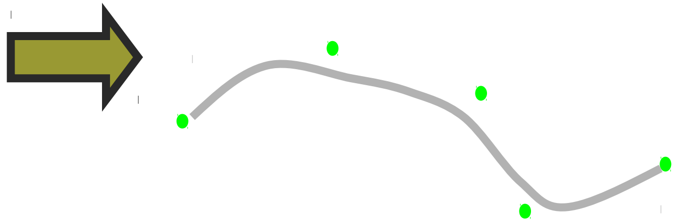
- Interpolation vs approximation

Construction des courbes à partir de *points de contrôle*.

- Etant donné *m* points $P_0, \ldots, P_m$, avec $m \geq 2$, appelés points de contrôle, nous cherchons une courbe lisse (par exemple de classe C^2) qui « ressemble » à la ligne polygonale formée par les points de contrôle.

- On distingue :
  - **<u>L'interpolation</u>** : la courbe doit passer par chaque point de contrôle
  - **<u>L'approximation</u>** : la courbe doit seulement passer à proximité des points de contrôle.





- Courbes de degré d quelconque
  - Une courbe Q : [a,b] → IR^3 qui associe Q(t)=(x(t),y(t),z(t)) est dite *polynomiale* si chacune des coordonnées x(t), y(t), z(t) s'exprime comme un polynôme en t.
  - Si Q est polynomiale, il existe un nombre d, appelé *degré* de la courbe Q, et des coefficients a_i, b_i, c_i pour i=0,…,d tels que pour tout t Є [a,b] on ait l'égalité (E) suivante :

$$(E):\begin{cases} x(t)=\sum_{i=0}^{d} a_i t^i \\ y(t)=\sum_{i=0}^{d} b_i t^i \\ z(t)=\sum_{i=0}^{d} c_i t^i \end{cases}$$

- Courbes de degré d quelconque
  - Soit T le vecteur à d+1 composantes T=(t^d, t^{d-1},…, t, 1)
  - Soit C la matrice

$$C = \begin{bmatrix} a_d & b_d & c_d \\ \vdots & \vdots & \vdots \\ a_0 & b_0 & c_0 \end{bmatrix}$$

  - L'équation (E) précédente est équivalente à l'écriture matricielle suivante :

(E) : Q(t)=(x(t),y(t),z(t))=T . C

- Courbe cubique
  - Une courbe cubique est une courbe polynomiale ayant pour degré d ≤ 3.
  - L'équation (E) s'écrit :

$$(E):\begin{cases} x(t)=a_3t^3+a_2t^2+a_1t+a_0 \\ y(t)=b_3t^3+b_2t^2+b_1t+b_0 \\ z(t)=c_3t^3+c_2t^2+c_1t+c_0 \end{cases}$$

  - L'équation matricielle est Q(t) = T . C avec T=(t^3,t^2,t,1) et la matrice C de taille 4 x 3 :

$$C=\begin{bmatrix} a_3 & b_3 & c_3 \\ a_2 & b_2 & c_2 \\ a_1 & b_1 & c_1 \\ a_0 & b_0 & c_0 \end{bmatrix}$$

- Courbe cubique
  - Connaissant la matrice C et une valeur de t, on peut calculer le point Q(t) à partir de l'équation matricielle précédente.
  - Notons $T'=(3t^2, 2t, 1, 0)$ la dérivée par rapport à t du vecteur T
  - La dérivée de la courbe Q peut être calculée grâce à

    $Q'(t)=T' . C$

- Matrice géométrique : Définition
  - Pour les courbes cubiques, il est fréquent de décomposer C en deux matrice M et G
    - M : matrice 4 x 4, matrice constante qui dépend du type de courbes considérées (courbes hermitiennes, courbes de Bézier, courbes splines, …)
    - G : matrice 4 x 3, matrice *géométrique* qui dépend des points de contrôle et caractérise les contraintes géométriques sur la courbe
    - C = M . G
    - (E) : Q(t) = T . M . G

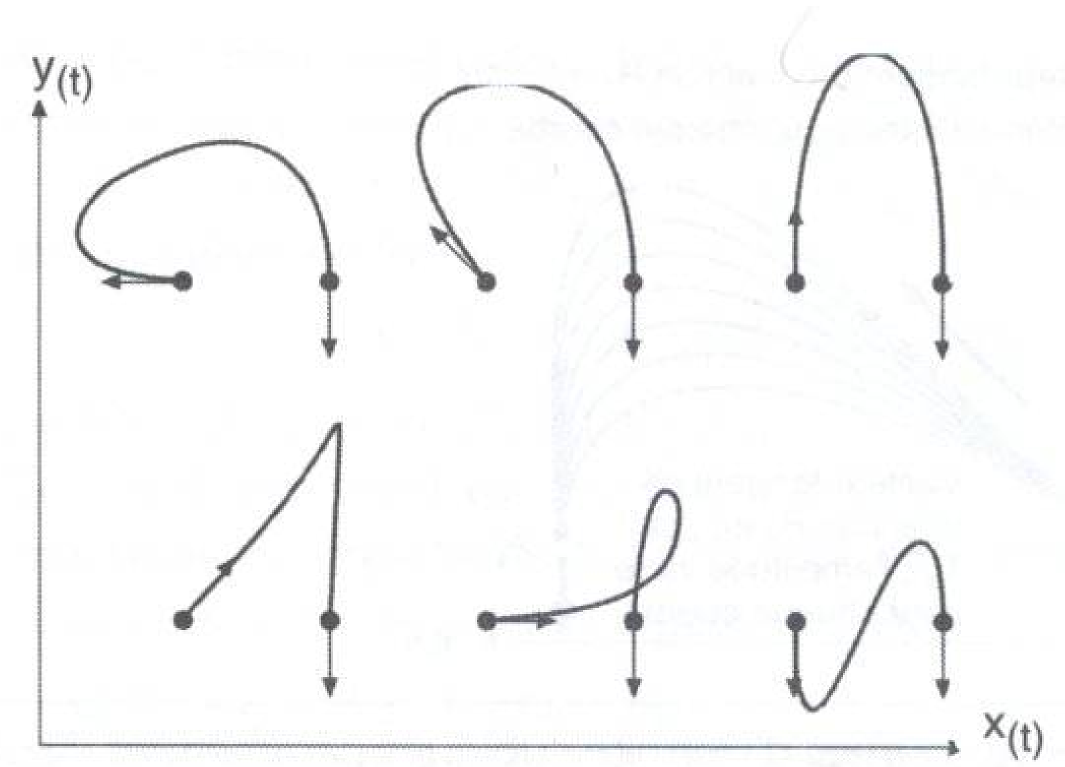
- Matrice géométrique
  - Pour construire une courbe cubique à partir de points de contrôle, on peut faire intervenir différentes sortes de contraintes géométriques, qui forment les coefficients de la matrice géométrique
    - Coordonnées des points de contrôle
    - Coordonnées des dérivées de la courbe aux points de contrôle
    - Contrainte de continuité de raccordement avec d'autres courbes dans le cas de courbes polynomiales par morceaux

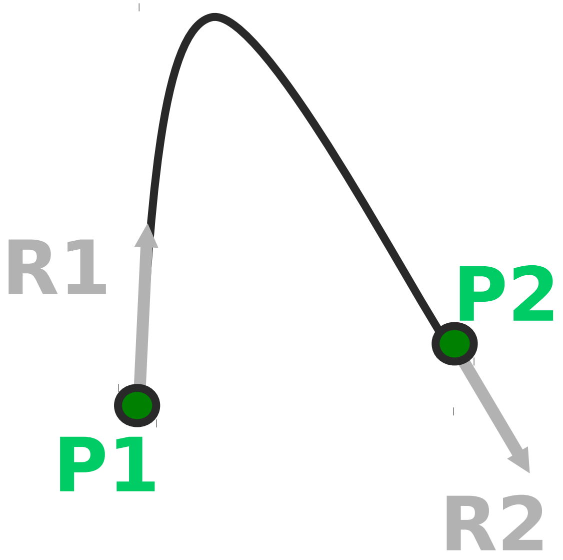
- Dans la suite de ce cours nous étudierons les types de courbes cubiques suivantes :
  - Les courbes hermitiennes, définies par deux points de contrôles et les dérivées en ces points
  - Les courbes de Bézier, définies par deux points de contrôle extrémités, et par deux autres points qui déterminent la dérivée aux extrémités

- Définition
  - Soient $P_1=(x_1,y_1,z_1)$ et $P_4=(x_4,y_4,z_4)$ deux points de contrôle.
  - Soient $R_1=(x'_1,y'_1,z'_1)$ et $R_4=(x'_4,y'_4,z'_4)$ deux vecteurs de IR^3.
  - La courbe hermitienne avec P_1 et P_4 pour points de contrôle et R_1 et R_4 pour dérivées aux points de contrôle est l'unique courbe cubique $Q : [0,1]\rightarrow IR^3$ telle que :
    - $Q(0)=P_1$ et $Q(1)=P_4$
    - $Q'(0)=R_1$ et $Q'(1)=R_4$.
  - On obtient la matrice géométrique pour les courbes hermitiennes G_H suivante :

$$G_H=\begin{bmatrix} x_1 & y_1 & z_1 \\ x_4 & y_4 & z_4 \\ x'_1 & y'_1 & z'_1 \\ x'_4 & y'_4 & z'_4 \end{bmatrix}=\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

- Exemple





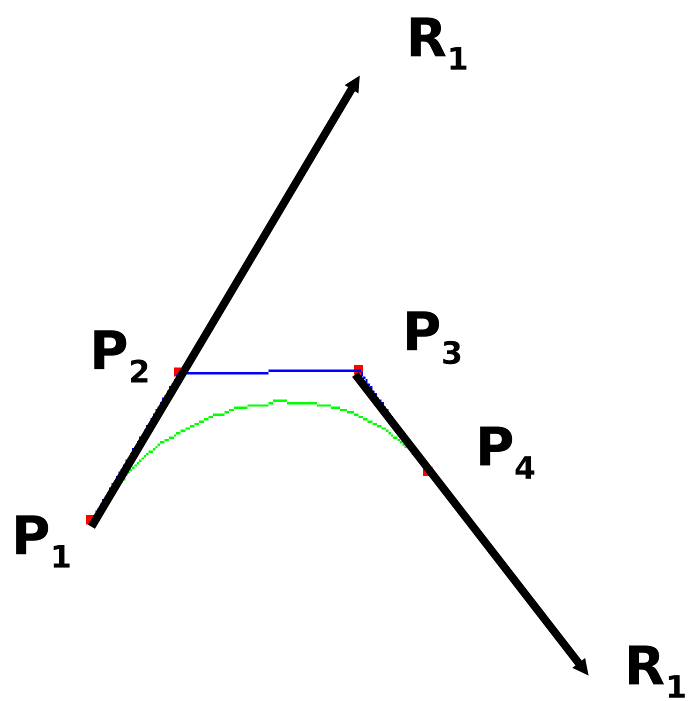
- Définition : Matrice hermitienne M_H
  - $Q(t) = T \cdot M_H \cdot G_H$ avec $T = (t^3, t^2, t, 1)$
  - G_H : matrice géométrique
  - M_H : *matrice hermitienne* (matrice constante)
- Calcul de la matrice hermitienne
  - Sachant que $Q(0)=P_1$, $Q(1)=P_4$, $R_1=Q'(0)$ et $R_4 = Q'(1)$ on a :

$$M_H = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$Q(t) = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

- Définition
  - Courbe hermitienne particulière approximant une ligne polygonale de 4 points de contrôles
  - La courbe de Bézier cubique de points de contrôles P_1, P_2, P_3, P_4 est la courbe hermitienne d'extrémités P_1 et P_4 avec pour dérivée aux extrémités R_1 et R_4 avec :
    - $R_1 = 3(P_2 - P_1)$
    - $R_4 = 3(P_4 - P_3)$

$$G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$



- Matrice de Bézier

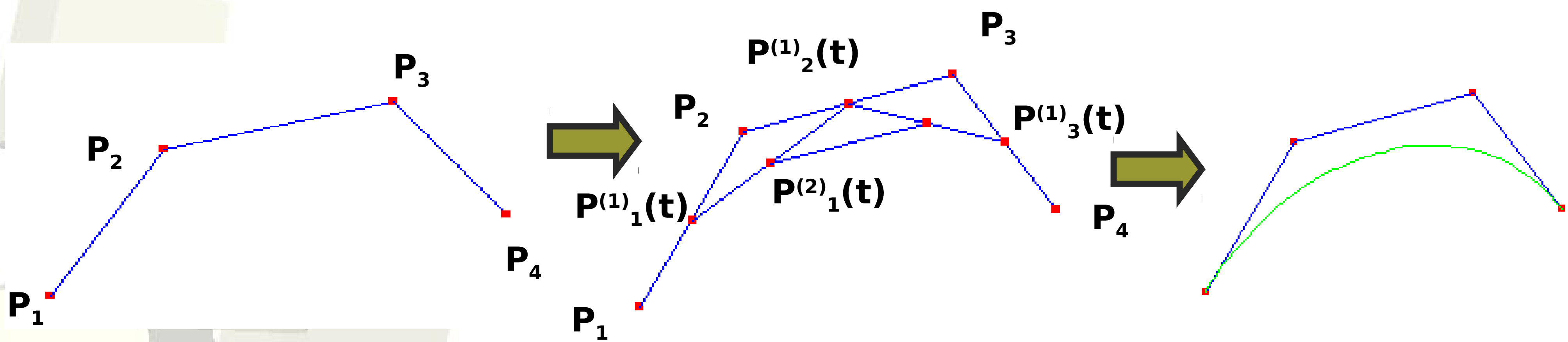
$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4$$

- Algorithme de Casteljau
  - Définition
    - Considérons une ligne brisée ayant pour sommets P1, P2, P3, P4 et une valeur t Є [0,1]. On peut construire pour i=1,2,3 un point

      $P_i^{(1)}(t)=(1-t)P_i+t\,P_{i+1}$
    - Pour i=1,2,3 le point $P_i^{(1)}(t)$ se trouve sur le segment $[P_i, P_{i+1}]$.
    - Considérons maintenant la ligne brisée formée par les trois points $P_1^{(1)}(t)$, $P_2^{(1)}(t)$ et $P_3^{(1)}(t)$. On peut appliquer le même procédé pour i=1,2

      $P_i^{(2)}(t)=(1-t)P^{(1)}_i+t\,P^{(1)}_{i+1}$



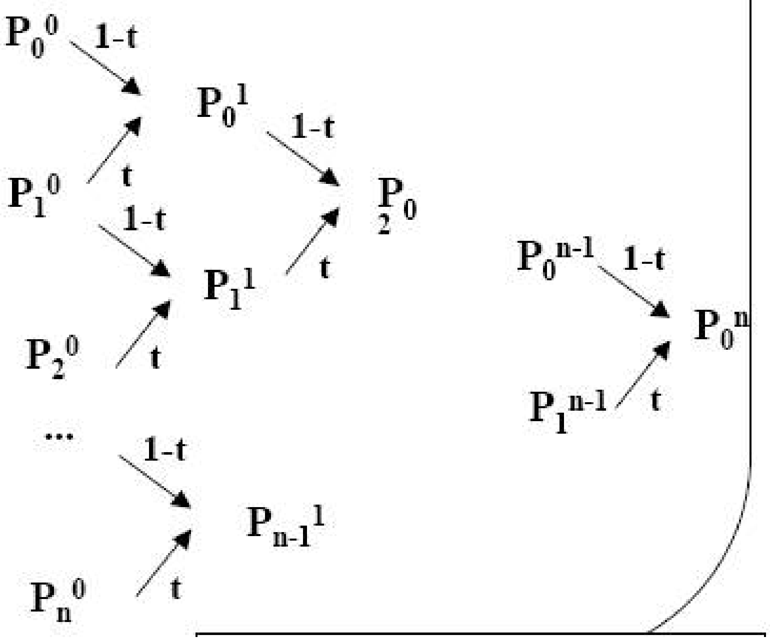
- Algorithme de De Casteljau

```
Début
    Pour j=0 à n
        Pi^0 = Pi
    FinPour
    Pour i=1 à n
        Pour j=0 à n-i
            Pj^i = (1-t) Pj^(i-1) + t Pj+1^(i-1)
        FinPour
    FinPour
    //                 C(t) = P0^n
Fin
```

$$P_i^0 = P_i$$

$$P_j^i = (1-t)\, P_j^{i-1} + t\, P_{j+1}^{i-1}$$

$$C(t) = P_0^n$$



- Définition par l'algorithme de Casteljau
  - On peut généraliser les courbes de Bézier à un nombre $n \geq 2$ quelconque de points de contrôles
  - Une courbe de Bézier d'ordre n-1 est définie par les points $P_0, \ldots, P_{n-1}$.
  - Pour définir Q(t)
    - On pose $b_i^{(0)}(t)=P_i$ pour $i=0,\ldots,n-1$
    - Pour $r=1,\ldots, n-1$ et $i = 0,\ldots, n-1-r$, on pose

      $$b_i^{(r)}(t)=(1-t)b_i^{(r-1)}(t)+tb_{i+1}^{(r-1)}(t)$$
    - On définit ainsi par récurrence un point $Q(t)=b_0^{(n-1)}(t)$

- Polynômes de Bernstein
  - Les polynômes de Bernstein $B_{i,n}$ de degré n sont définis pour i=0,…, n par la formule

$$Q(t) = \sum_{i=0}^{n-1} P_i B_{i,n-1}(t)$$

  - Il y a (n+1) polynômes de Bernstein de degré n
  - Théorème : Soit maintenant P_0, …, P_{n-1} les points de contrôle. Soit t Є[0,1]. La courbe de Bézier Q d'ordre (n-1) ayant les points P_i pour points de contrôles s'exprime par

$$B_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}$$

  - La courbe Q s'exprime comme une somme des polynômes $B_{i,n-1}$, pondérée par les points de contrôles

Polynômes de Bernstein

- <u>**Propriété fondamentale 2**</u> **: positivité**

$$\forall t \in [0,1] \quad \varphi_{n,i}(t) \geq 0$$

$$\forall t \in]0,1[\quad \varphi_{n,i}(t) > 0$$

- <u>**Propriété fondamentale 3**</u> **: dite de symétrie**

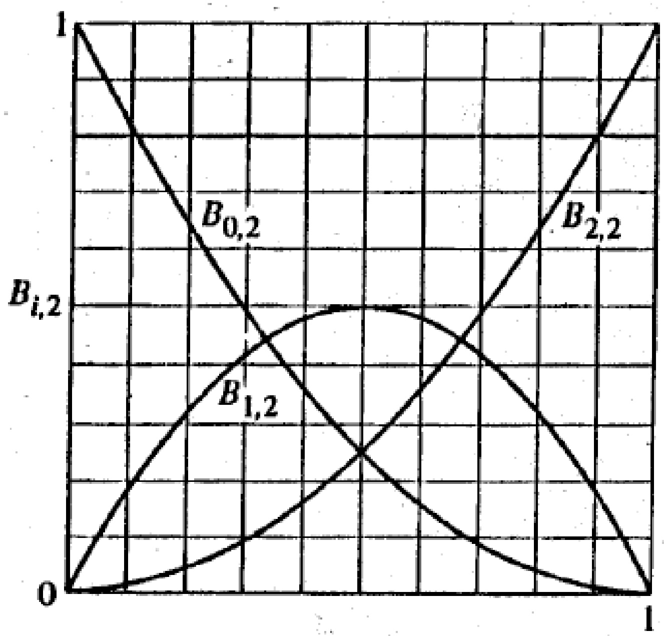
$$\forall t \in [0,1] \quad \varphi_{n,i}(t) = \varphi_{n,n-i}(1-t)$$

- <u>**Propriété fondamentale 4**</u> **: partition de l'unité**

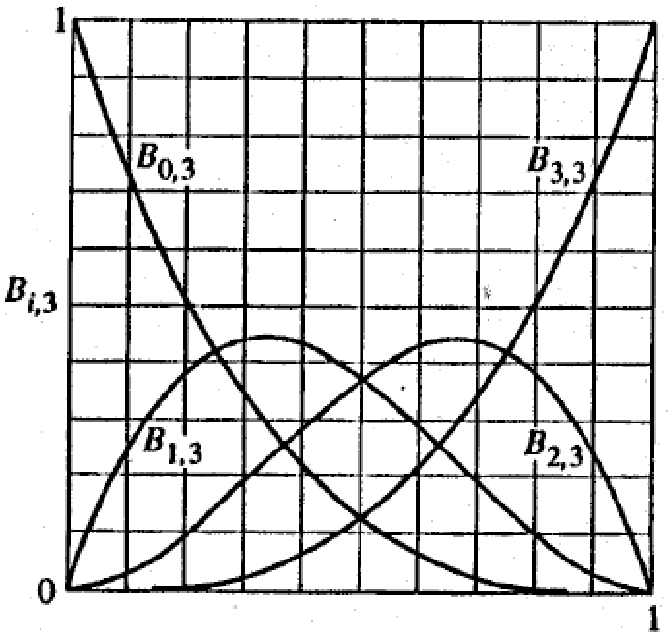
$$\forall t \in [0,1] \quad \sum_{i=0}^{n} \varphi_{n,i}(t) = 1 \qquad \text{(vrai sur } \Re\text{)}$$

Polynômes de Bernstein

Quelques exemples



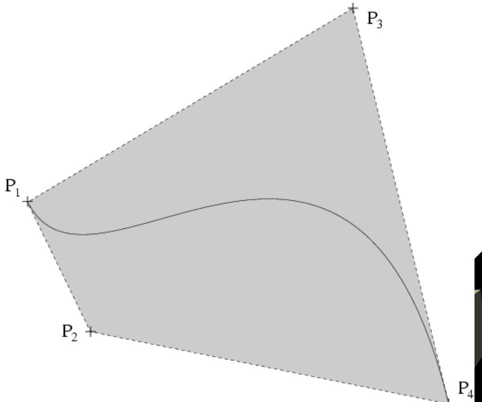
$$\begin{cases} B_{0,2}(t) = t^2 - 2t + 1 \\ B_{1,2}(t) = -2t^2 + 2t \\ B_{2,2}(t) = t^2 \end{cases}$$



$$\begin{cases} B_{0,3}(t) = -t^3 + 3t^2 - 3t + 1 \\ B_{1,3}(t) = 3t^3 - 6t^2 + 3t \\ B_{2,3}(t) = -3t^3 + 3t^2 \\ B_{3,3}(t) = t^3 \end{cases}$$

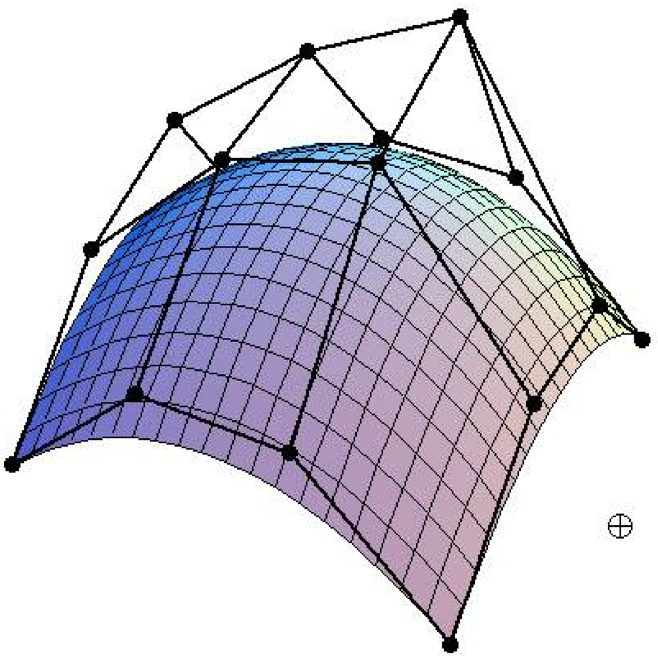
Représentations des Courbes / Courbes polynomiales (Avantages / Inconvénients)

