INFO 805 - Introduction à l'informatique graphique

Jacques-Olivier Lachaud

Laboratoire de Mathématiques

CNRS / Université de Savoie

3 avril 2017

Organisation du cours

- 10 cours/td
  1. Géométrie dans l'espace
  2. Modélisation géométrique
  3. Matériaux, illumination, textures
  4. Caméra et perspective
  5. Algorithmes de rendu
  6. Pipe-line graphique et shaders
- 3 TPs
  1. soupe de triangles : visualisation, simplification/compression (C++, QGLViewer)
  2. rendu par lancer de rayons (surfaces implicites, matériaux, lumières) (C++)
  3. manipulation de shaders (webGL)

Un mot sur OpenGL et le pipeline graphique

- Plusieurs librairies 2D/3D pour exploiter les cartes graphiques : OpenGL, OpenGL ES, DirectX, WebGL
- OpenGL (et OpenGL ES) ont l'avantage d'être cross-platform et sont supportés en grande partie par les cartes graphiques (nVidia, ATI).
- Supporté par de nombreuses compagnies (Khronos group : ARM, Samsung, Huawei, Intel, Nokia, Sony, Google, AMD, Apple, nVidia, etc) / Microsoft sorti en 2003
- Implémentation sans carte graphique avec MESA.
- OpenGL écrit pour le C / C++, mais des bindings existent avec de nombreux langages (java, lua, C#)
- modifications majeures entre OpenGL2 et OpenGL3 : refonte pour utiliser les shaders.

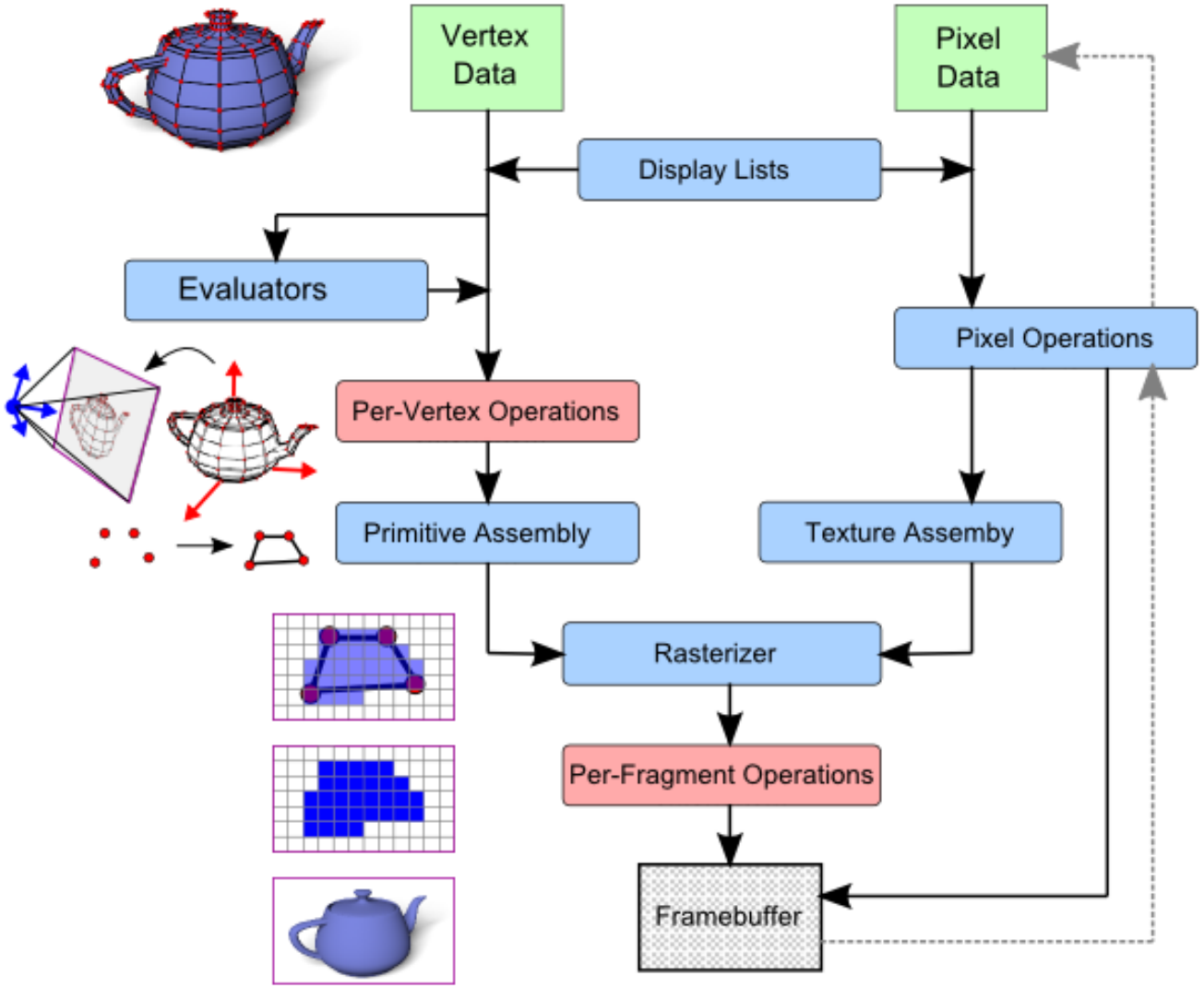
Un mot sur OpenGL et le pipeline graphique