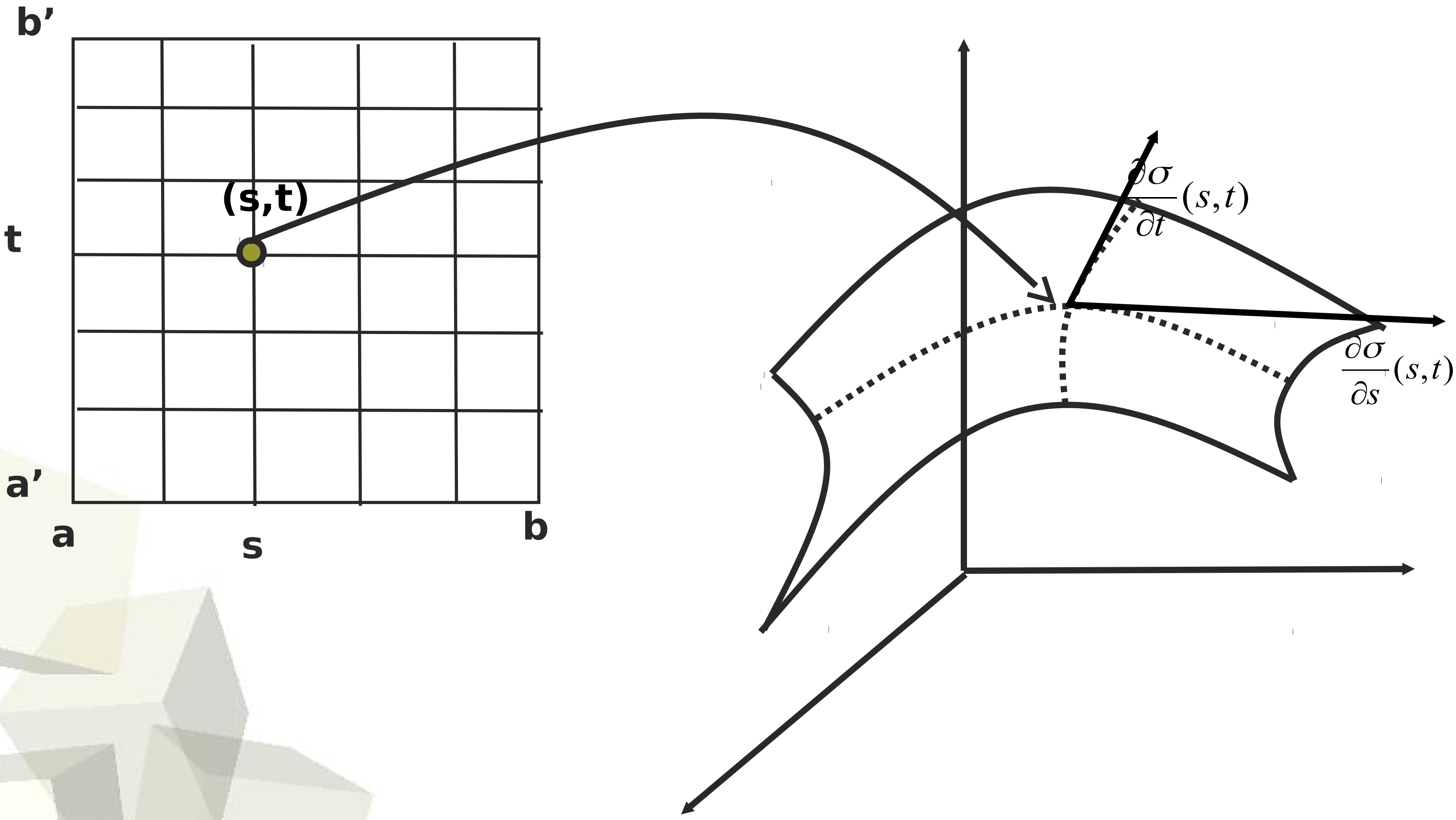
- Invariance par translation, changement d'échelle ou rotation (cf algo de Casteljau) -> intéressant si besoin de transformations
- Enveloppe convexe : la courbe se trouve dans l'enveloppe convexe des points de contrôle
- Problème du contrôle global : la modification d'un seul point affecte toute la courbe -> problématique pour un graphiste qui veut affiner son dessin en modifiant un point de contrôle
- Problème du degré élevé : si la forme doit être complexe (nombreux virages), sa représentation sous forme de courbe de Bézier aura un degré très élevé -> Coûteux en calcul



- Les courbes B-Splines se définissent à partir d'une base de fonctions.
- A la différence des courbes de Bézier, les courbes B-Splines ne sont pas polynomiales, mais polynomiales par morceaux
- Permet d'approximer un nombre quelconque de points de contrôles par des courbes de degré fixé, par exemple cubique par morceaux
- Les avantages:
  - Contrôle local de la courbe
  - Le degré des courbes est peu élevé

- Surface définie par un ensemble de points de contrôles
- Manipulation des points de contrôles afin de modifier la surface sous jacentes




b'
a'
t
(s,t)
a
s
b
$\frac{\partial \sigma}{\partial t}(s,t)$
$\frac{\partial \sigma}{\partial s}(s,t)$

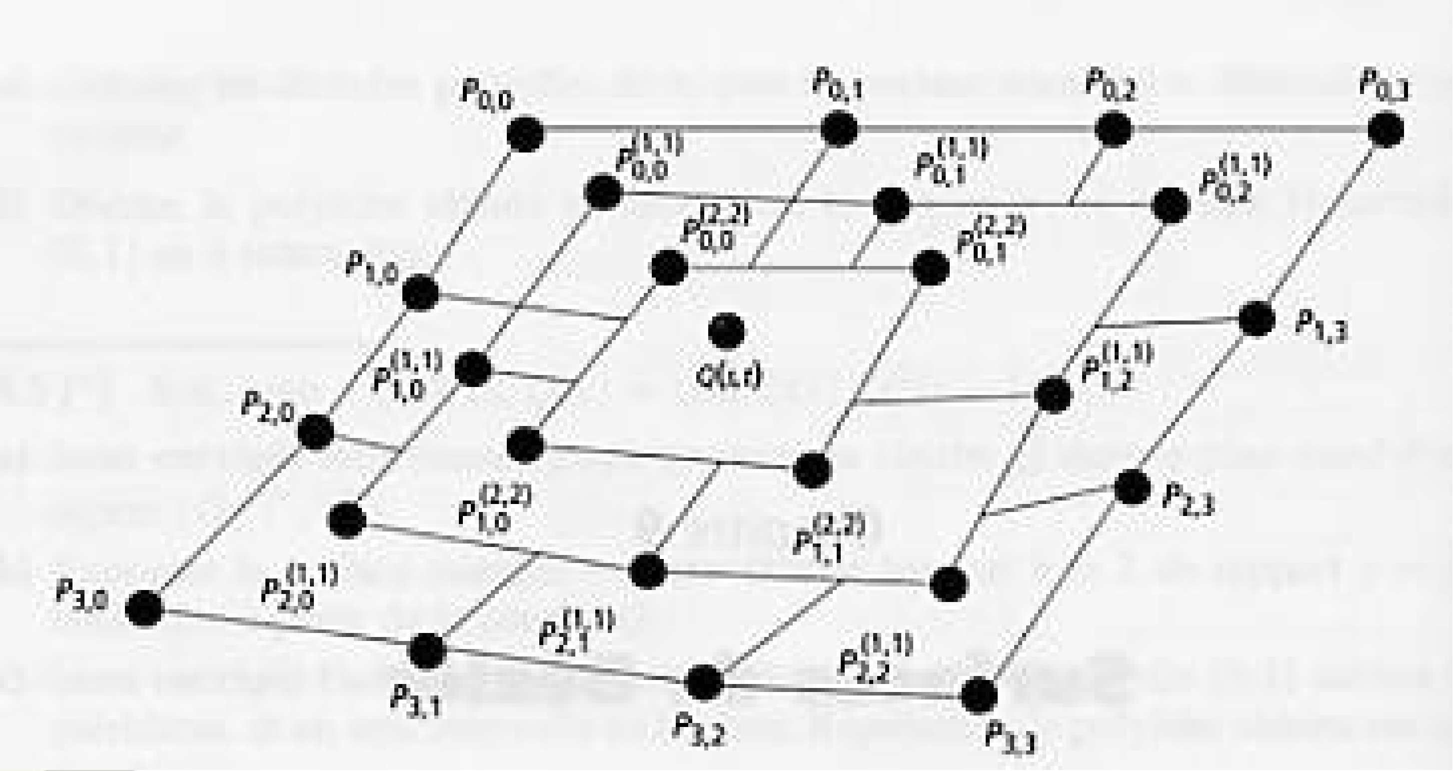
- Définition : Interpolation bilinéaire et algorithme de Casteljau
  - Dans le cas des surfaces, les points de contrôles forment un réseau $P_{i,j}$, pour i=0,…m-1 et j=0,…,n-1
  - Considérons le cas de la surface la plus simple qui passe par $P_{i,j}$, $P_{i+1,j}$, $P_{i,j+1}$, $P_{i+1,j+1}$: *la surface réglée*
  - Points intermédiaires

$$\begin{cases} P_{i,j}^{(0,1)} & = & (1-t)P_{i,j} + t\,P_{i,j+1} \\ P_{i,j}^{(1,0)} & = & (1-t)P_{i+1,j} + t\,P_{i+1,j+1} \end{cases}$$

puis on pose :

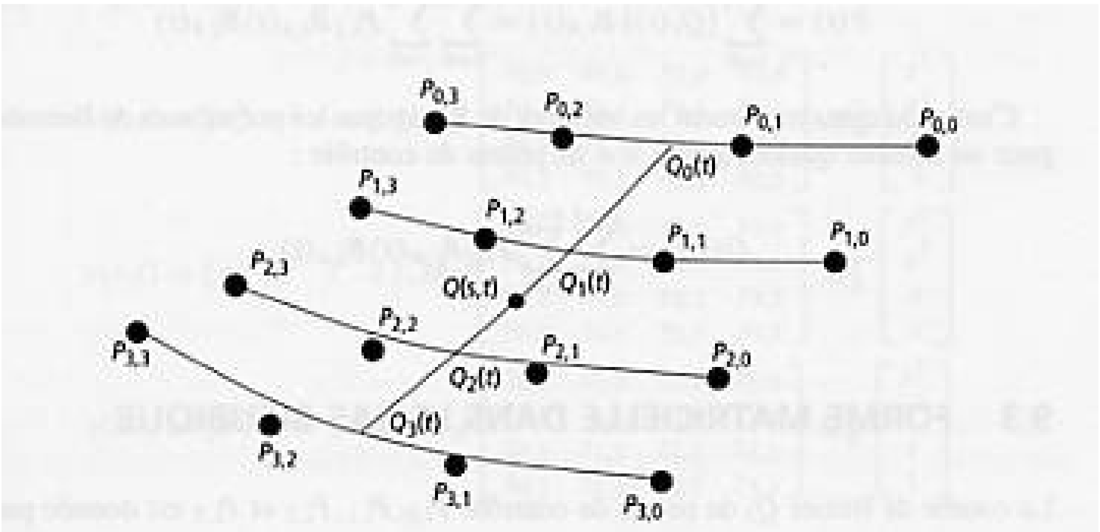
$$P_{i,j}^{(1,1)}(s,t) = (1-s)P_{i,j}^{(0,1)} + s\,P_{i,j}^{(1,0)} = \begin{bmatrix} 1-s & s \end{bmatrix} \begin{bmatrix} P_{i,j} & P_{i,j+1} \\ P_{i+1,j} & P_{i+1,j+1} \end{bmatrix} \begin{bmatrix} 1-t \\ t \end{bmatrix}$$

- Interpolation bilinéaire et algorithme de Casteljau



- Définition : Produit tensoriel; réseaux de courbes de Bézier
  - Soit des points de contrôles $P_{i,j}$, pour i=0,…,3 et j=0,…, 3
  - Soit Q_0 la courbe de Bézier définie par les points de contrôles $P_{0,0}$, $P_{0,1}$, $P_{0,2}$ et $P_{0,3}$
  - Expression des courbes de Bézier par les polynômes de Bernstein :

$$Q_0(t) = \sum_{j=0}^{3} P_{0,j} B_{j,4}(t)$$



- Produit tensoriel : réseaux de courbes de Bézier
  - Soit Q_1 la courbe de Bézier définie par les points de contrôles $P_{1,0}$, $P_{1,1}$, $P_{1,2}$ et $P_{1,3}$
    - Expression des courbes de Bézier par les polynômes de Bernstein :

$$Q_1(t) = \sum_{j=0}^{3} P_{1,j} B_{j,4}(t)$$

  - Soit Q_2 la courbe de Bézier définie par les points de contrôles $P_{2,0}$, $P_{2,1}$, $P_{2,2}$ et $P_{2,3}$
    - Expression des courbes de Bézier par les polynômes de Bernstein :

$$Q_2(t) = \sum_{j=0}^{3} P_{2,j} B_{j,4}(t)$$

  - Soit Q_3 la courbe de Bézier définie par les points de contrôles $P_{3,0}$, $P_{3,1}$, $P_{3,2}$ et $P_{3,3}$
    - Expression des courbes de Bézier par les polynômes de Bernstein :

$$Q_3(t) = \sum_{j=0}^{3} P_{3,j} B_{j,4}(t)$$

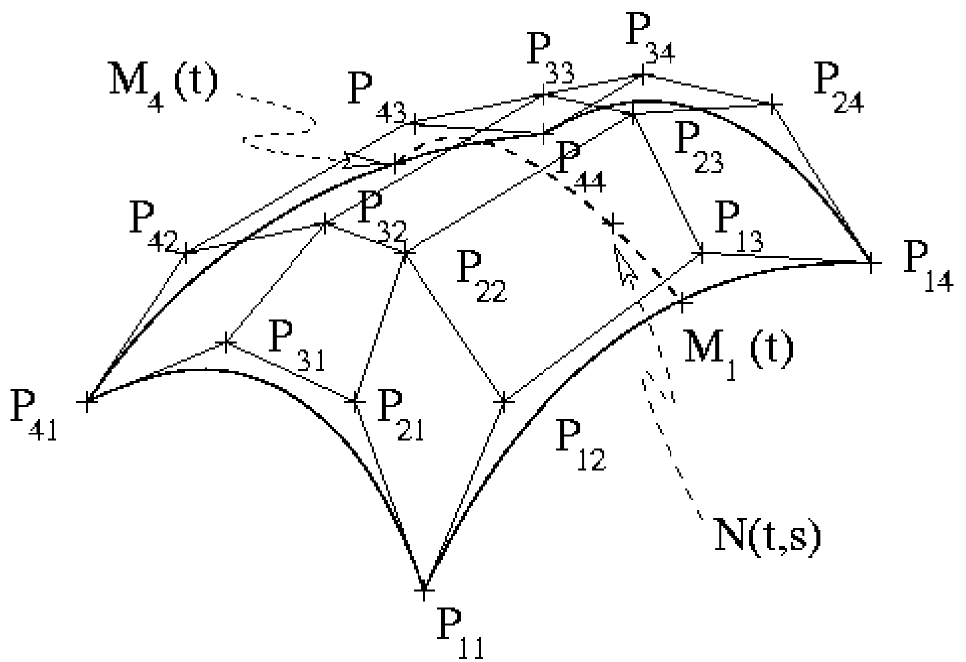
- Produit tensoriel : réseaux de courbes de Bézier
  - Soit Q(s,t) = P(s), où P est la courbe de Bézier de points de contrôles $Q_0(t)$, $Q_1(t)$, $Q_2(t)$ et $Q_3(t)$
  - On a

$$P(t) = \sum_{i=0}^{3} [Q_{i(t)}] B_{j,4}(s) = \sum_{i=0}^{3} \sum_{j=0}^{3} P_{i,j} B_{i,4}(s) B_{j,4}(t)$$

- Cas général (m x n points de contrôles)

$$Q(s,t) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} P_{i,j} B_{i,m}(s) B_{j,n}(t)$$

- Une surface bicubique de Bézier est définie par 16 points de contrôle P*i,j* (*i,j* dans {1,4}x{1,4}) en calculant :
  - quatre points $M_k(t)$ (*k* dans {1,4}) sur les courbes de Bézier cubiques définies par $P_{k,j}$ (*j* dans {1,4}) ,
  - et le point N(*t*,*s*) sur la courbe de Bézier définie par les quatre points $M_k(t)$ précédents.



Courbe de Bézier

$$M_1(t) = \begin{bmatrix} B_0^3(t) & B_1^3(t) & B_2^3(t) & B_3^3(t) \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \\ P_{14} \end{bmatrix}$$

Surface de Bézier

$$N(t,s) = B_0(s)\,M_1(t) + B_1(s)\,M_2(t) + B_2(s)\,M_3(t) + B_3(s)\,M_4(t)$$

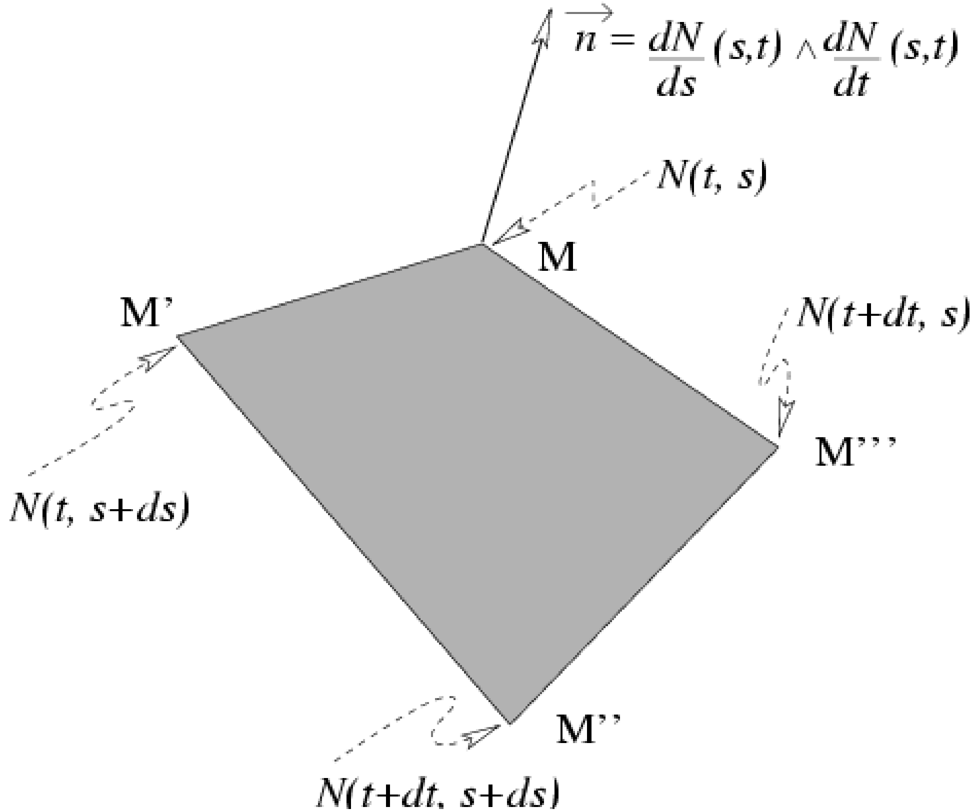
$$N(t,s) = \begin{bmatrix} B_0^3(t) \\ B_1^3(t) \\ B_2^3(t) \\ B_3^3(t) \end{bmatrix}^t \begin{bmatrix} P_{11} & P_{21} & P_{31} & P_{41} \\ P_{12} & P_{22} & P_{32} & P_{42} \\ P_{13} & P_{23} & P_{33} & P_{43} \\ P_{14} & P_{24} & P_{34} & P_{44} \end{bmatrix} \begin{bmatrix} B_0^3(s) \\ B_1^3(s) \\ B_2^3(s) \\ B_3^3(s) \end{bmatrix}$$

$$N(t,s) = \begin{bmatrix} t^3 & t^2 & t^1 & t^0 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{11} & P_{21} & P_{31} & P_{41} \\ P_{12} & P_{22} & P_{32} & P_{42} \\ P_{13} & P_{23} & P_{33} & P_{43} \\ P_{14} & P_{24} & P_{34} & P_{44} \end{bmatrix}$$

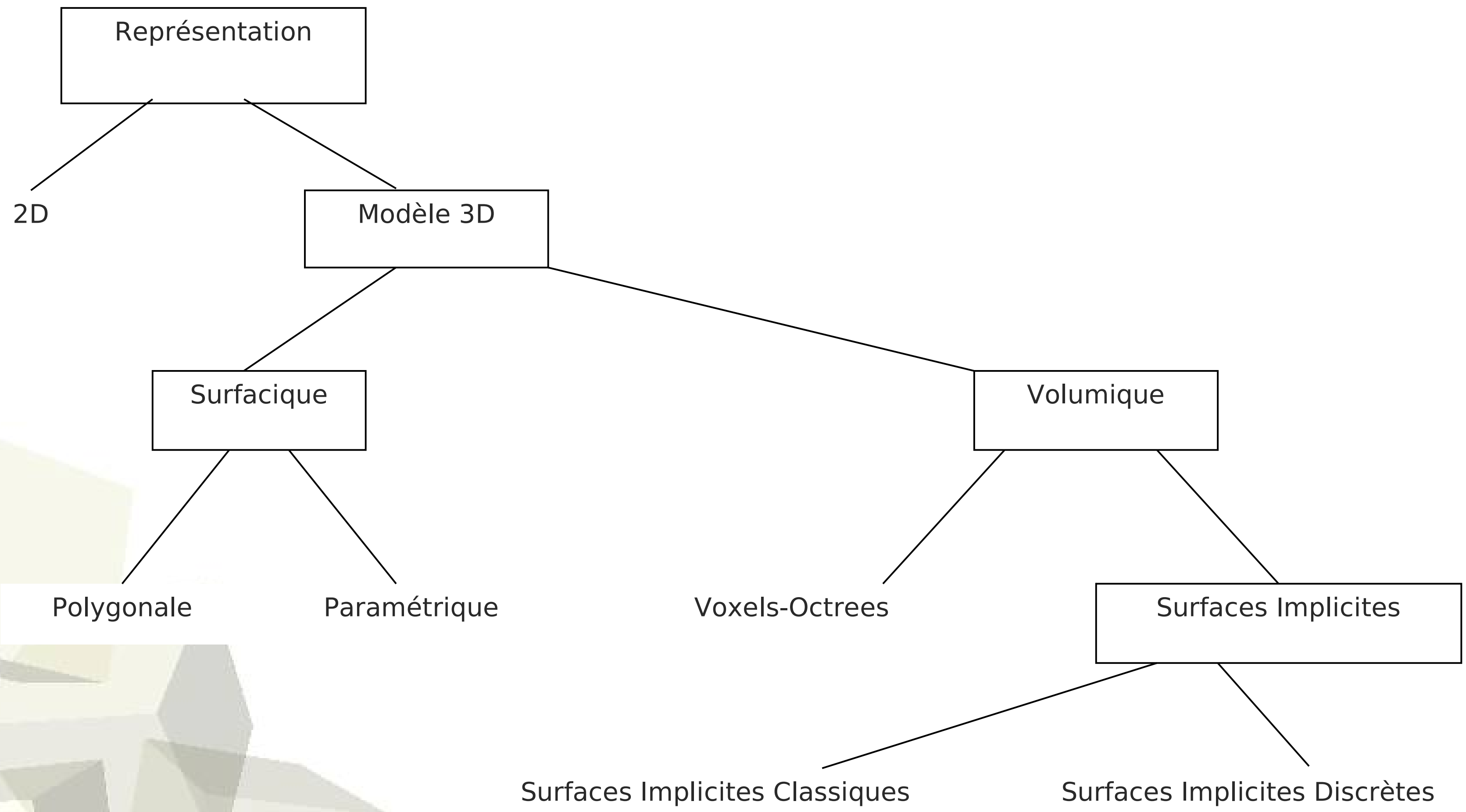
$$\times \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} s^3 \\ s^2 \\ s^1 \\ s^0 \end{bmatrix}$$

- Facettisation des surfaces paramétrées
  - Facettiser ⇔ approximer par un polyèdre
  - Recherche des sommets
    - Découpage de l'intervalle [a,b] en m intervalles
    - Découpage de l'intervalle [a',b'] en p intervalles
  - Construction des facettes
    - Construction de 2mp facettes triangulaires (=> planes)

- **Mode de construction du maillage**
  - La courbe se construit par un maillage formé de polygones (non nécessairement plans).
    Les sommets des polygones sont obtenus en faisant varier ***t* de *dt*** et ***s* de *ds***.



Représentations volumiques



- Propriétés des représentations

  - Densité

La densité d'une méthode de représentation est évaluée par rapport au coût mémoire nécessaire à sa sauvegarde.

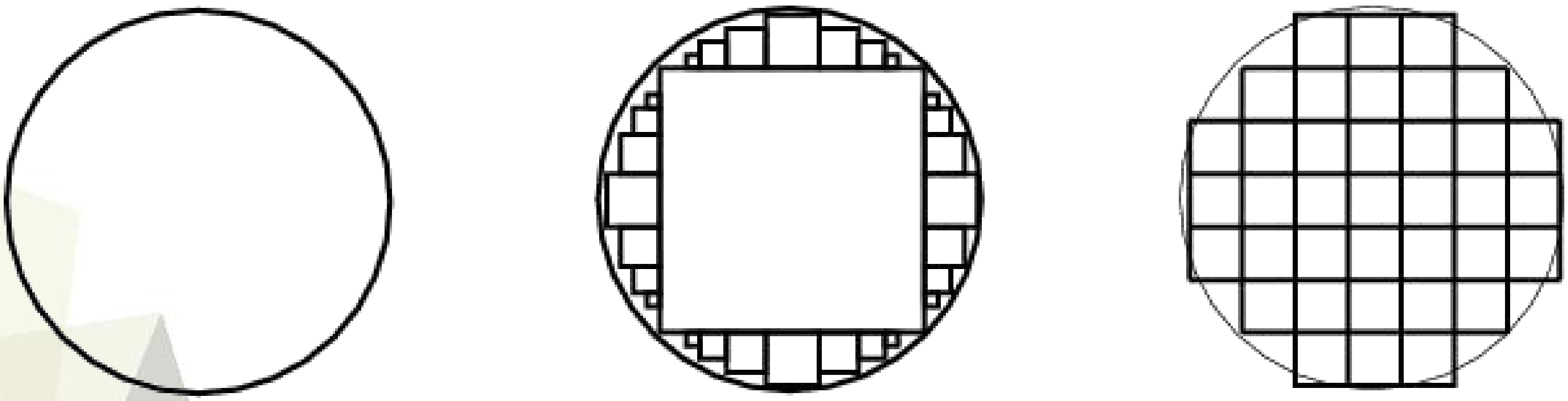
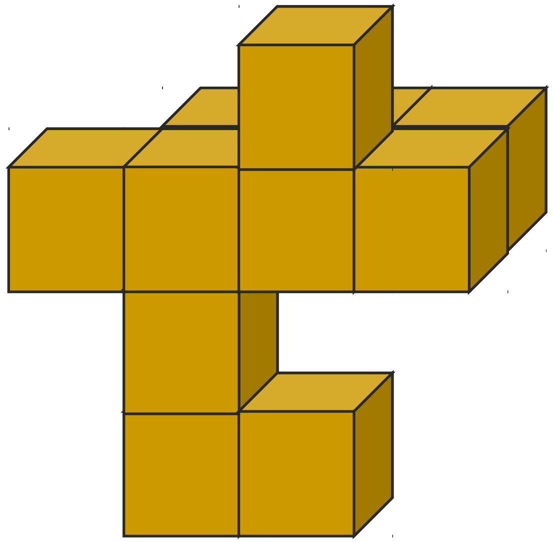


Figure 23: Exemples de représentations données en ordre décroissant de la densité: (a) la représentation paramétrique- instanciation de primitives, (b) la représentation décompositive avec des cellules adaptives, et (c) la représentation décompositive avec des cellules uniformes

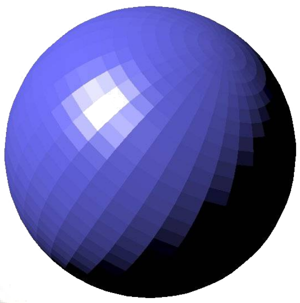
Volumes discrets

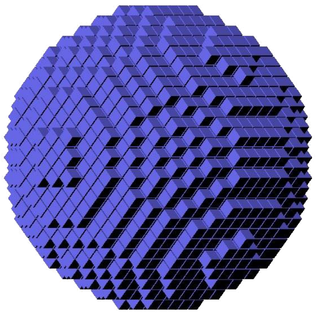
- *Voxels* = éléments d'une grille 3D
- Présence ou absence de matière



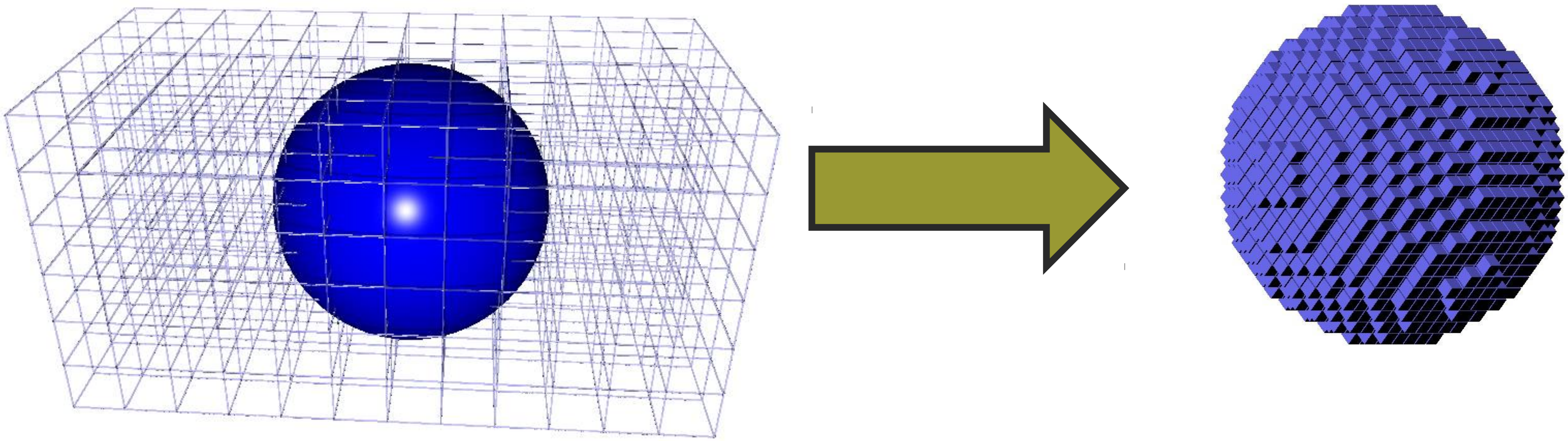
Visualisation

- Rendu volumique
- Marching cubes

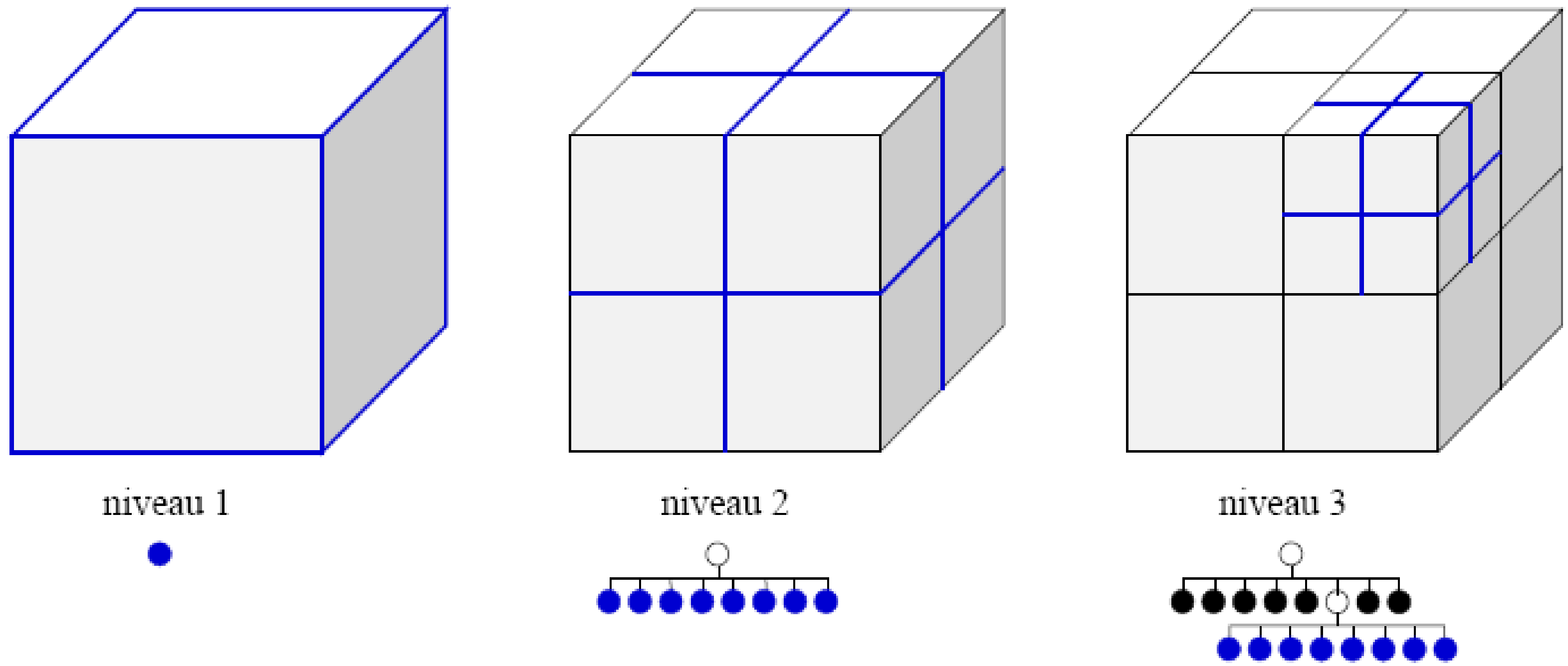


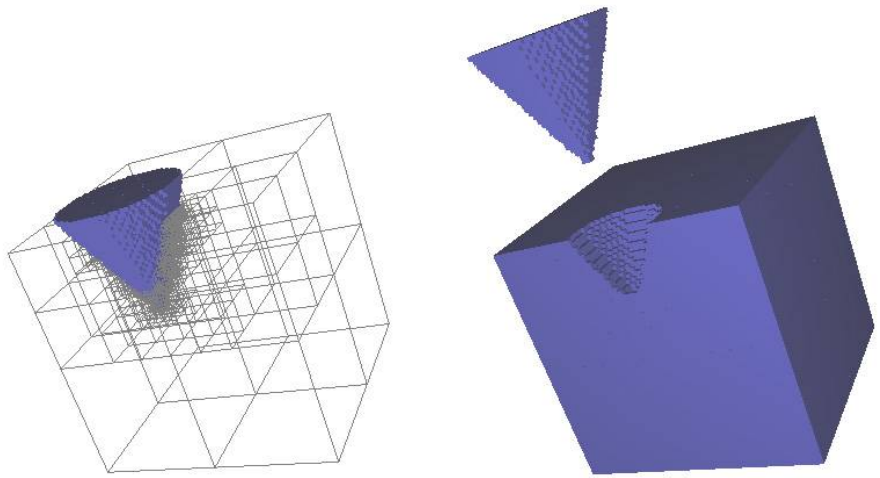


- Décomposition de l'objet en « cellules »



Représentations volumiques / arbre octal





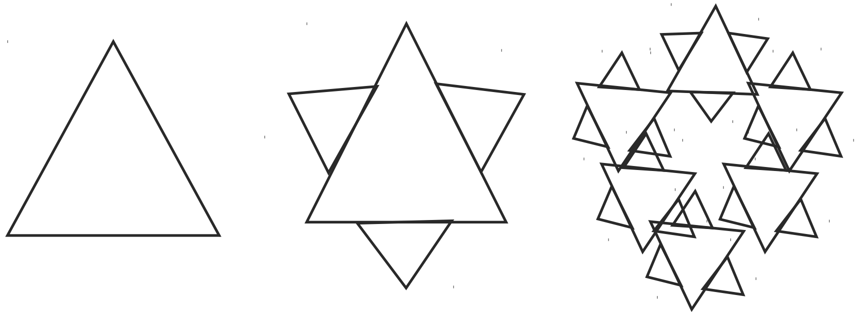
- Primitives géométriques créées par une procédure
  - Croissance progressive
  - Placement procédural

- Utile pour objets complexes et répétitifs

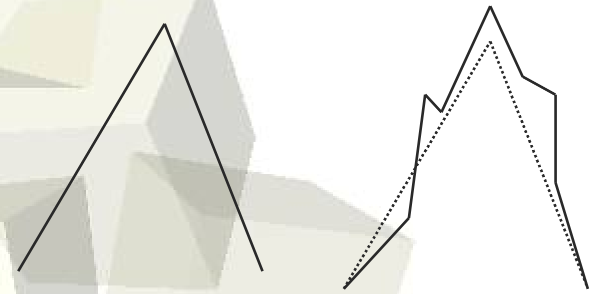
ex : plante, paysage, ville

« règles de construction »

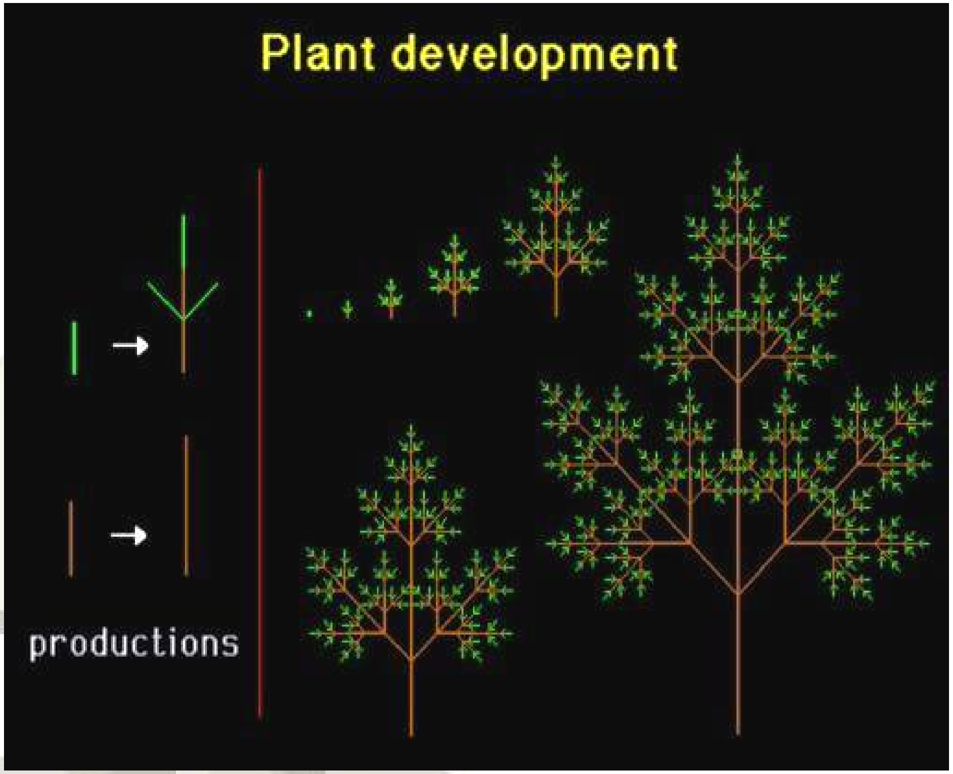
- Exemple 1 :
  - Fractales
    - Ajout récursif de détails

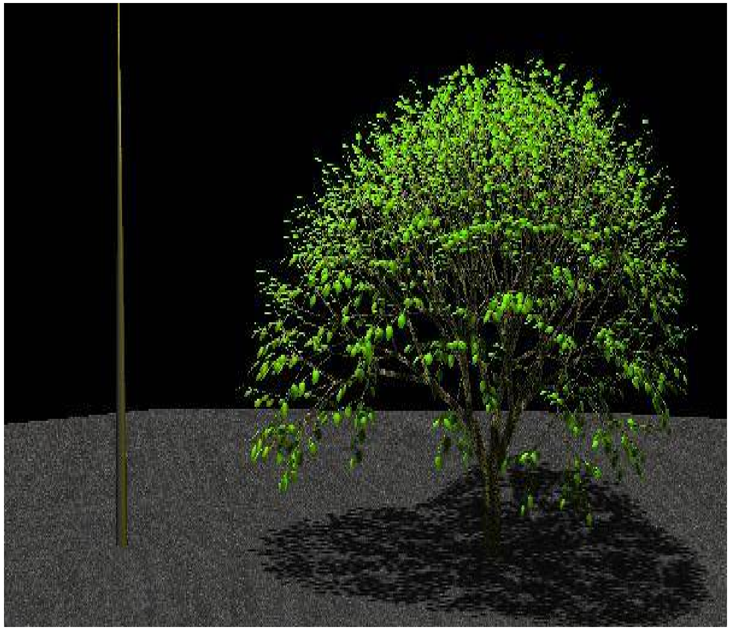


  - Terrain fractals
    - Déplacement aléatoire à chaque étape



- Exemple 2 :
  - Plantes : L-systèmes
    - Grammaire régissant la croissance





GÉOMÉTRIE CONSTRUCTIVE DES SOLIDES: BUTS

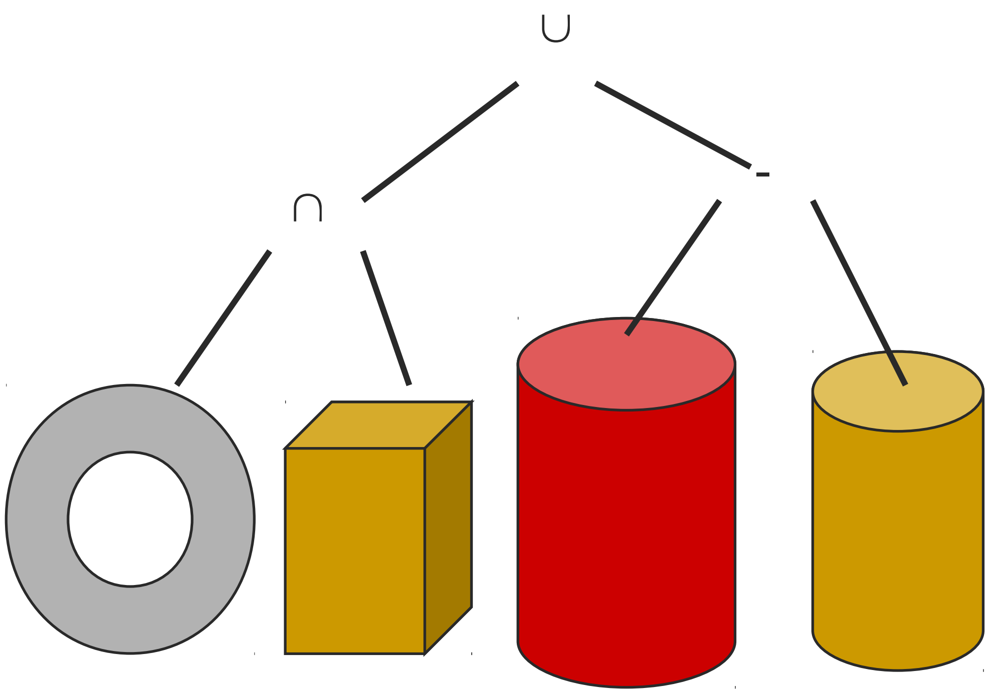
Buts

Construire des solides complexes à partir d´**opérations élémentaires** sur des **solides élémentaires**.

- limiter les librairies d´objets géométriques élémentaires,
- fournir des mécanismes opératoires pour la constructions des solides par **agglomération**, **intersection** ou **extrusion**,
- fournir un mécanismes récursif de construction d´objets permettant la création de librairies.

Constructive Solid Geometry

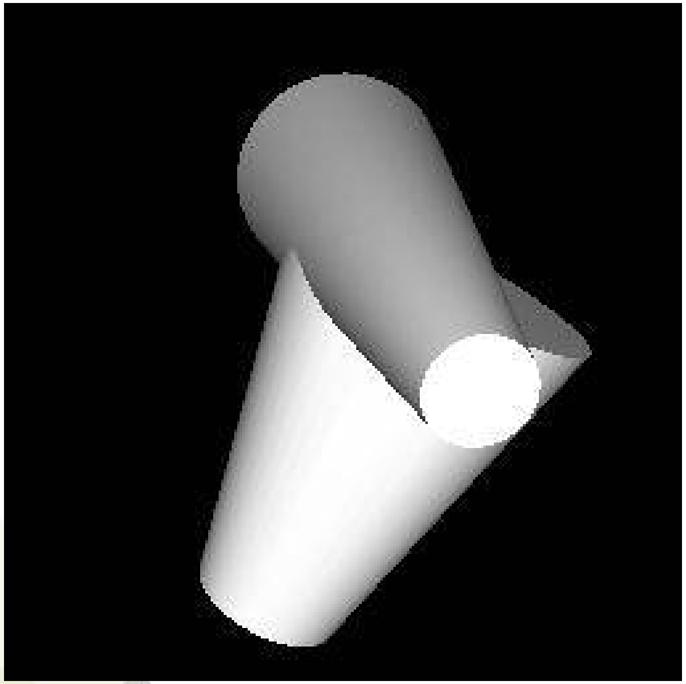
- Opérateurs booléens
  - Union (ou)
  - Intersection (et)
  - Différence (not)
- Arbre de construction



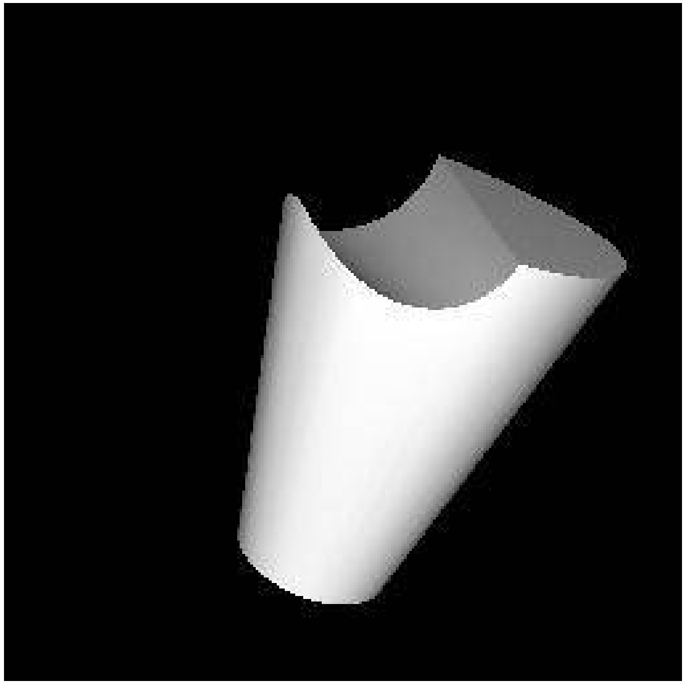
- Très utilisé en CAO,

mais visualisation délicate

- Exemple avec 2 primitives :



Union



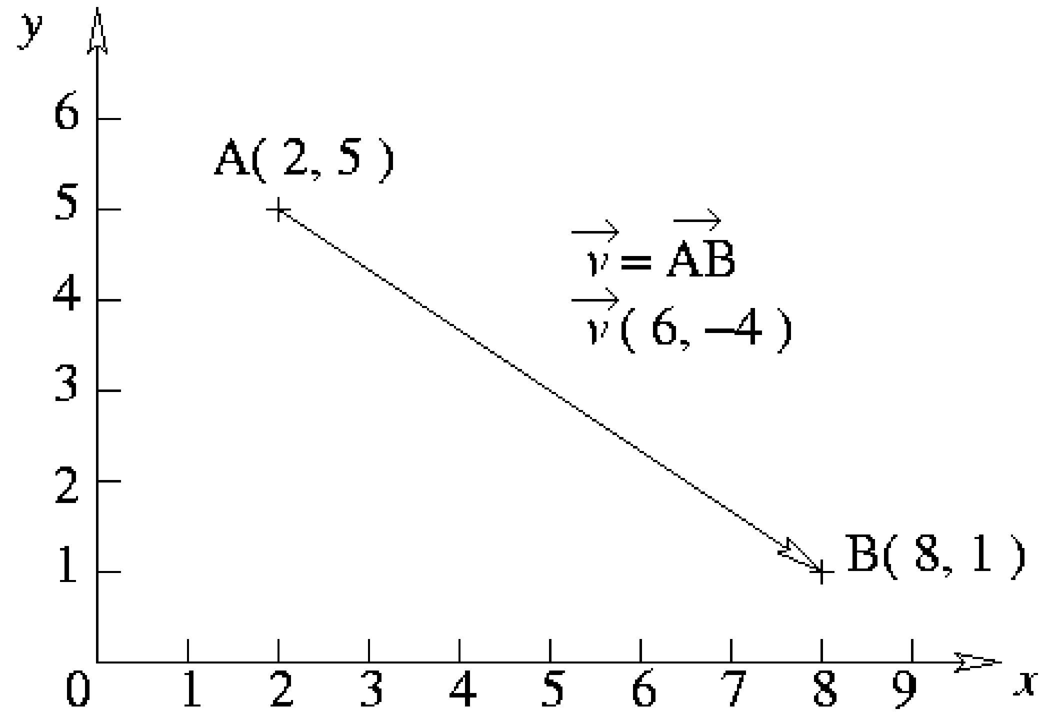
Différence

- Plan du module
  - introduction
  - représentations des objets
  - courbes et surfaces
  - interpolation vs approximation
  - transformations & projections
  - découpage

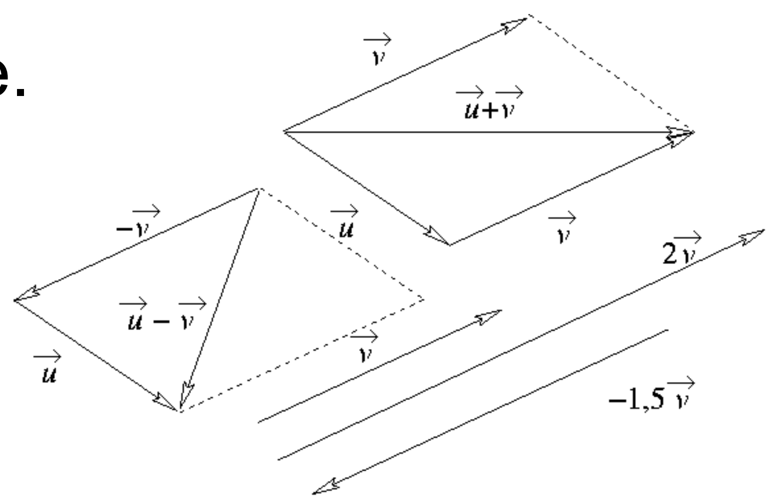
1) Vecteur

- coordonnées (x, y, ...) telles que

$$x = B.x - A.x$$
$$y = B.y - A.y$$
$$\ldots$$



- opérations :
  - addition ;
  - soustraction ;
  - multiplication par un scalaire.



Définition de la norme d'un vecteur

Dans un repère orthonormé, la norme d'un vecteur est :

$$\|\vec{u}\| = \sqrt{{u_x}^2 + {u_y}^2 + \ldots}$$

Propriétés de la norme d'un vecteur

- la norme d'un vecteur $\overrightarrow{AB}$ est la ***distance*** de A à B
- un vecteur de norme 1 est dit ***normé***.
- pour tout vecteur $\vec{u}$ non nul, il existe un vecteur normé de même direction :

$$\vec{u}_{normé} = \frac{\vec{u}}{\|\vec{u}\|}$$

Définition du produit scalaire de deux vecteurs

Dans un repère orthonormé, le produit scalaire associe deux vecteurs à un nombre réel :

$$\vec{u} \cdot \vec{v} = u_x \times v_x + u_y \times v_y + \ldots$$

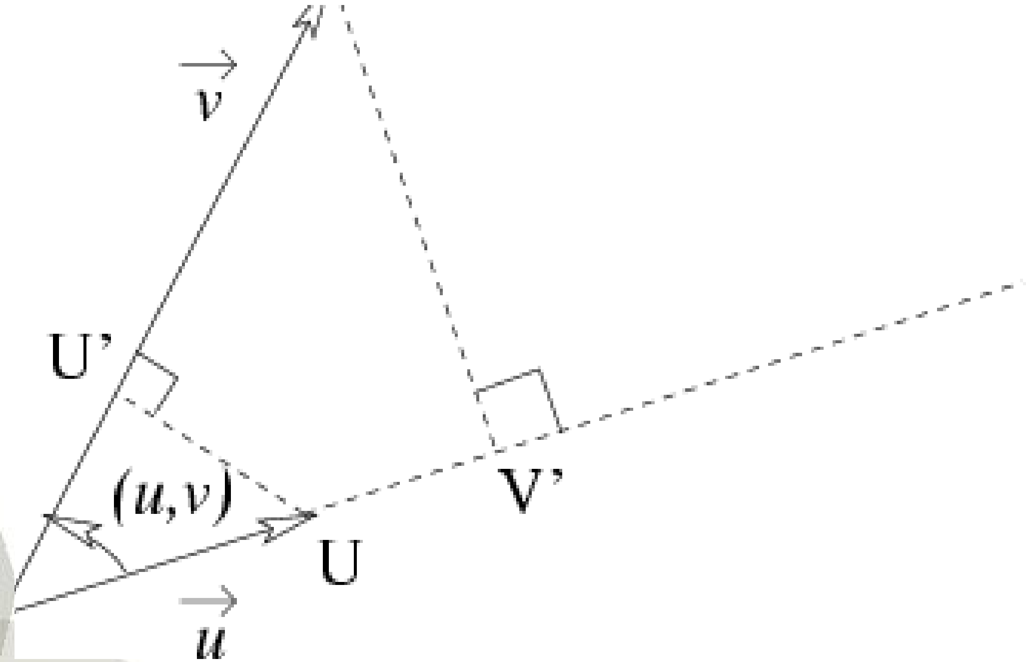
Propriétés du produit scalaire de deux vecteurs

- symétrie : $\vec{u} \cdot \vec{u'} = \vec{u'} \cdot \vec{u}$
- distributivité : $(\vec{u_1} + \vec{u_2}) \cdot \vec{u_3} = \vec{u_1} \cdot \vec{u_3} + \vec{u_2} \cdot \vec{u_3}$
- homogénéité : $(\alpha \vec{u}) \cdot \vec{u'} = \alpha (\vec{u} \cdot \vec{u'})$
- lien avec la norme : $\vec{u} \cdot \vec{u} = \|\vec{u}\|^2$

Application du produit scalaire dans le plan

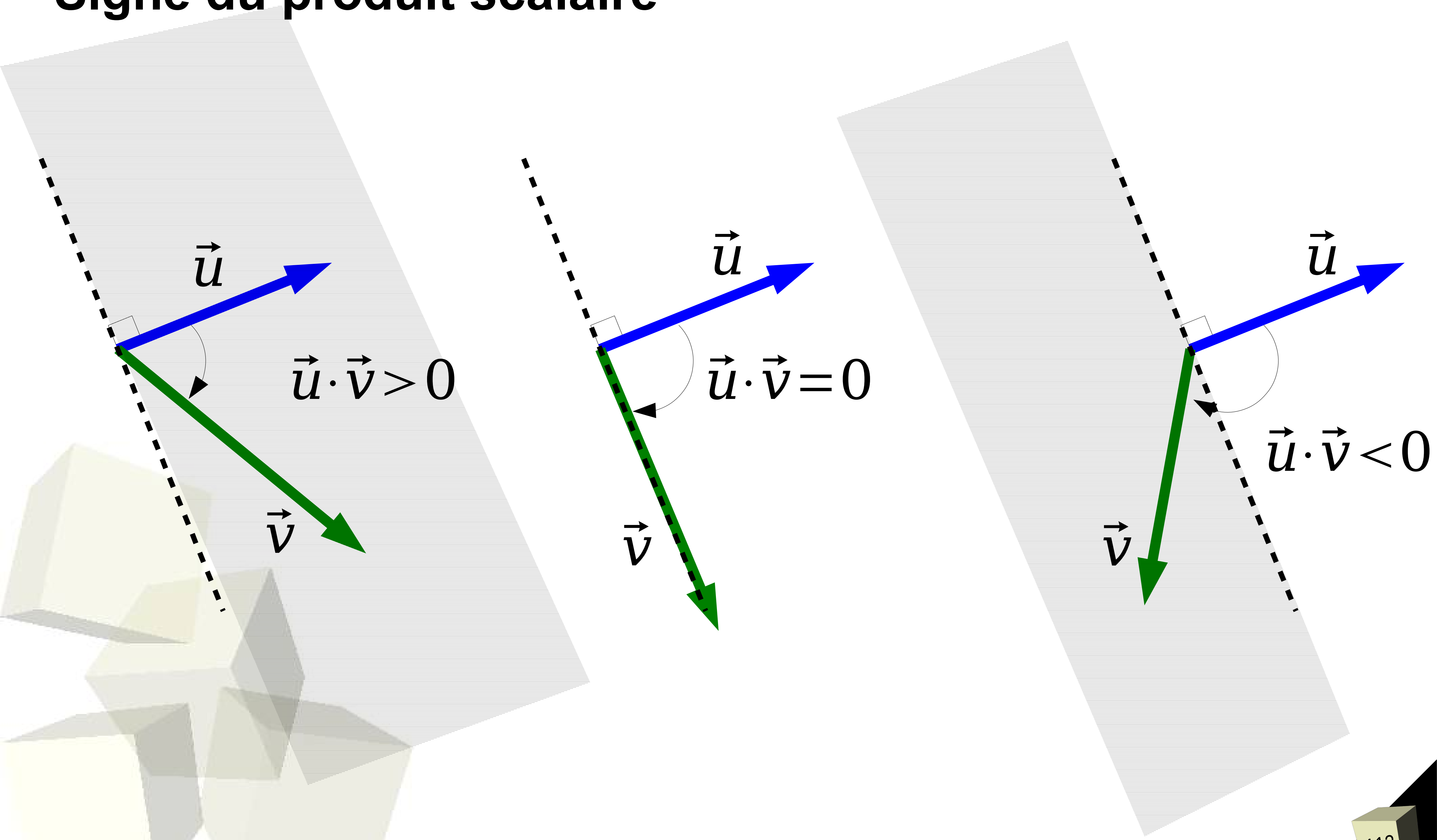
$$\vec{u}\cdot\vec{v}=\|\vec{u}\|\left(\|\vec{v}\|\cos(\vec{u},\vec{v})\right)=\overrightarrow{OU}\cdot\overrightarrow{OV}'$$

$$\vec{u}\cdot\vec{v}=\|\vec{v}\|\left(\|\vec{u}\|\cos(\vec{u},\vec{v})\right)=\overrightarrow{OV}\cdot\overrightarrow{OU}'$$



-> calcul de l'angle entre 2 vecteurs

Signe du produit scalaire

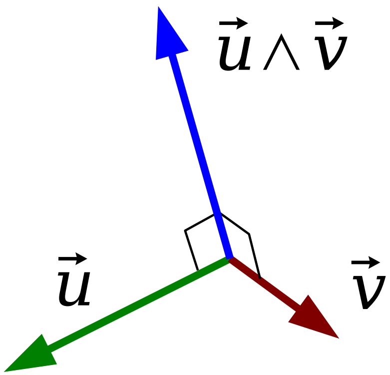


Définition du produit vectoriel de deux vecteurs

Dans un repère orthonormé, le produit vectoriel associe deux vecteurs à un vecteur résultat :

$$\vec{u} \wedge \vec{v} = \begin{bmatrix} u_y \times v_z - u_z \times v_y \\ u_z \times v_x - u_x \times v_z \\ u_x \times v_y - u_y \times v_x \end{bmatrix}$$

->le vecteur résultat est **orthogonal** aux deux premiers (utile pour les repères, les normales, ...)



2) Matrices

$$\begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ \cdots & \cdots & \cdots \end{bmatrix}$$ = collection de vecteurs

Opérations :

- addition :

$$\begin{bmatrix} a_0 & a_1 \\ b_0 & b_1 \end{bmatrix} + \begin{bmatrix} t_0 & t_1 \\ u_0 & u_1 \end{bmatrix} = \begin{bmatrix} a_0+t_0 & a_1+t_1 \\ b_0+u_0 & b_1+u_1 \end{bmatrix}$$

- multiplication par un scalaire :

$$n \times \begin{bmatrix} t_0 & t_1 \\ u_0 & u_1 \end{bmatrix} = \begin{bmatrix} n \times a_0 & n \times a_1 \\ n \times b_0 & n \times b_1 \end{bmatrix}$$

- multiplication par une matrice :

$$\begin{bmatrix} a_0 & a_1 \\ b_0 & b_1 \end{bmatrix} \times \begin{bmatrix} t_0 & t_1 \\ u_0 & u_1 \end{bmatrix} = \begin{bmatrix} (a_0 \times t_0 + a_1 \times u_0) & (a_0 \times t_1 + a_1 \times u_1) \\ (b_0 \times t_0 + b_1 \times u_0) & (b_0 \times t_1 + b_1 \times u_1) \end{bmatrix}$$

Opérations supplémentaires sur les matrices

- déterminant, cofacteurs ;
- inversion (délicat), peut ne pas être possible (revient au problème de recherche de solutions d'équations) ;
- transposée

Autres besoins :

- *équations paramétriques de droites et courbes ;*
- *équations implicites de plans ;*
- *changements de repères.*

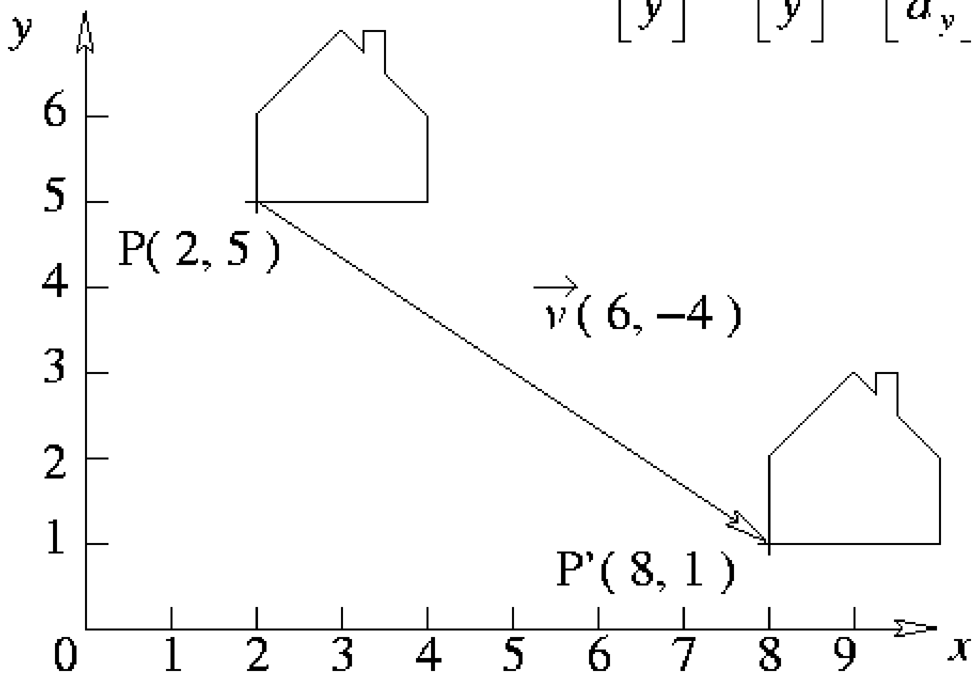
- ~~quelques rappels de maths ;~~
- transformations de l'espace ;
- projections ;
- visualisation ;
- parties cachées.

- survol en 2D ;
- coordonnées homogènes ;
  - translation ;
  - mise à l'échelle (*scaling*) ;
  - rotation ;
  - réflexions ;
- compositions de transformations.

1) Survol en 2D

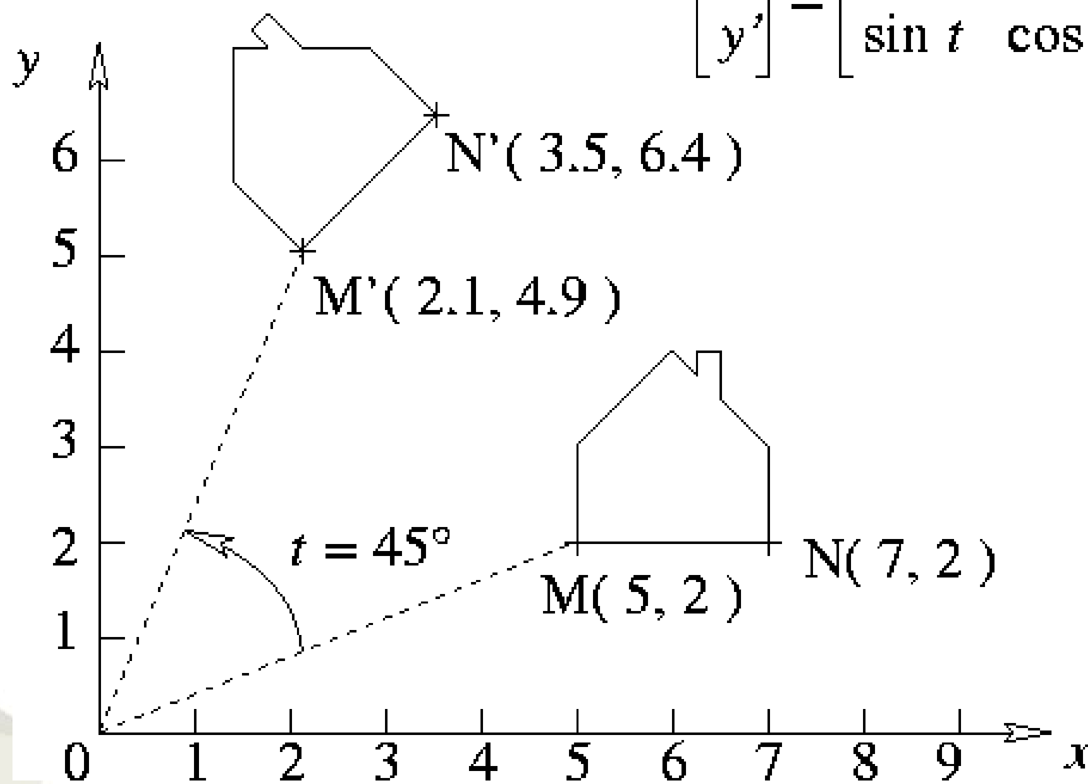
Translation de vecteur $\vec{v}$ du point P : $P' = P + \vec{v}$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$



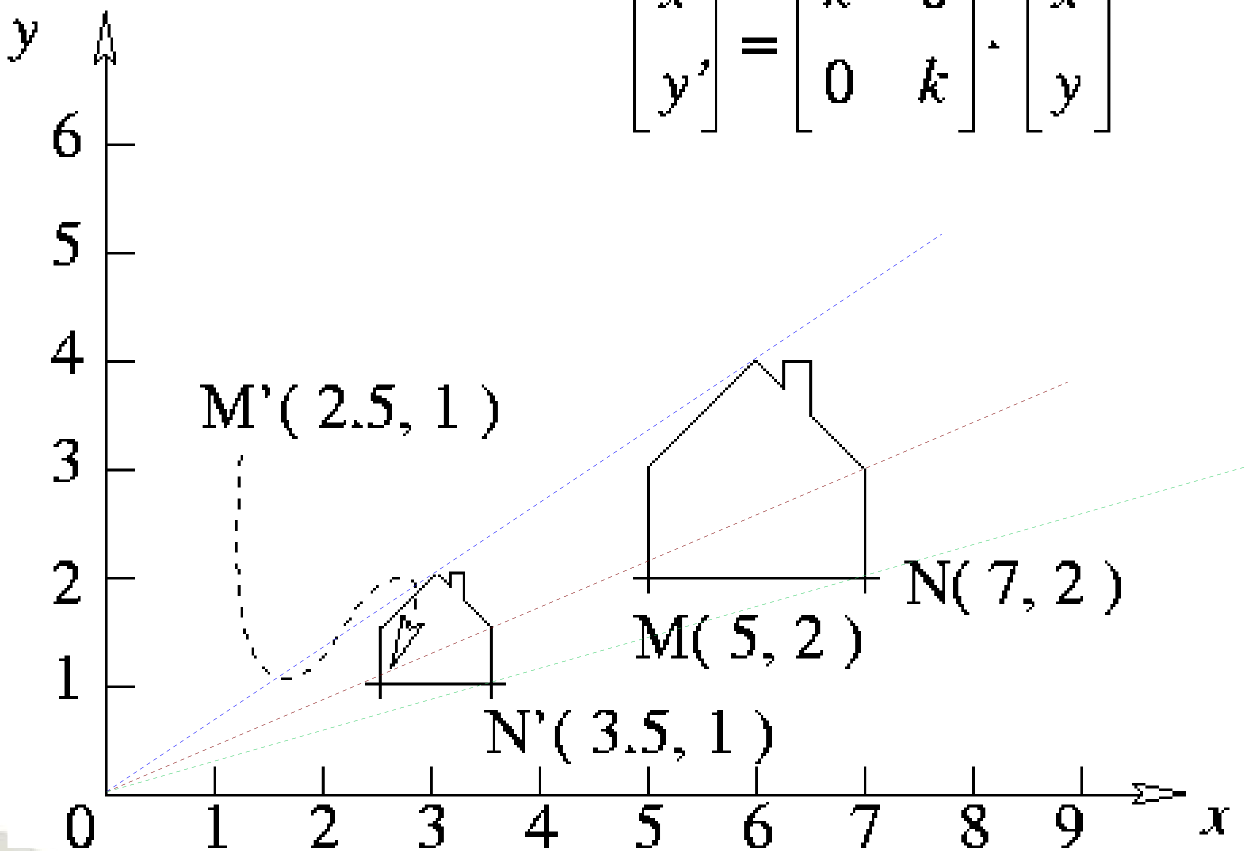
Rotation de centre *O* et d'angle *t* : $P' = R_t \cdot P$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$



Homothétie de centre *O* et de rapport *k* : $P' = k \cdot P$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$



Les mêmes relations peuvent êtres écrites en 3D. Il se pose alors 3 problèmes :

- opérations différentes pour les translations (addition de matrices) ;
- pb de commutativité ;
- pb de la rotation 3D (définition de centre, continuité).

- ~~survol en 2D ;~~
- coordonnées homogènes ;
  - translation ;
  - mise à l'échelle (scaling) ;
  - rotation ;
  - réflexions ;
- compositions de transformations.

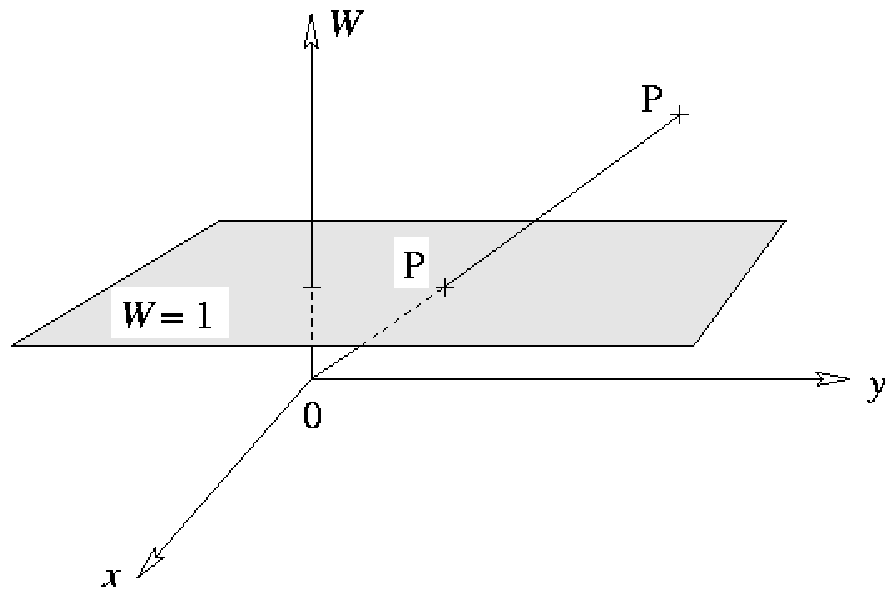
Pour résoudre le problème d'écriture des opérations : on passe en ***coordonnées homogènes***.

-> on ajoute une coordonnée, ***w***

Les coordonnées homogènes permettent de représenter toutes les transformations affines comme des produits de matrice.

Valeurs de ***w*** :

- *1 pour les points ;*
- *0 pour les vecteurs.*



Passage en coordonnées homogènes

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x_h \\ y_h \\ z_h \\ w_h \end{bmatrix} \quad \text{avec} \quad \begin{matrix} x = x_h / w_h \\ y = y_h / w_h \\ z = z_h / w_h \end{matrix}$$

Translation

$$P' = P + \vec{T} \rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}_{P'} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}_T \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_P$$

Mise à l'échelle

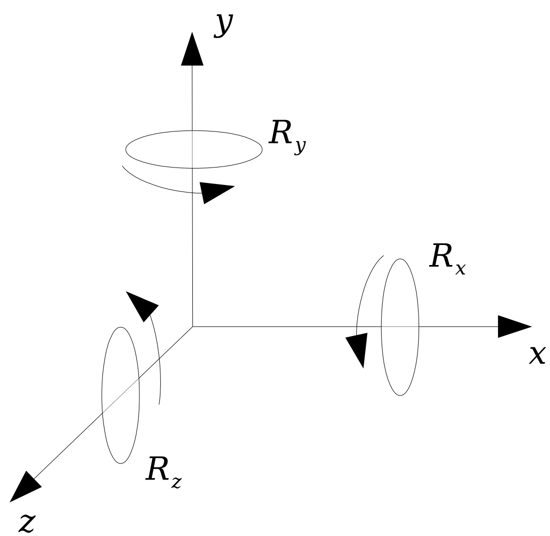
$$P' = S \cdot P \rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}_{P'} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_S \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_P$$

Réflexion

$$P' = M \cdot P \rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}_{P'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_M \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_P$$

(par rapport à *y*)

Rotation 3D en coordonnées homogènes



Rotation autour d'un axe *(x, y, z)* d'angle α :

$$R=\begin{bmatrix} tx^2+c & txy+sz & txz-sy & 0 \\ txy-sz & ty^2+c & tyz+sx & 0 \\ txz+sy & tyz-sx & tz^2+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ avec } \begin{matrix} s=\sin\alpha \\ c=\cos\alpha \\ t=1-\cos\alpha \end{matrix}$$

Rotation autour de *(Ox)*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation autour de *(Oy)*

$$\begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation autour de *(Oz)*

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- ~~survol en 2D ;~~
- ~~coordonnées homogènes ;~~
  - ~~translation ;~~
  - ~~mise à l'échelle (scaling) ;~~
  - ~~rotation ;~~
  - ~~réflexions ;~~
- compositions de transformations.

Les compositions s'écrivent comme des suites de transformations :

Exemple de 2 transformations : rotation puis mise à l'échelle.

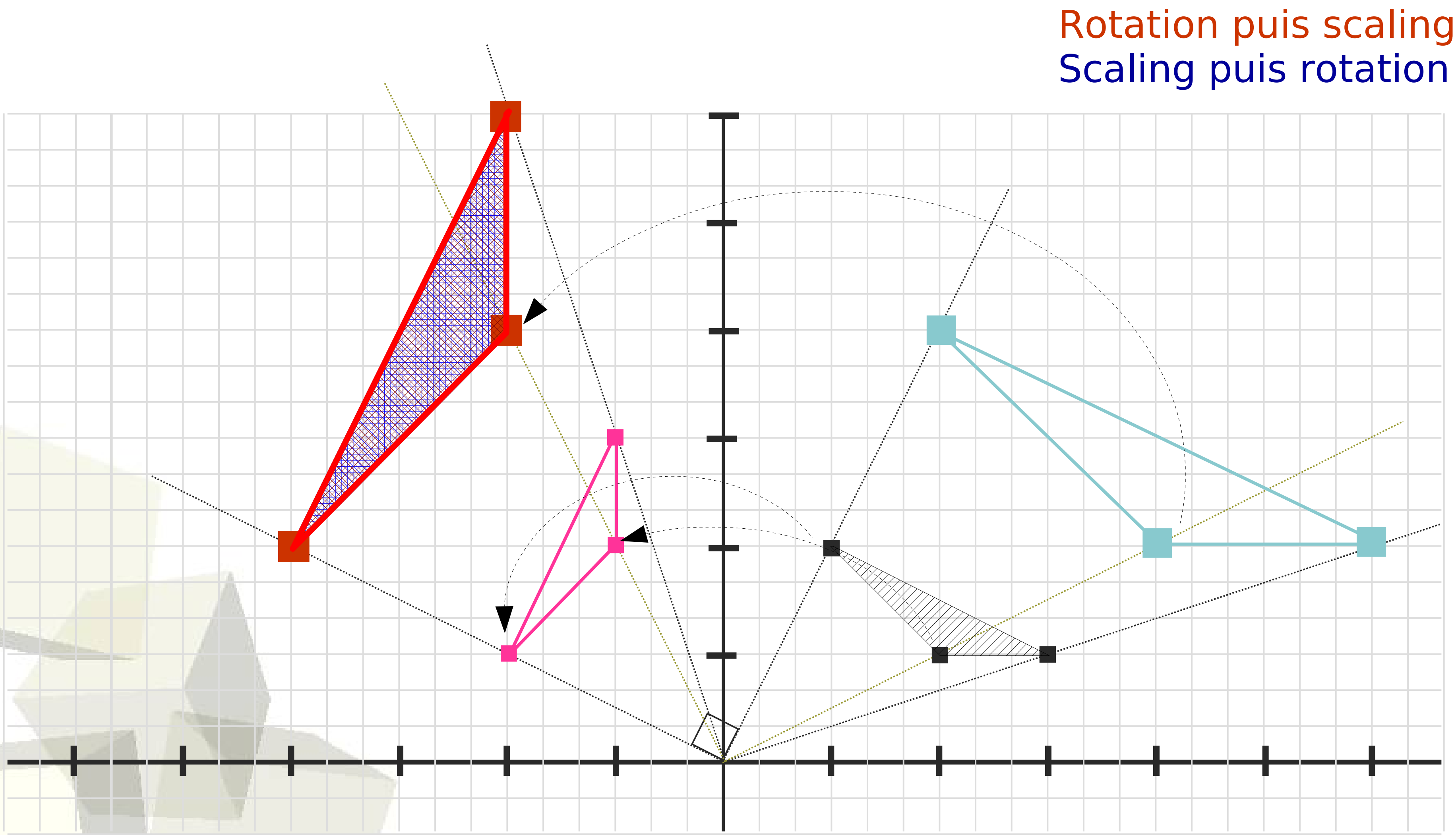
$$P' = R_\alpha P \qquad P'' = S_k P' \quad \rightarrow \quad P'' = (S_k R_\alpha) P$$

Problème : translation puis mise à l'échelle :

$$P' = P + \vec{T} \qquad P'' = S_k P' \quad \rightarrow \quad P'' = S_k P + S_k \vec{T}$$

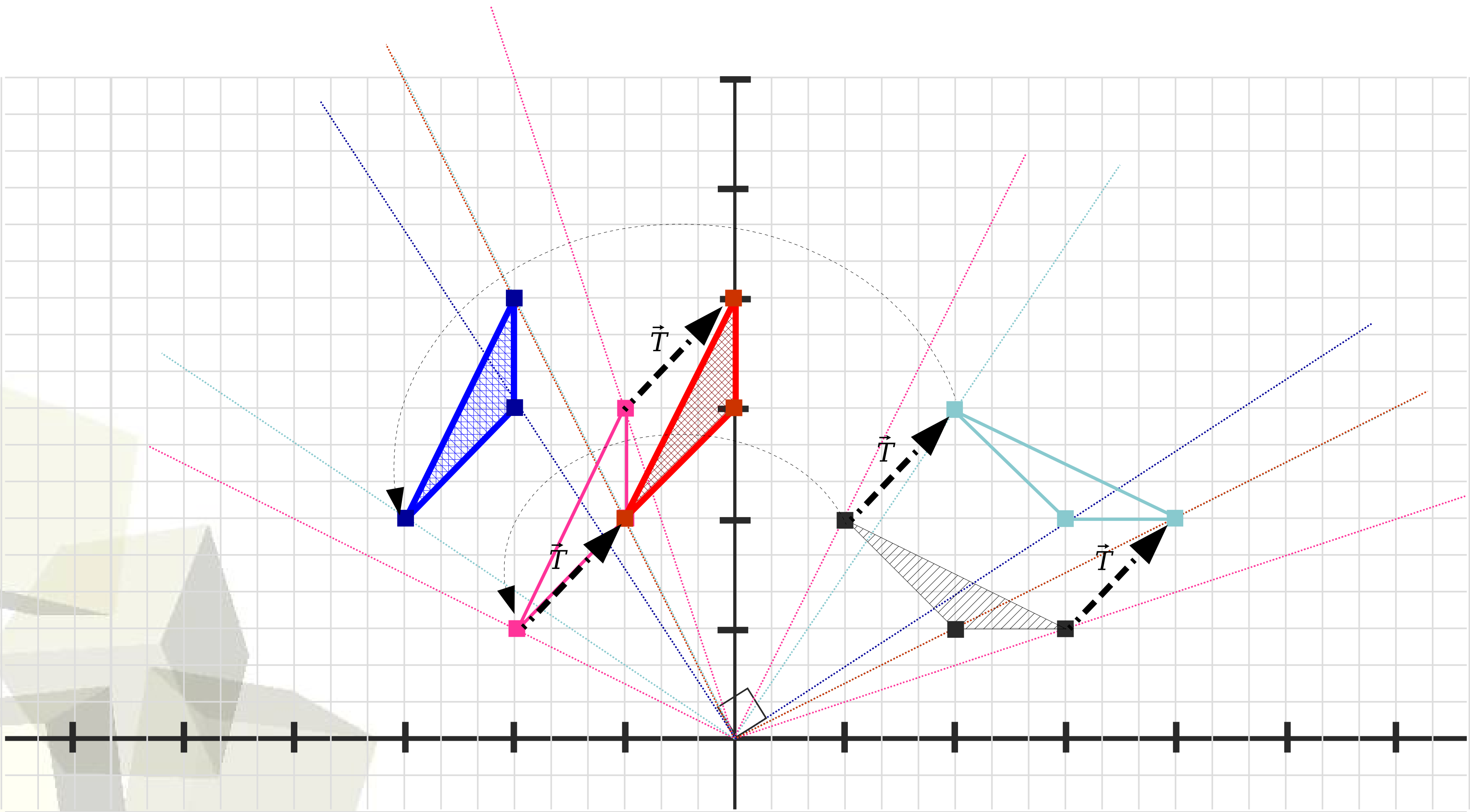
(par contre elles sont homogènes)

Pb de commutativité des transformations ? l'ordre est-il important ?

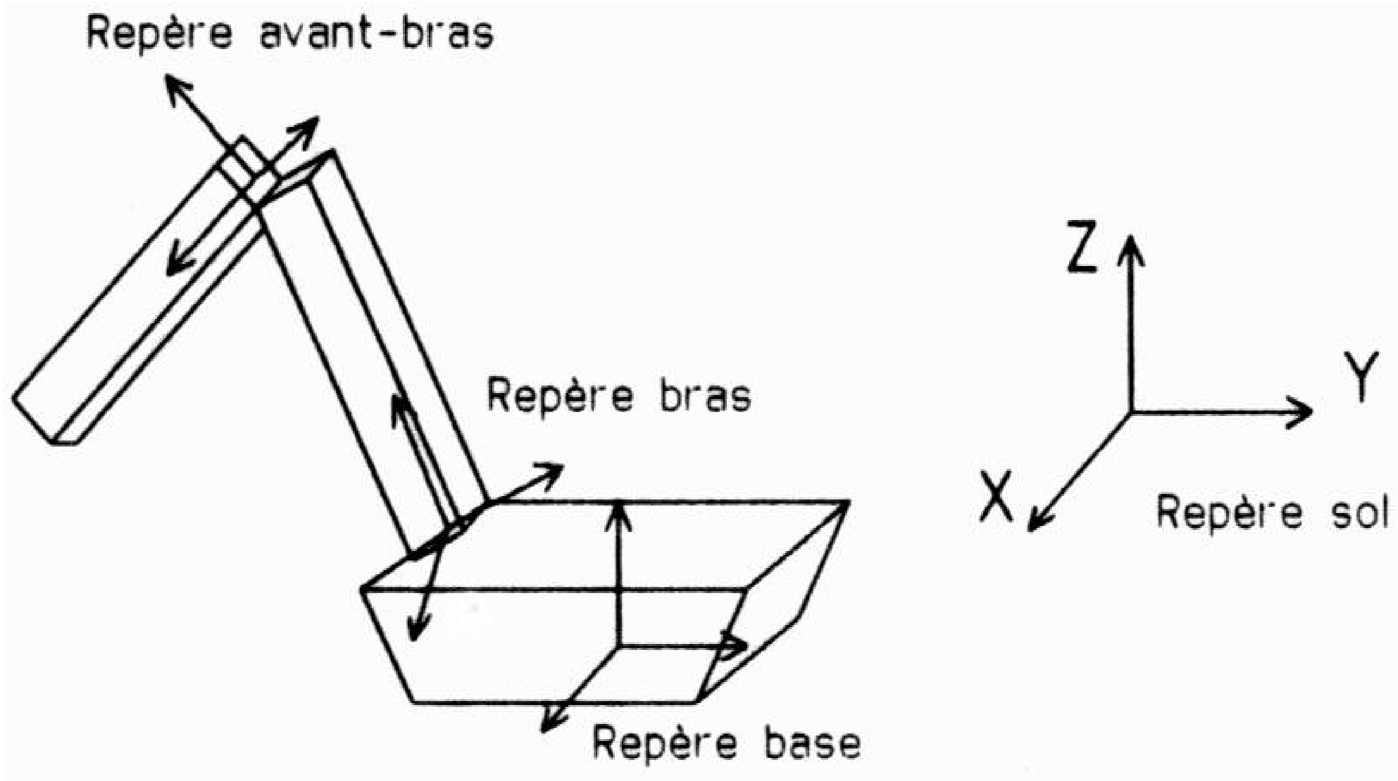


Pb de commutativité ?

Rotation puis translation
Translation puis rotation

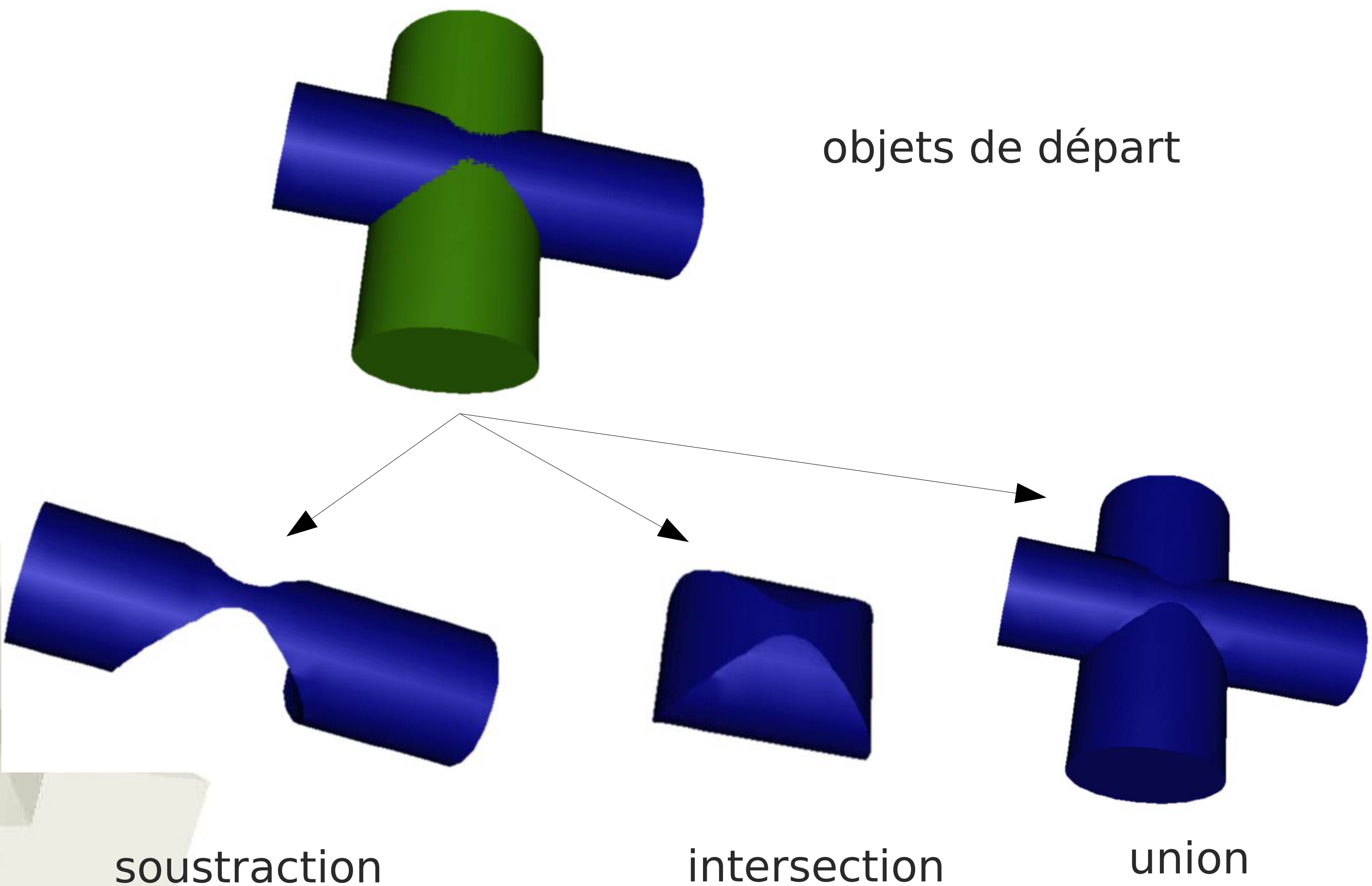


Exemples d'utilisation (1) : principe de modélisation



Exemples d’utilisation (2) : principe de visualisation

- arbre graphique ou graphe de scène
- CSG (*Constructive Solid Geometry*)

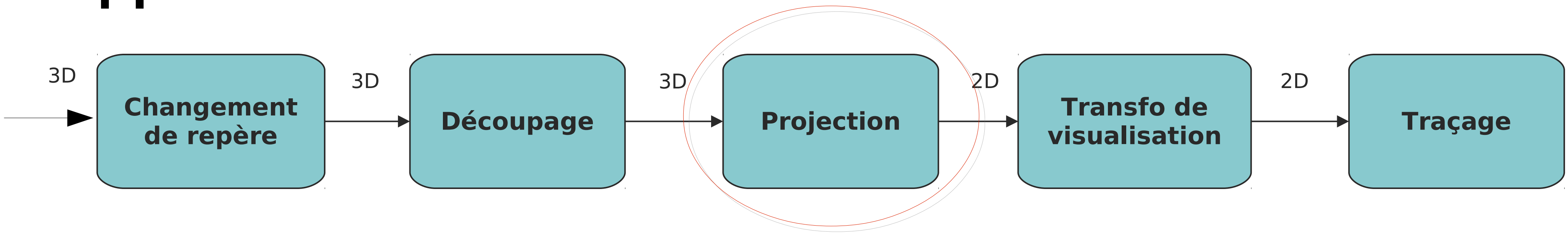


- ~~survol en 2D ;~~
- ~~coordonnées homogènes ;~~
  - ~~translation ;~~
  - ~~mise à l'échelle (scaling) ;~~
  - ~~rotation ;~~
  - ~~réflexions ;~~
- ~~compositions de transformations.~~

Plan

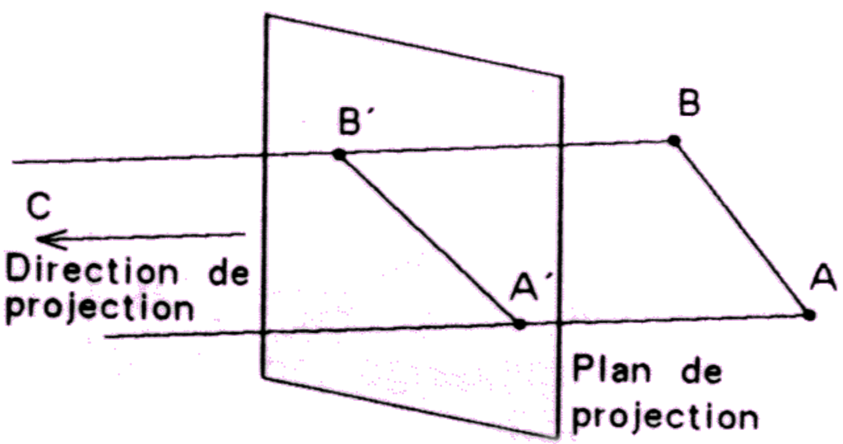
- ~~quelques rappels de maths ;~~
- ~~transformations de l’espace ;~~
- projections ;
- visualisation ;
- parties cachées.

Rappel

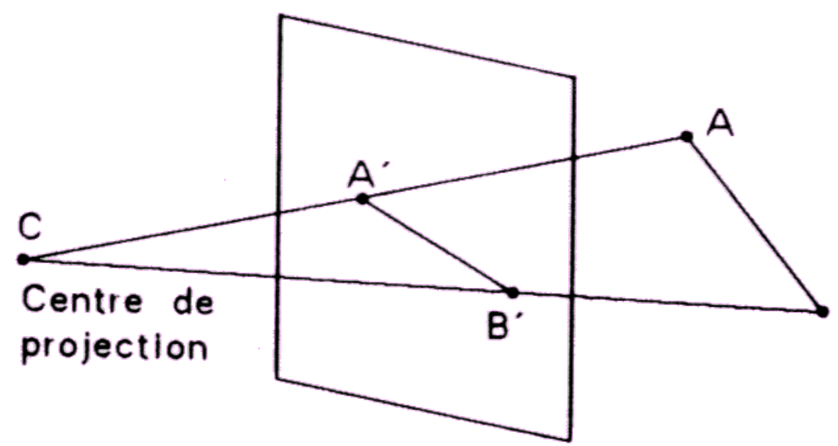


- changement de repère
- projection 3D - 2D

1) projections ***parallèles*** :
- orthographique
- oblique



2) projections ***perspectives*** :
- selon un axe principal
- générale



1) Projection parallèle

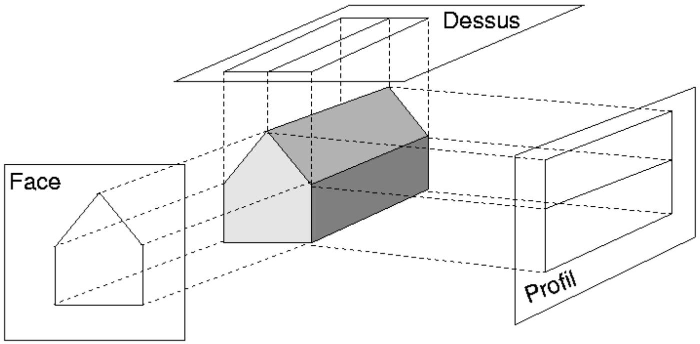
Il existe deux types de projections parallèles

- projection orthographique lorsque la direction de projection est perpendiculaire au plan de projection ;
- projection oblique sinon.

Propriétés géométriques des projections parallèles

- conservent le ***parallélisme*** des droites ;
- conservent ***les rapports des distances*** selon une direction donnée.

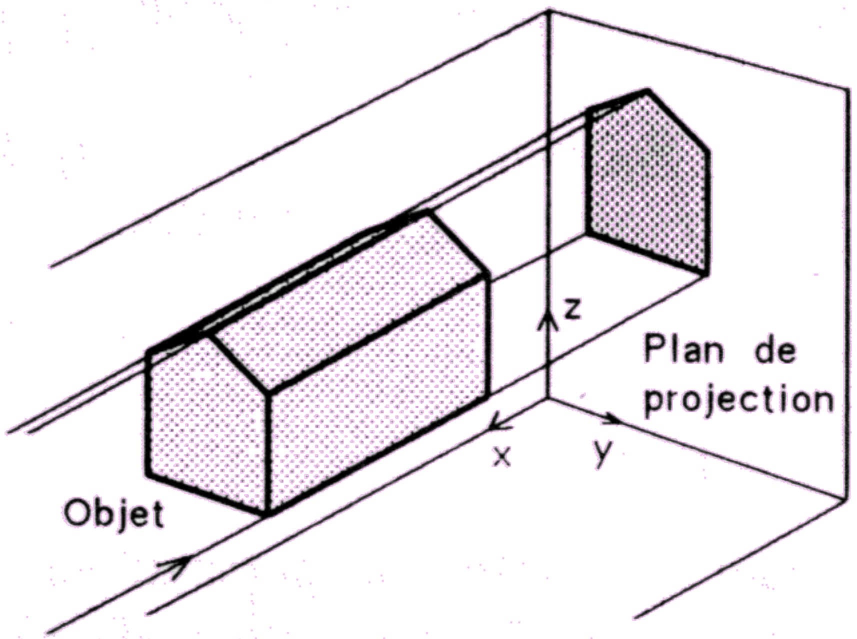
a) orthographique



éliminer Z → vue de dessus
éliminer Y → vue de côté
éliminer X → vue de face

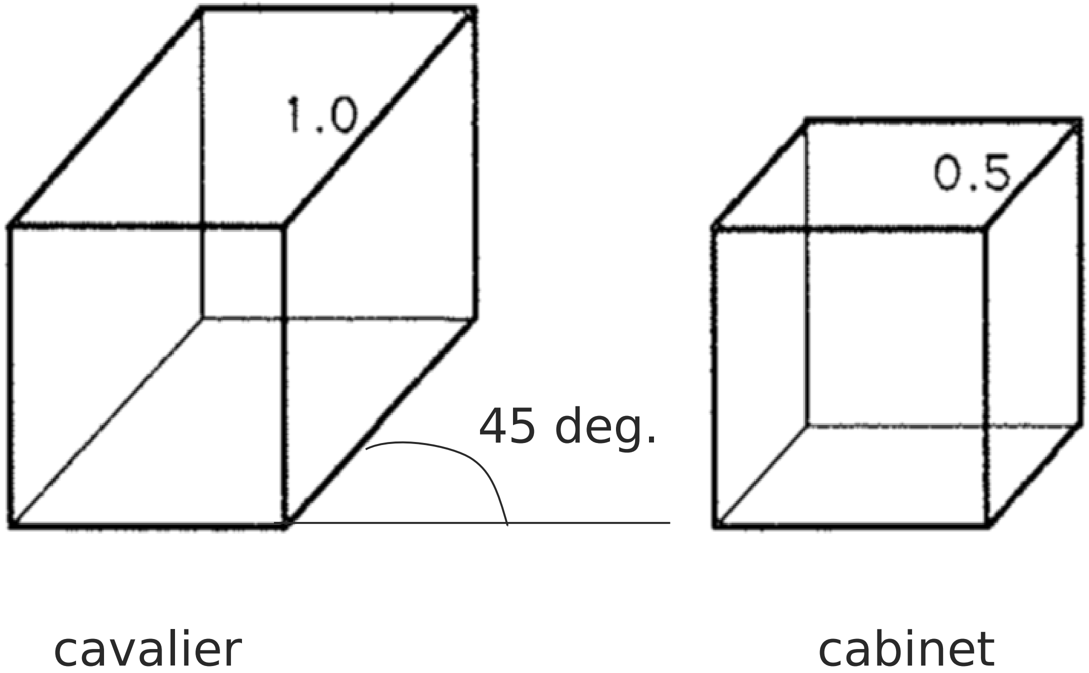
$$M_{\mathrm{orth},z}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$$

$$M_{\mathrm{orth},x}=\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$



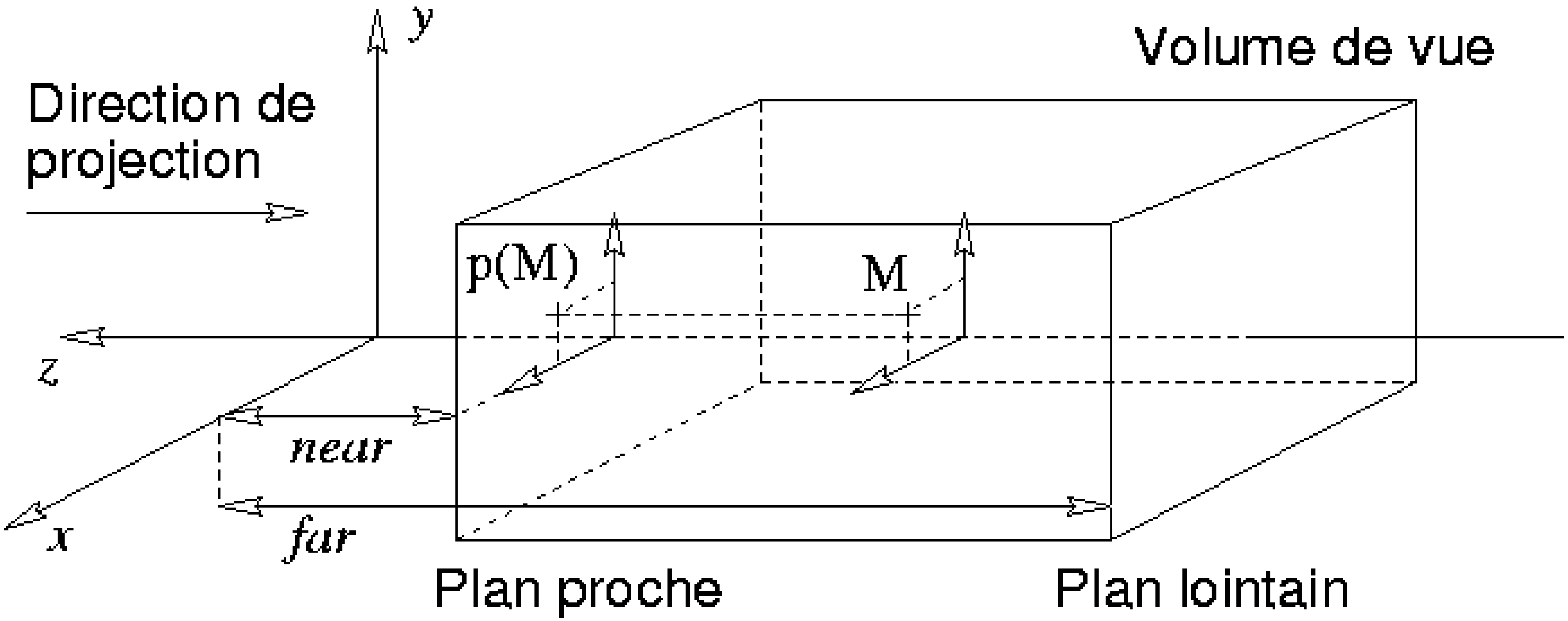
b) oblique

$$R_z\left(\frac{\pi}{4}\right)$$



$$M_{\text{axo}}=M_{\text{orth},x}\cdot R_z\cdot R_y\cdot R_x\cdot T$$

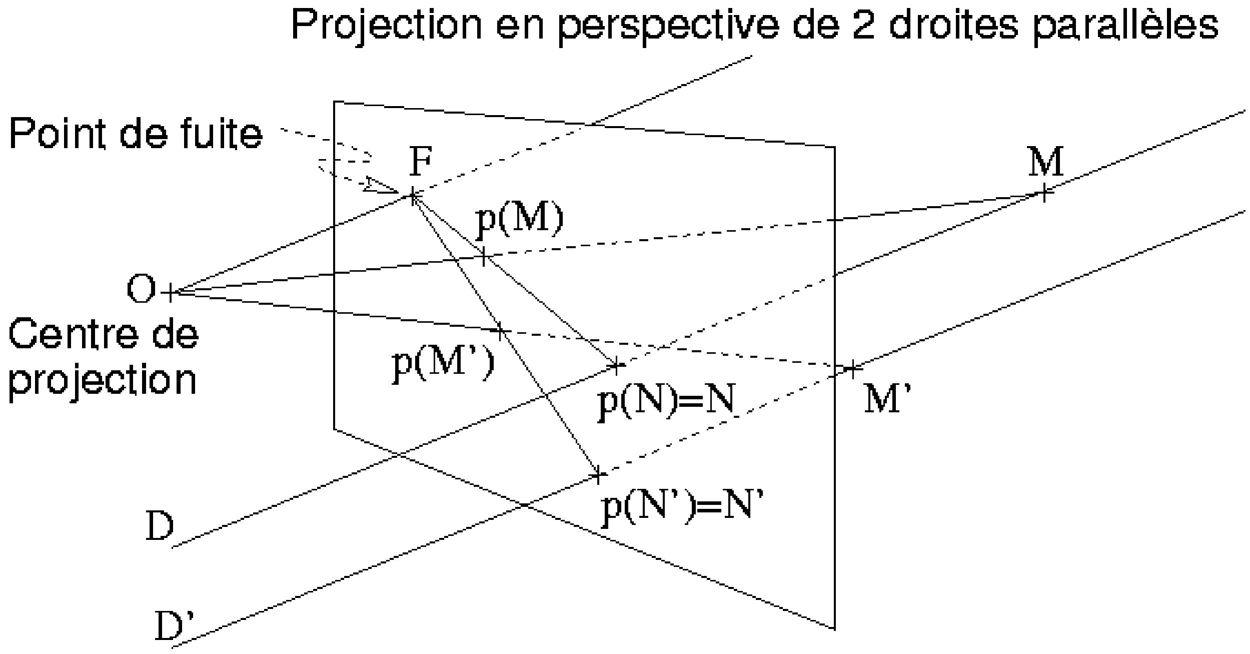
volume de vue en projection parallèle



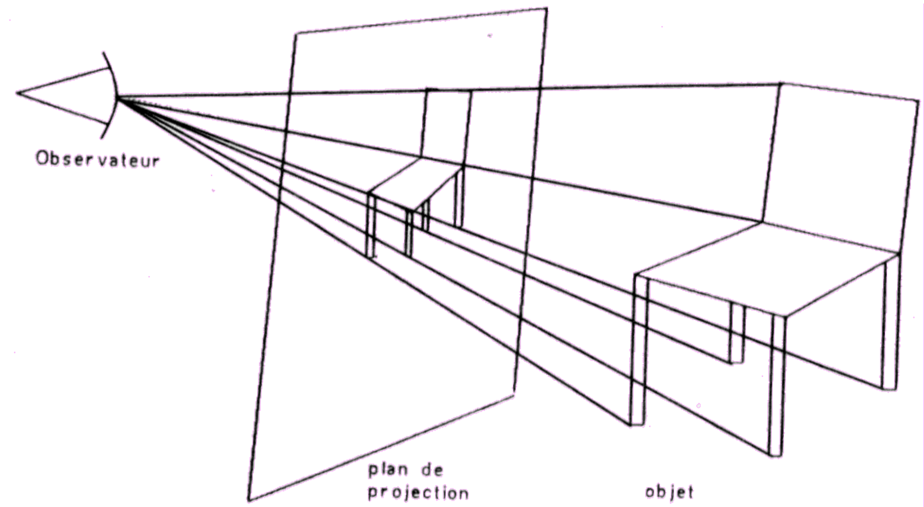
2) projection perspective

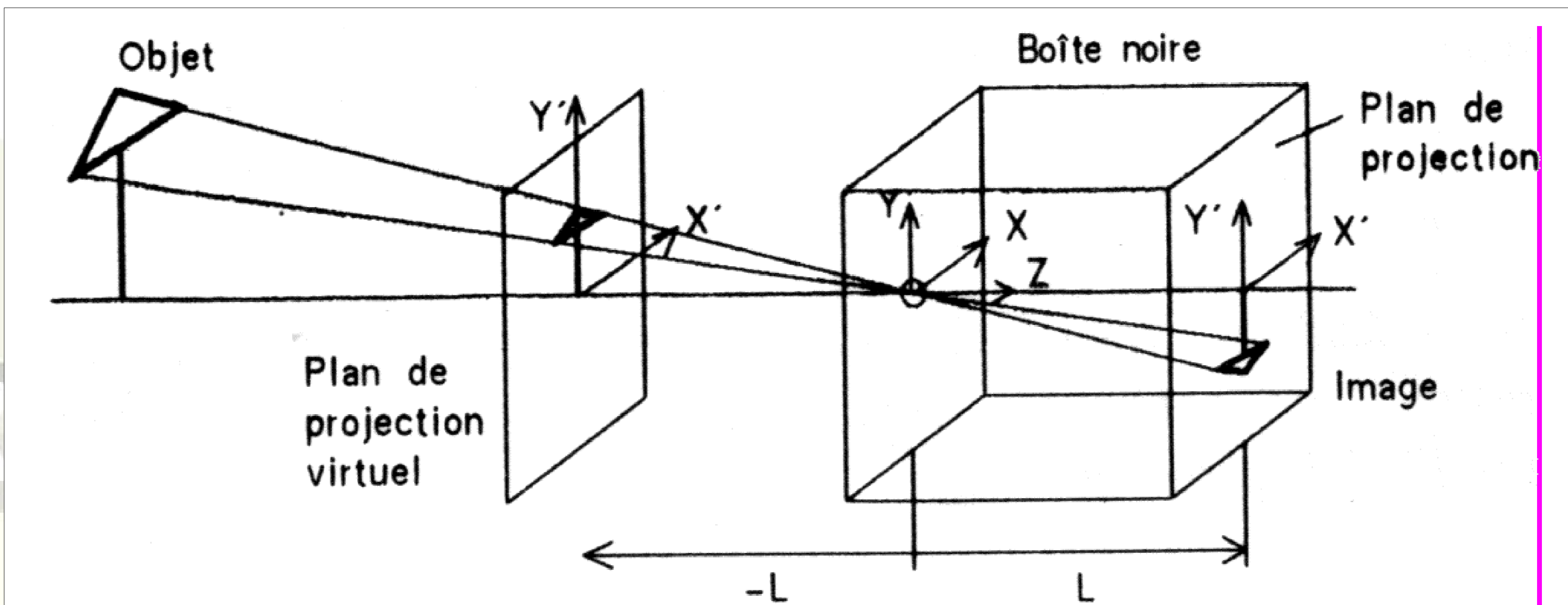
Définitions :

- l'image d'un point *M* par une projection en perspective sur le plan *P* de centre *O* est l'***intersection*** de la droite *(OM)* avec le plan *P* ;
- une projection en perspective dont le centre de projection est à l'***infini*** est une projection parallèle.



Analogie avec une caméra

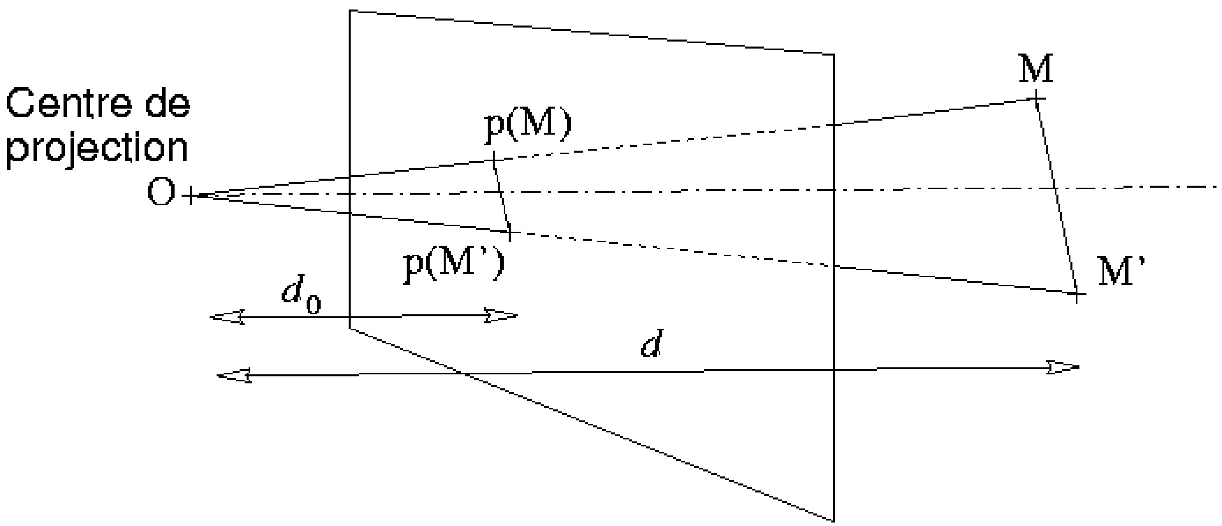




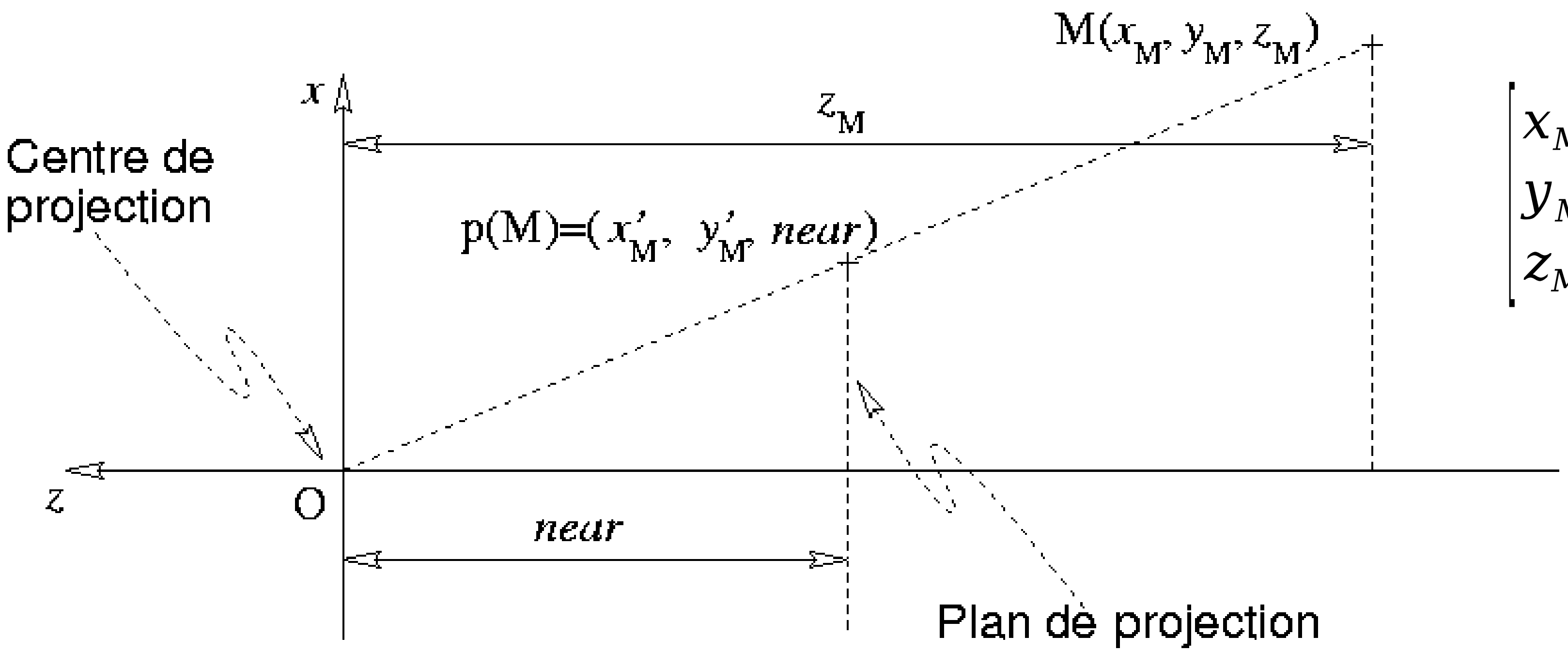
Propriétés géométriques des projections en perspective

- les projections **ne conservent pas** le parallélisme des droites non parallèles au plan de projection ;
- la taille d'un objet est **inversement proportionnelle à sa distance** au point de projection :

$$\left\|\overrightarrow{Proj(M)Proj(M')}\right\| = \left\|\overrightarrow{MM'}\right\| \times \frac{d_0}{d}$$



Coordonnées du point projeté en fonction de celles du point source, projection du point *M* sur le plan *near* :



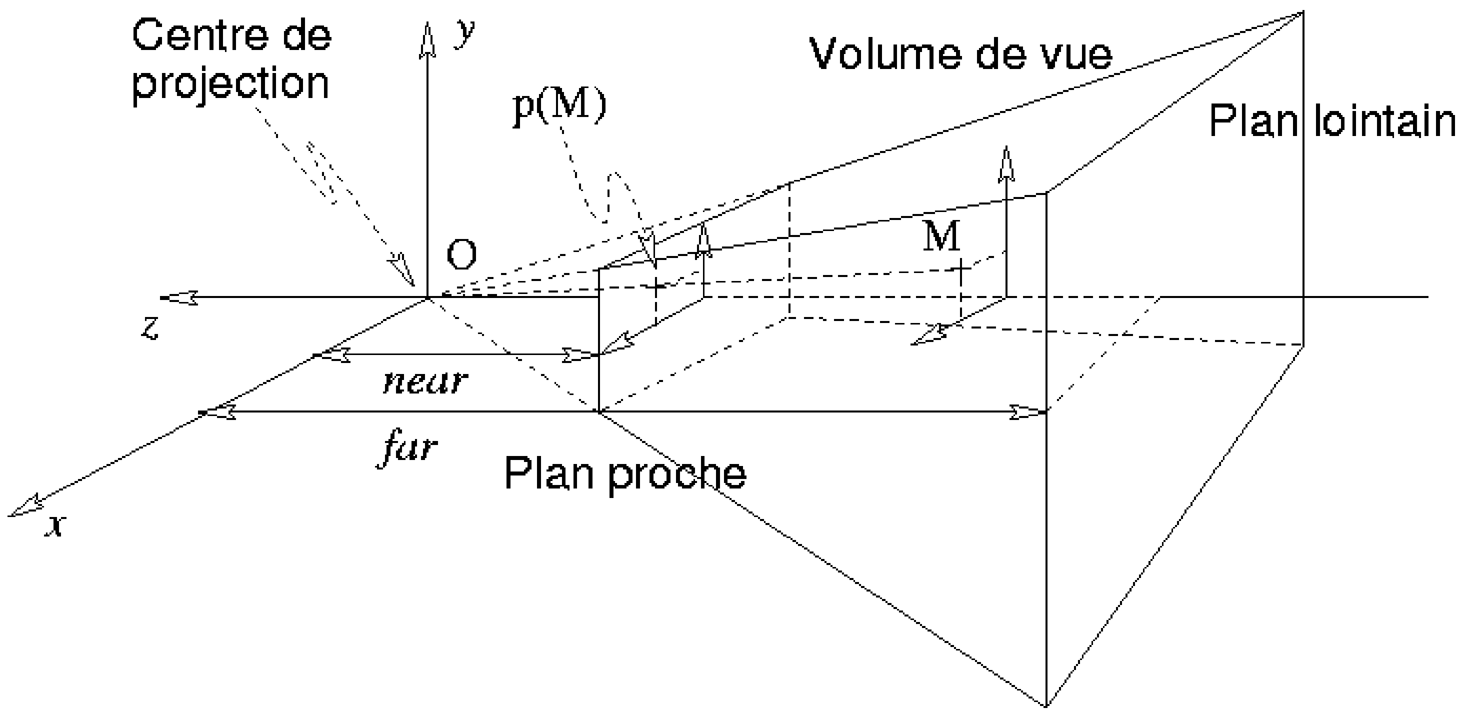
$$\begin{bmatrix} x_{M'} \\ y_{M'} \\ z_{M'} \end{bmatrix} = \begin{bmatrix} \dfrac{x_M}{(z_m/near)} \\ \dfrac{y_M}{(z_m/near)} \\ near \end{bmatrix}$$

Matrice en coordonnées homogènes de la projection

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}_{P'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{near} & 0 \end{bmatrix}_T \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_P$$

Volume de vue en projection perspective

Pour se ramener à un volume de vue canonique, on effectue une rotation et une translation du repère.



Calcul de la pseudo-profondeur dans une projection en perspective

- On conserve une valeur de la profondeur telle que deux points ayant la même projection soient distinguables.
- On utilise une fonction homogène avec celle de x et y :

$$M'_z=(a\cdot M_z+b)/(-M_z)$$

- et on choisit $M'_z=-1$ pour $M_z=-near$ et $M'_z=1$ pour $M_z=-far$

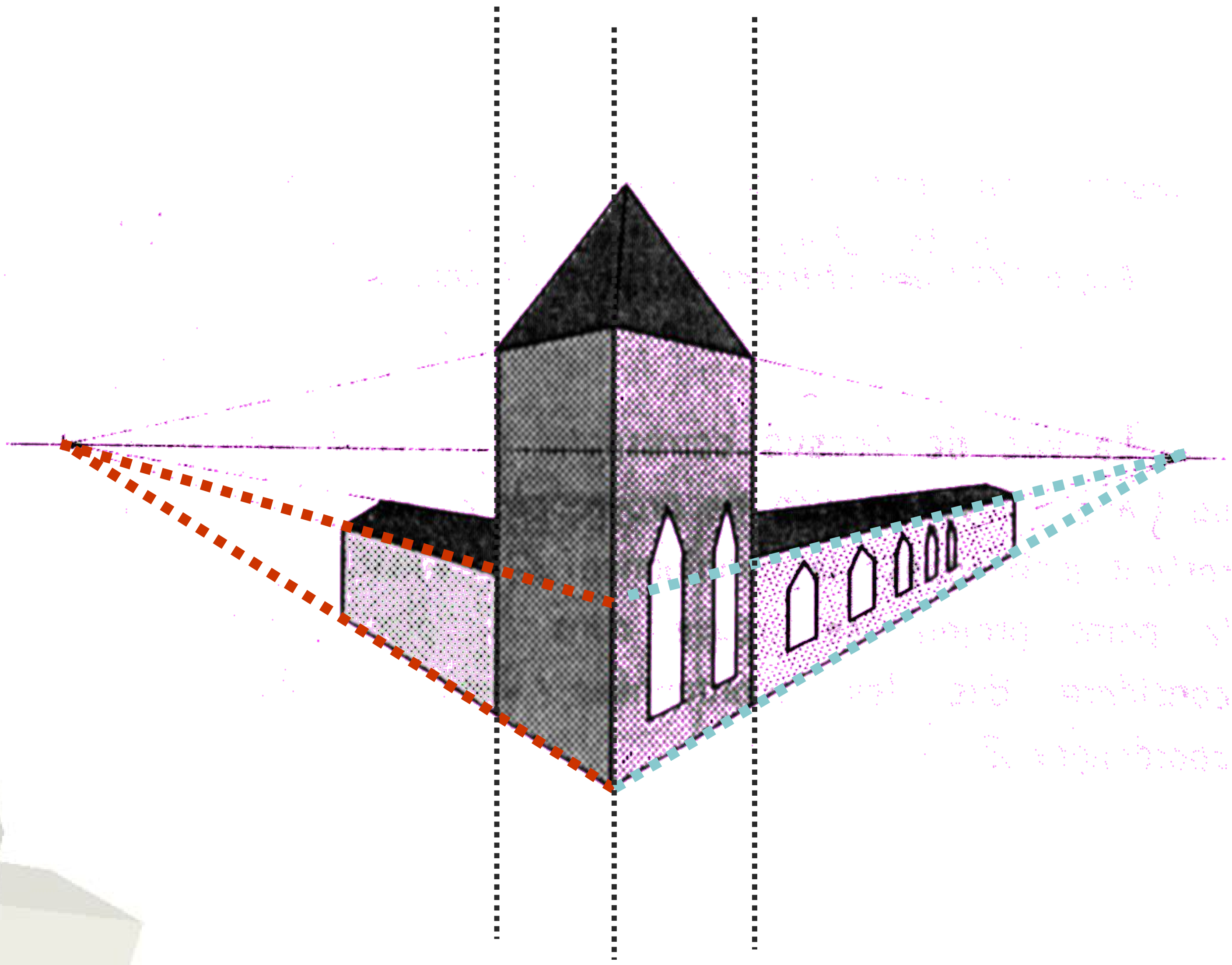
(On rend les faces avant et arrière du volume de vue coplanaires avec les faces du volume de vue canonique.)

- donc $a=\dfrac{-(far+near)}{(far-near)}$ et $b=-2\dfrac{(far\times near)}{(far-near)}$

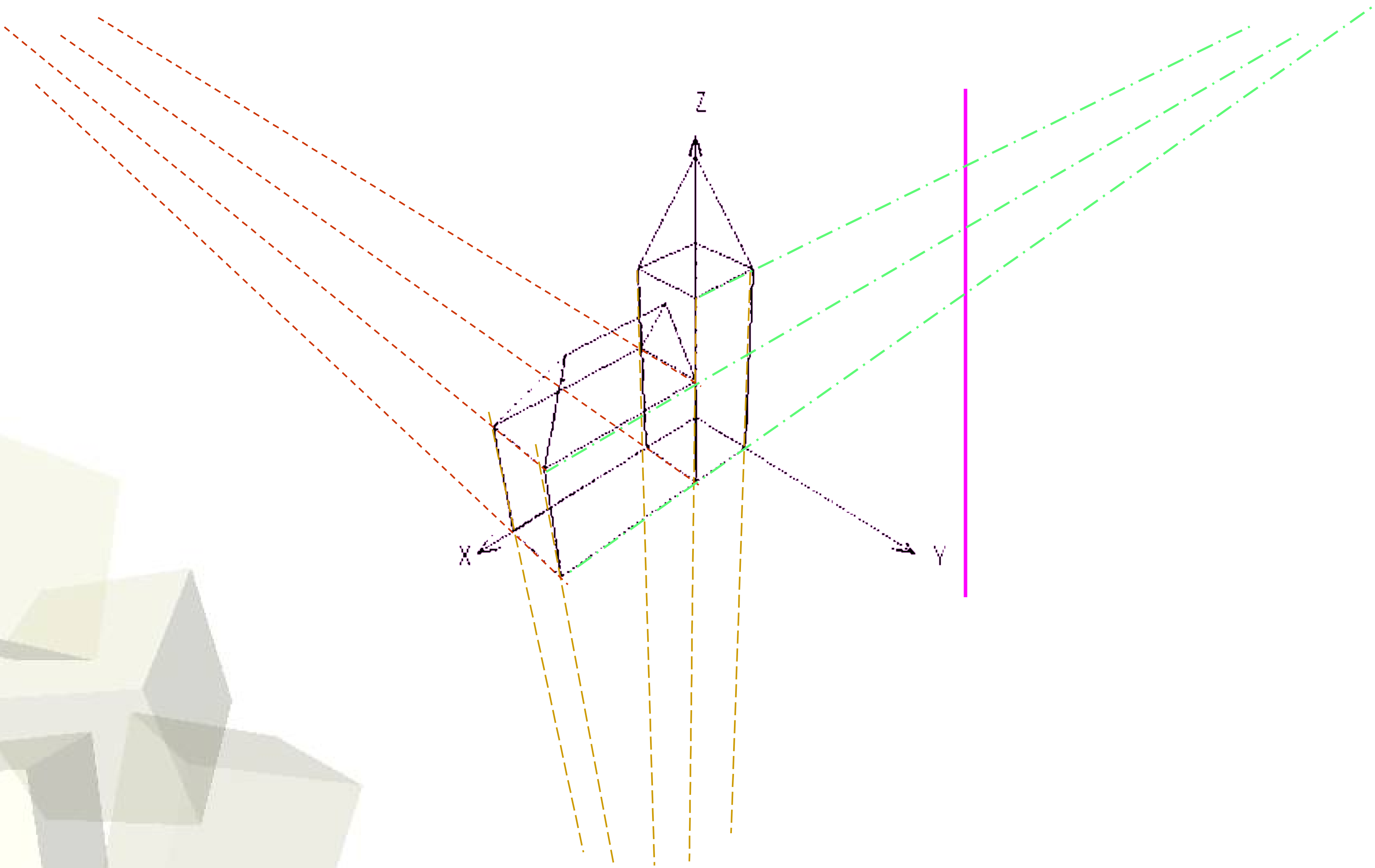
Points de fuite

- si une droite *D* coupe le plan de projection, il existe un point *F*, appelé ***point de fuite*** appartenant à la projection de toute droite parallèle à *D* (*cf.* diapo précédente) ;
- on différencie les projections en perspective par le nombre de points de fuite pour les directions des axes du repère (le nombre d'intersections des axes de coordonnées avec le plan) ;
- en général on a deux points de fuite (caméra verticale non parallèle à un des axes) ;
- le troisième point de fuite n'augmente pas significativement le réalisme.

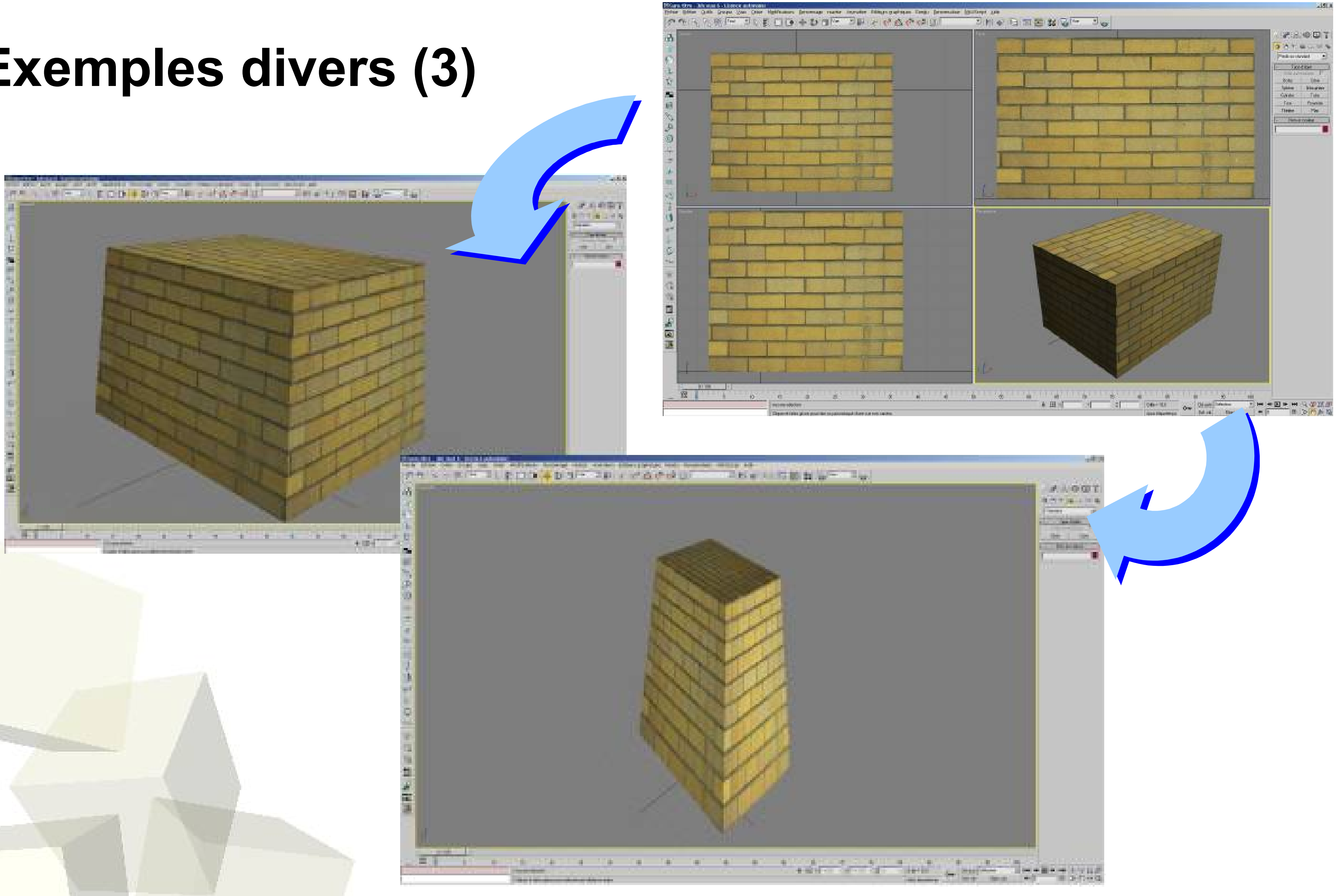
Exemple (1) : 2 points de fuite



Exemple (2) : 3 points de fuite



Exemples divers (3)

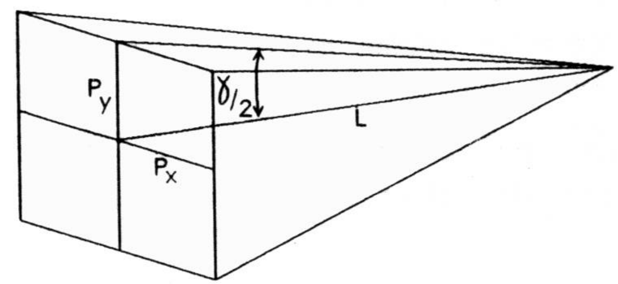


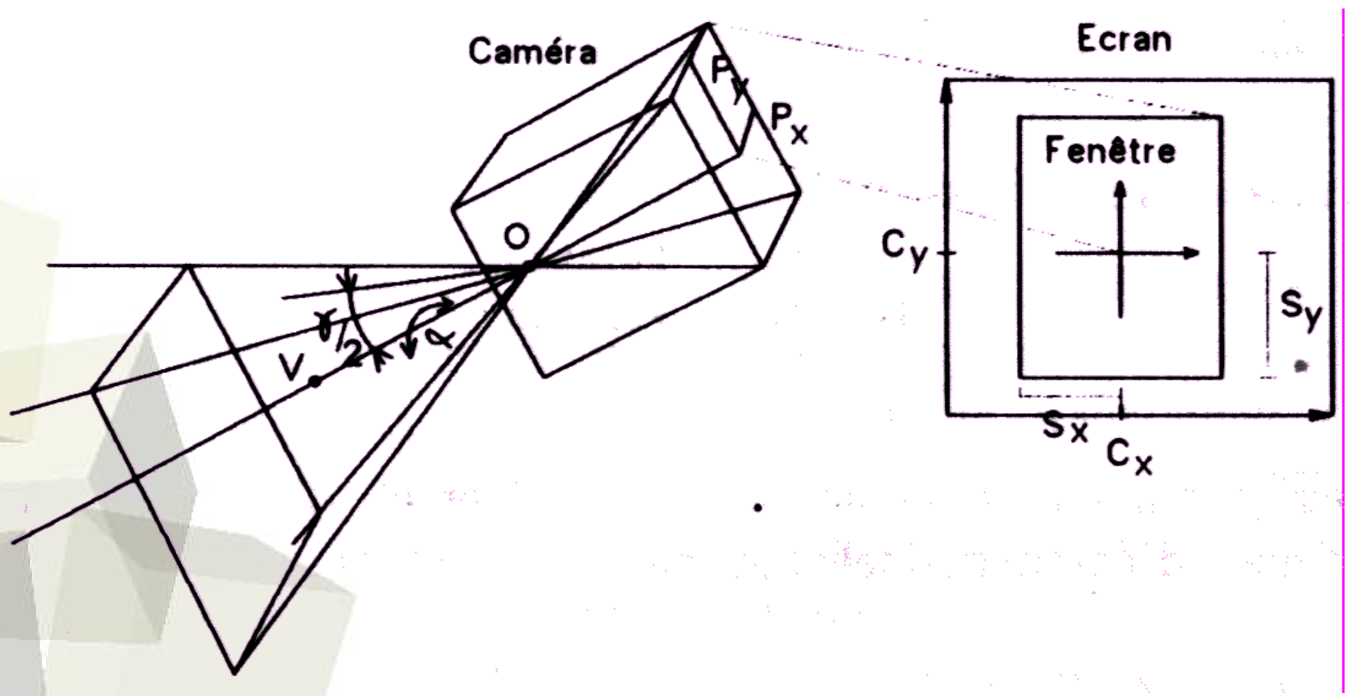
Plan

- ~~quelques rappels de maths ;~~
- ~~transformations de l'espace ;~~
- ~~projections ;~~
- visualisation ;
- parties cachées.

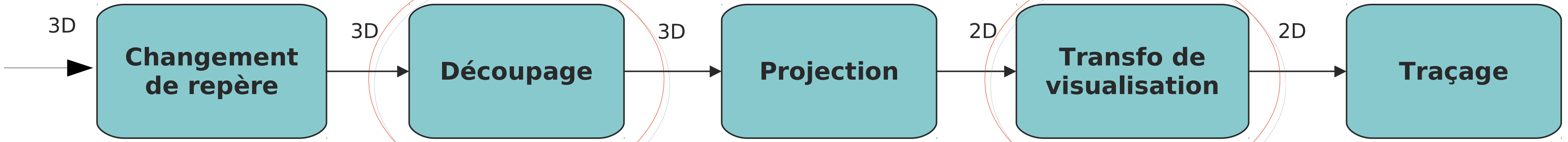
si on reprend l'analogie de la caméra virtuelle :

$$\tan(\gamma/2) = P_y/L$$



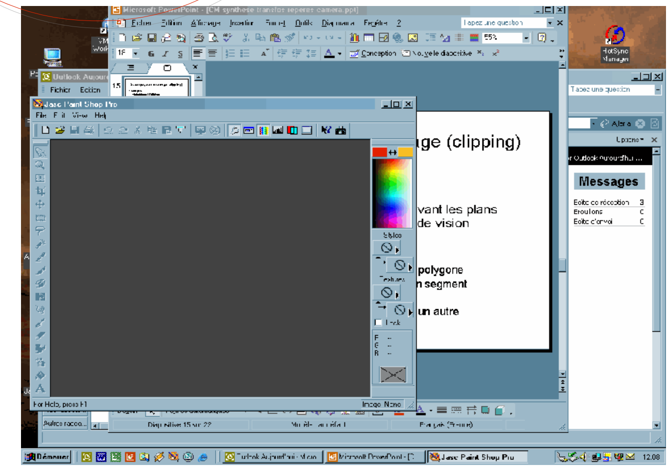


- Plan du module
  - introduction
  - représentations des objets
  - courbes et surfaces
  - interpolation vs approximation
  - transformations & projections
  - découpage



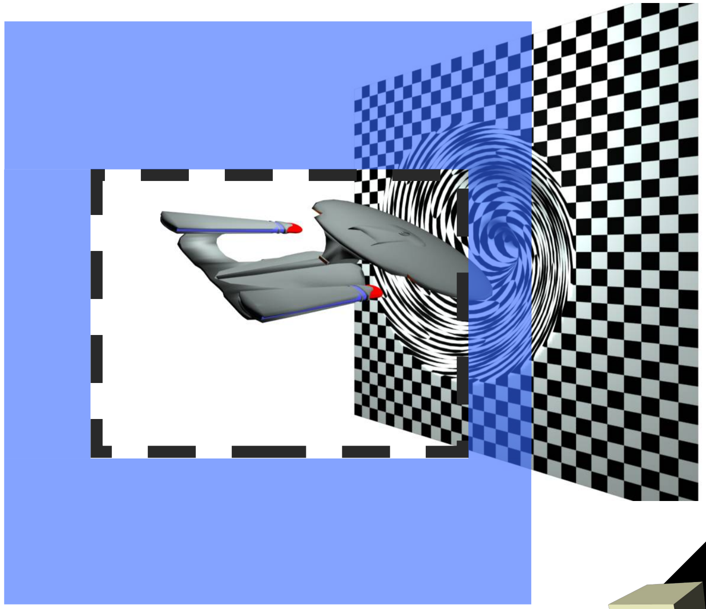
Que faut-il faire ?

- le **découpage** de la vue suivant les plans avant et arrière du volume de vision
- le **fenêtrage** à l'affichage



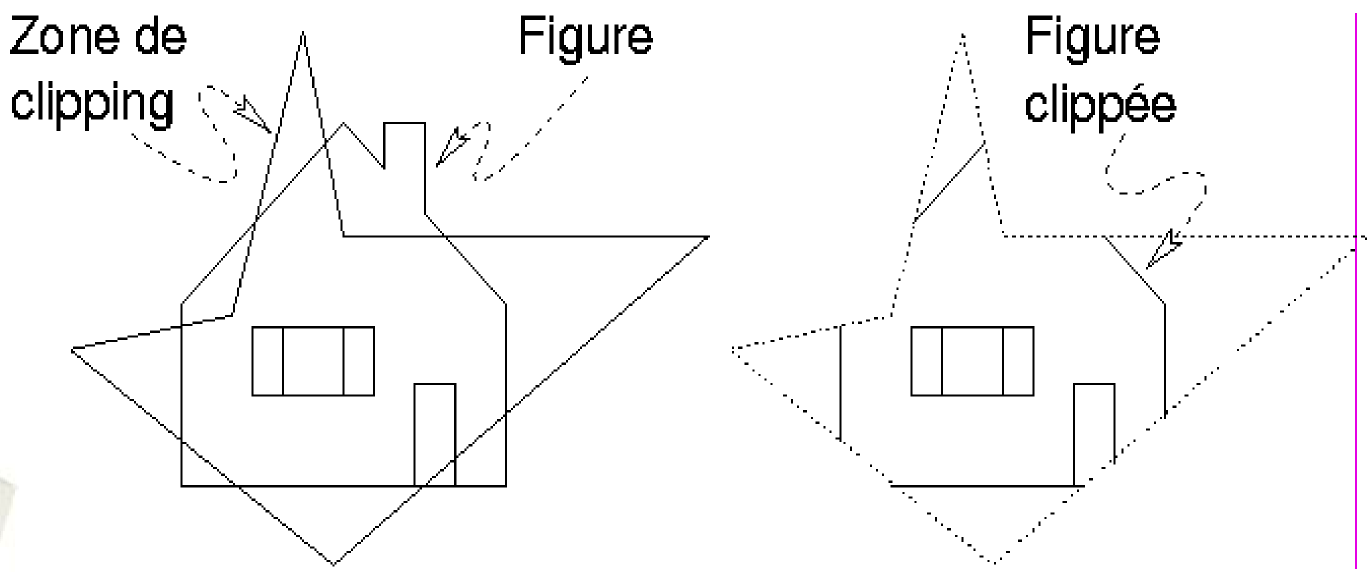
Quelques outils :

- intériorité d'un point pour un polygone
- appartenance d'un point à un segment
- intersection de segments
- inclusion d'un polygone dans un autre



-> pour afficher une fenêtre, pour cacher un polygone par un autre, pour ne pas dessiner ce qui n'est pas vu ...

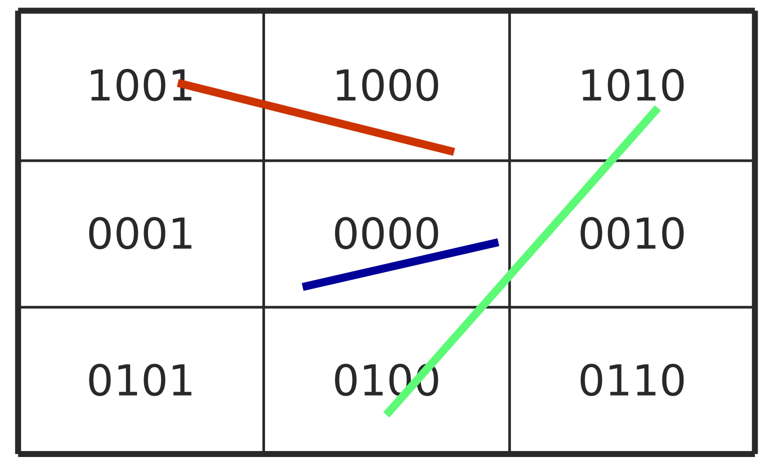
Algo simple : calcul des intersections de tous les segments avec le polygone de fenêtre -> long et complexe



Découpage de segments

Algo. de Cohen-Sutherland

- pour chaque segment on affecte aux extrémités les 4 bits de la zone
- On teste ensuite chaque paire d'extrémités *(A,B)* :



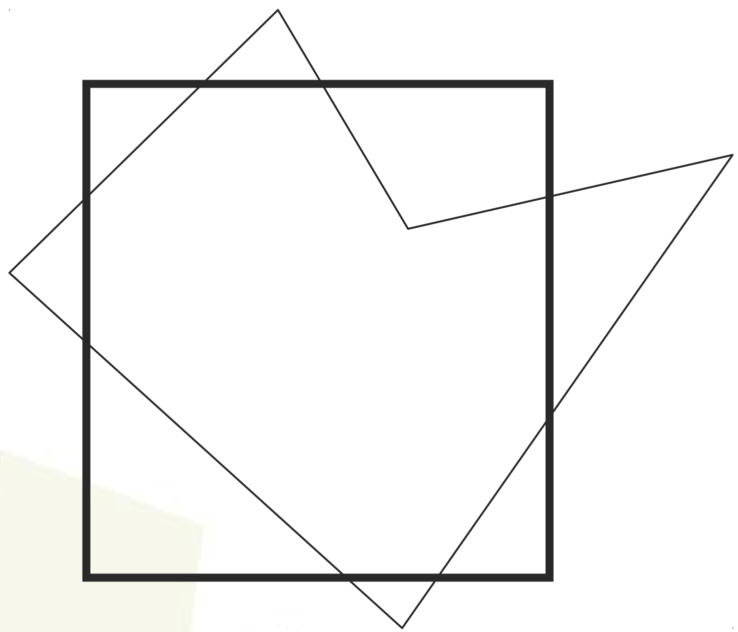
=1 si au dessus	=1 si en dessous	=1 si à droite	=1 si à gauche

si A=B=0000, segment ***visible***

sinon si *ET(A,B)!=0*, segment ***invisible***

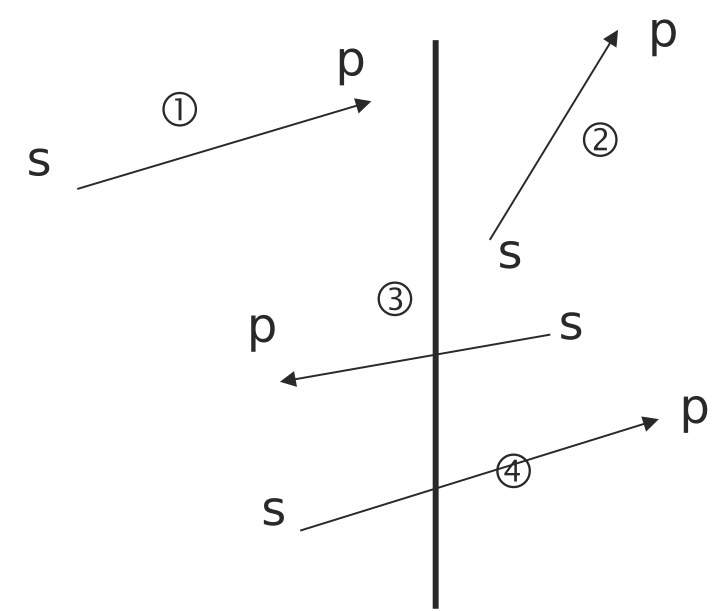
sinon ***découpage*** du segment et ***réitération***

Découpages de polygones (*Sutherland-Hodgman*)



Par chacun des bords de la fenêtre,

- suivant un ordre prédéfini : *gauche*, *haut*, *droit*, *bas* (par ex.)
- en utilisant des segments orientés, on parcourt les sommets successivement



1) conserver *p*
2) ne rien conserver
3) conserver *p* et *[sp)* inter clôture
4) (conserver *s*) conserver *[sp)* inter clôture

Besoin d’outils

- intersection de droites et appartenance d'un point à une droite
  -> en utilisant les équations paramétriques

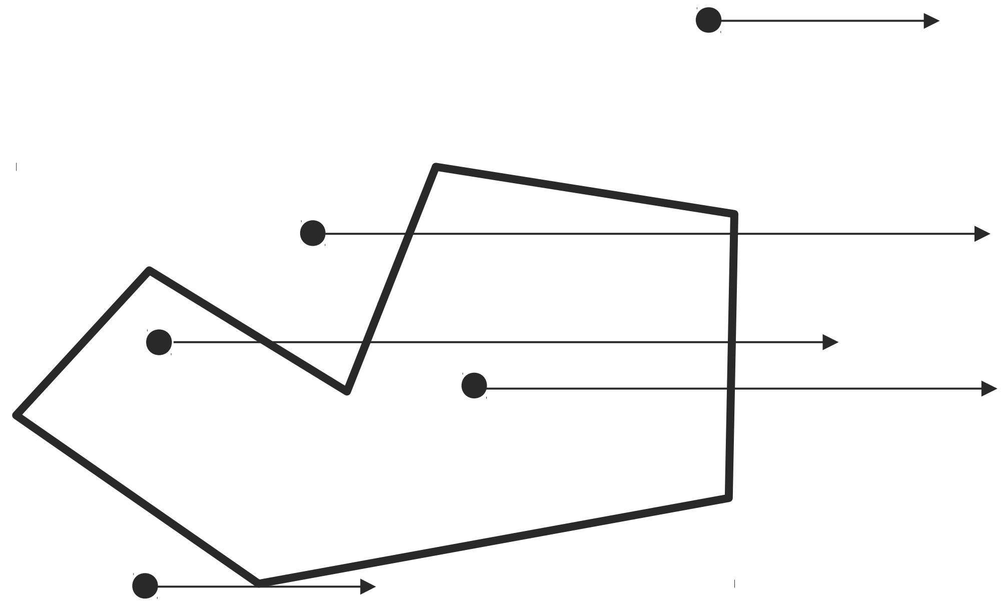
- intériorité d'un point pour un polygone
  - solution simple : intersection de ½ droites
  - 2ème solution : exprimer les angles polaires des sommets
  - 3ème solution : secteurs polaires

Point intérieur (1) : intersections de ½ droites

On tire des demi-droites à partir d'un point et // à un axe

-> point ***intérieur*** si nombre intersections ***impair***

PB : extremum local



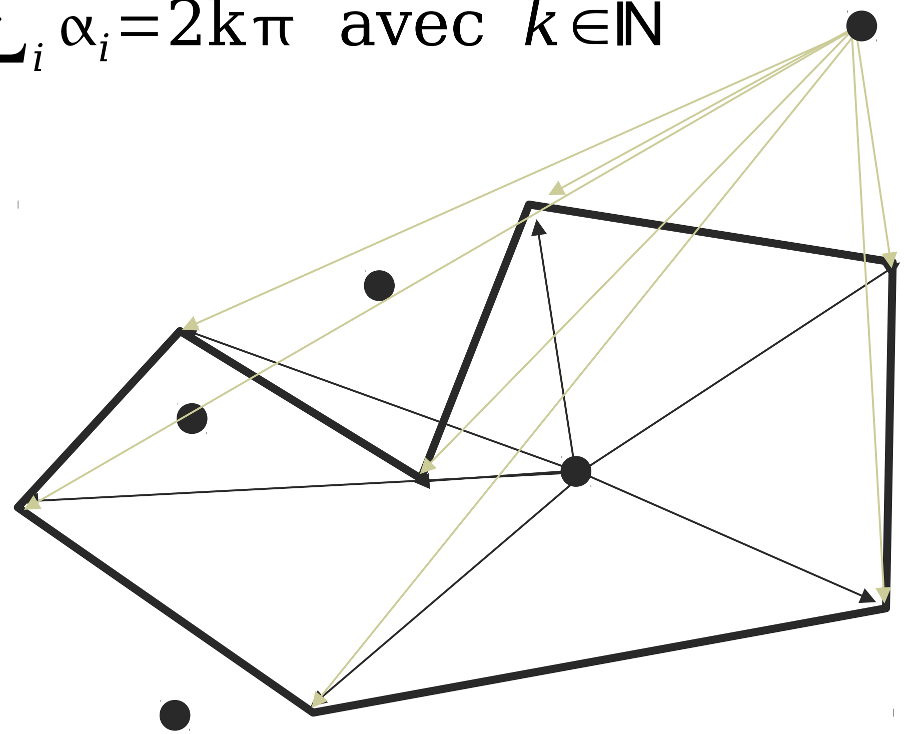
Point intérieur (2) : somme d'angles orientés

On calcule les angles α_i que fait le point avec 2 sommets successifs (angles orientés)

-> point **intérieur** si

$$\sum_i \alpha_i = 2k\pi \quad \text{avec} \quad k \in \mathbb{N}$$

PB : long !!

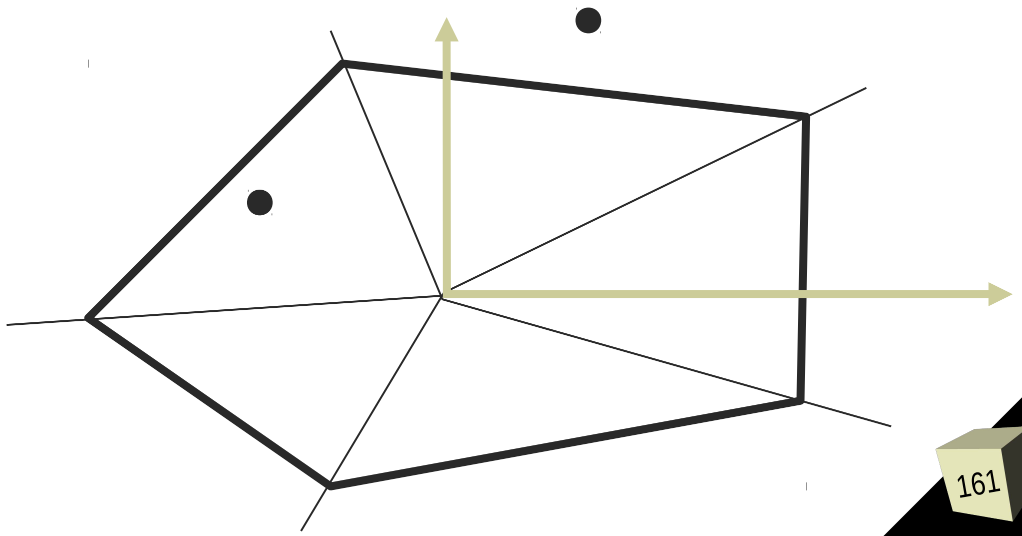


Point intérieur (3) : secteur polaires

(pour un polygône convexe)

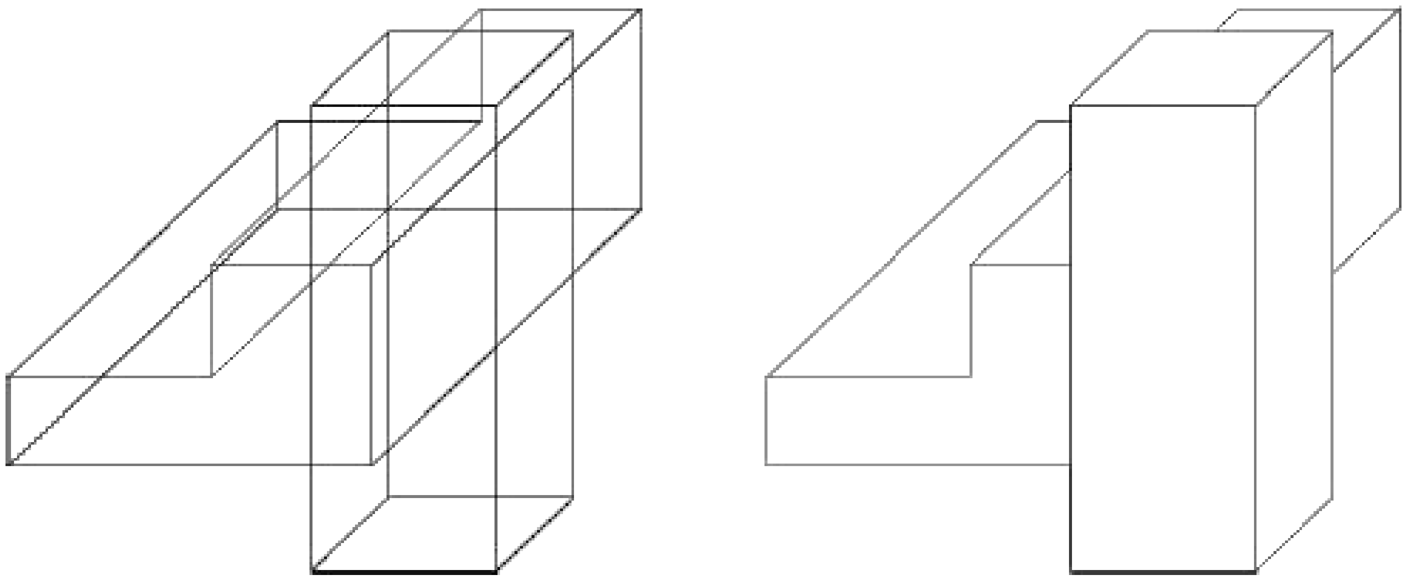
à partir d'un point intérieur :

- on construit un repère ;
- on délimite des secteurs polaires avec les sommets ;
- on trouve le secteur qui contient le point ;
- on cherche ensuite si le point est intérieur ou pas au segment définissant le secteur.



- ~~quelques rappels de maths ;~~
- ~~transformations de l'espace ;~~
- ~~projections ;~~
- ~~visualisation ;~~
- parties cachées.

Exemple d'élimination de parties cachées



INFORMATIONS UTILES POUR LA SUPPRESSION DES PARTIES CACHÉES 1/2

Points

Le calcul des parties cachées se fait généralement après la projection en perspective.
On a donc deux types d'information :

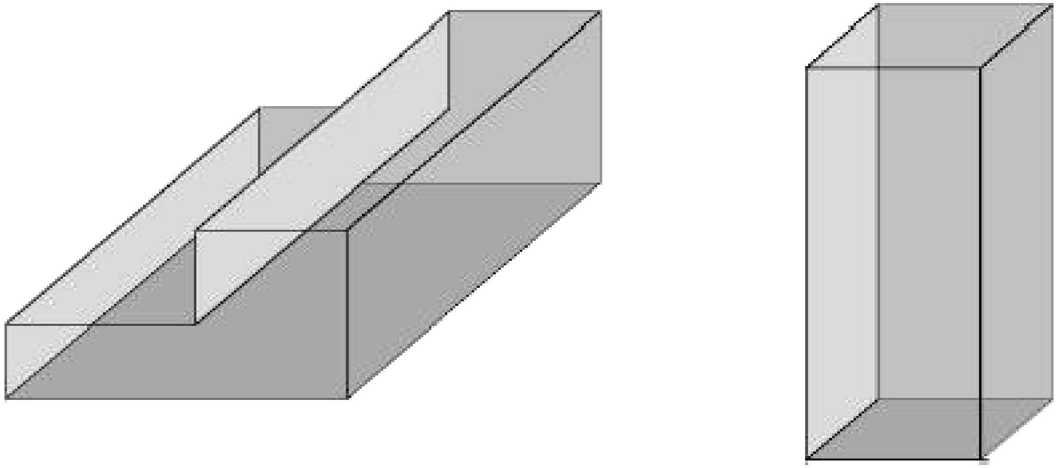
- **abscisse et ordonnée dans le plan de projection**, et
- **pseudo-profondeur** vue dans le cours sur les projections.

Les coordonnées des points dans le plan de projection **ne sont pas arrondies** pour la précision des calculs de parties cachées.

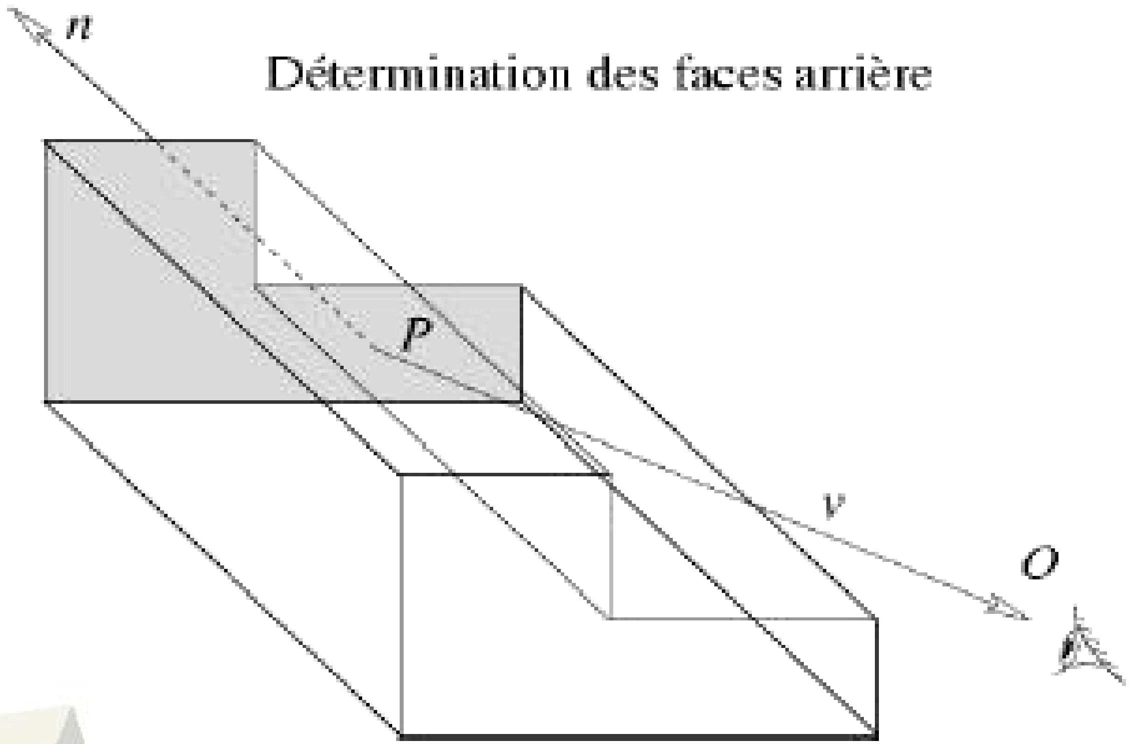
Faces

- **sommets de la face** avec 3 coordonnées,
- équation du **plan de la face**,
- coordonnées du **cube englobant la face** pour accélérer le clipping et les calculs de parties cachées.

Cas particulier : les faces arrières d'un objet



Elimination des faces arrières « *Backface Culling* »



$$\text{si } \vec{n} \cdot \vec{v} < 0 \quad \text{alors supprimer la face}$$

(mauvaise orientation de la normale par rapport à la direction de visée)

La suppression des faces arrières suffit à éliminer les parties cachées si :

- l'objet est seul dans la scène
  (il ne faut pas qu'un objet puisse en masquer un autre)
- **et** l'objet est convexe
  (dans un objet concave, des faces peuvent être masquées par d'autres)

- ***espace objet***
  -> calculs d'intersections entre : les plans de clipping et les objets, les objets entre eux, vérification de tous les sommets (comme le *Cohen-Sutherland* en 2D)

- ***espace image***
  -> sans *a priori* sur l'objet et sa modélisation, limité à la résolution de sortie, imprécis mais rapide

Algorithme de *Roberts* (63)

- élimination des lignes cachées d'un objet par lui-même (dépend du point de vue)

- pour chaque face restante, tester avec les autres faces des autres objets pour déterminer les lignes cachées

Algorithme du peintre

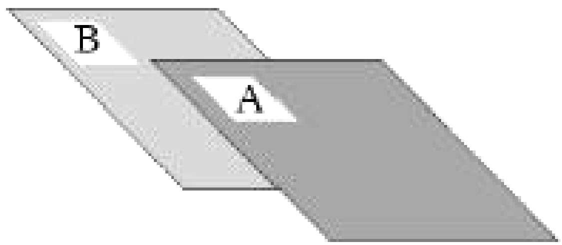
- tri des faces selon leurs éloignements

- *Newell*, *Newell* et *Sancha* (72)
  -> résolution des intersections ou de la cyclicité par *Sutherland-Hodgman*

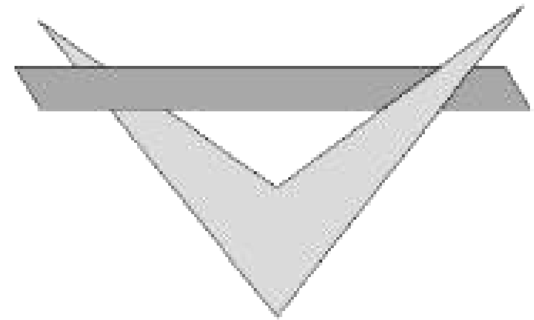
  -> comme la majorité des déplacements sont de faibles importances, on peut construire une liste de priorités, indépendante du point de vue

Quelques cas problématiques dans l'algorithme du peintre «insouciant»

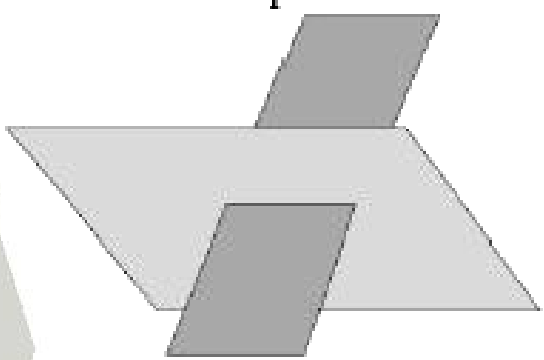
Nécessité de reclassement
A masque B alors que l'extension z maximale de A est supérieure à celle de B



Nécessité de fragmentation
Polygone convexe



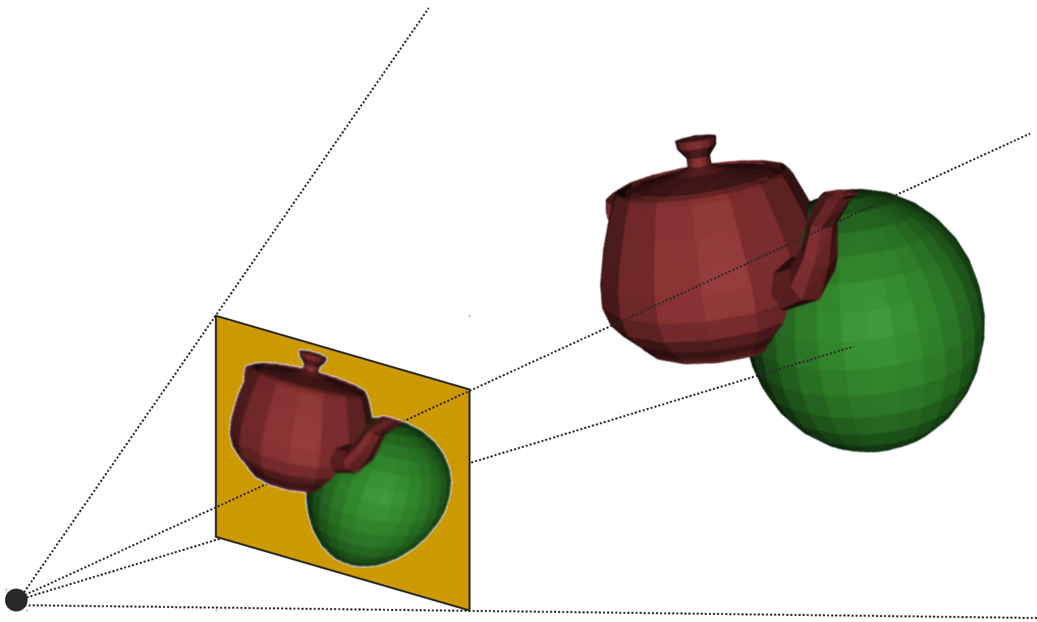
Nécessité de fragmentation
Interpénétration



Nécessité de fragmentation
Recouvrement cyclique

Z-buffer

- travaille facettes par facettes, dans l'espace image (*Catmull 74*)
- rapide, implémenté dans les cartes graphiques
- nécessite une mémoire de travail



Z-buffer (2)

principe (*Catmull 74*) :

- une mémoire d'image est initialisé avec la **couleur du fond** ;

- la mémoire en profondeur est initialisé avec la plus grande valeur (+*infini* ou *plan de clipping=0*)

- **pour chaque face** :
  on calcule pour chaque point *(x,y)*, la coordonnée z. On compare avec la valeur dans le Z-buffer et on met à jour, si nécessaire, le tampon et la mémoire image

PB : calcul de la coordonnée *z* ; imprécision de l'espace image (entiers) ; antialiasing difficile

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

5	**5**	**5**	**5**	**5**	**5**	**5**
5	**5**	**5**	**5**	**5**	**5**	
5	**5**	**5**	**5**	**5**		
5	**5**	**5**	**5**			
5	**5**	**5**				
5	**5**					
5						

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

3					
4	**3**				
5	**4**	**3**			
6	**5**	**4**	**3**		
7	**6**	**5**	**4**	**3**	
8	**7**	**6**	**5**	**4**	**3**

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
6	5	5	3	0	0	0	0
7	6	5	4	3	0	0	0
8	7	6	5	4	3	0	0
0	0	0	0	0	0	0	0

Z-buffer (3)

Mise en oeuvre dans OpenGL

Création des deux buffers :

```
glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH );
```

Activation :

```
glEnable(GL_DEPTH_TEST);
```

RAZ des deux buffers :

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```