- Plusieurs librairies 2D/3D pour exploiter les cartes graphiques : OpenGL, OpenGL ES, DirectX, WebGL
- OpenGL (et OpenGL ES) ont l'avantage d'être cross-platform et sont supportés en grande partie par les cartes graphiques (nVidia, ATI).
- Supporté par de nombreuses compagnies (Khronos group : ARM, Samsung, Huawei, Intel, Nokia, Sony, Google, AMD, Apple, nVidia, etc) / Microsoft sorti en 2003
- Implémentation sans carte graphique avec MESA.
- OpenGL écrit pour le C / C++, mais des bindings existent avec de nombreux langages (java, lua, C#)
- modifications majeures entre OpenGL2 et OpenGL3 : refonte pour utiliser les shaders.
- Bizarrement, OpenGL est plus une librairie 2D que 3D, notamment depuis la version 3 !

Un mot sur OpenGL et le pipeline graphique

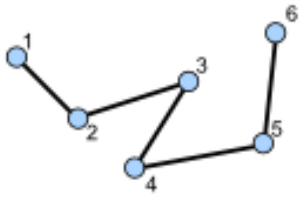
- Plusieurs librairies 2D/3D pour exploiter les cartes graphiques : OpenGL, OpenGL ES, DirectX, WebGL
- OpenGL (et OpenGL ES) ont l'avantage d'être cross-platform et sont supportés en grande partie par les cartes graphiques (nVidia, ATI).
- Supporté par de nombreuses compagnies (Khronos group : ARM, Samsung, Huawei, Intel, Nokia, Sony, Google, AMD, Apple, nVidia, etc) / Microsoft sorti en 2003
- Implémentation sans carte graphique avec MESA.
- OpenGL écrit pour le C / C++, mais des bindings existent avec de nombreux langages (java, lua, C#)
- modifications majeures entre OpenGL2 et OpenGL3 : refonte pour utiliser les shaders.
- Bizarrement, OpenGL est plus une librairie 2D que 3D, notamment depuis la version 3 !
- On pourrait se passer d'OpenGL / cartes graphiques pour le rendu réaliste

Le pipe-line OpenGL

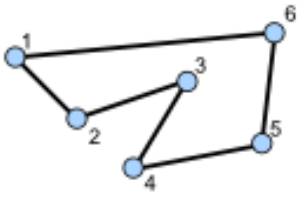


Les primitives graphiques en OpenGL

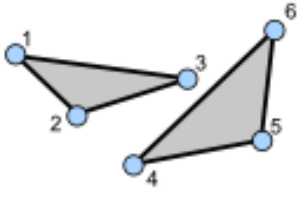
GL_POINTS



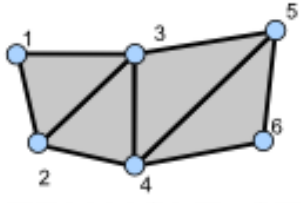
GL_LINE_STRIP



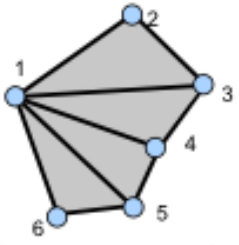
GL_LINE_LOOP



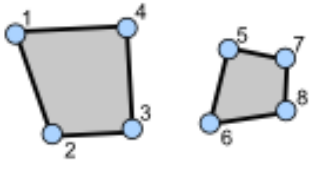
GL_TRIANGLES



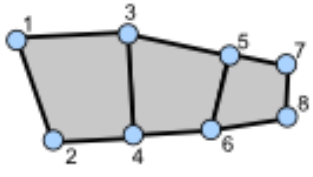
GL_TRIANGLE_STRIP



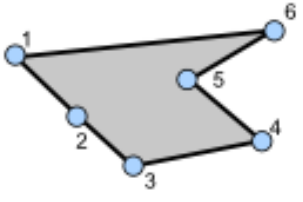
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

Matériaux, illumination, textures

Caméra et perspective

Algorithmes de rendu

Pipe-line graphique et shaders

La géométrie dans l'espace : pour quoi faire ?

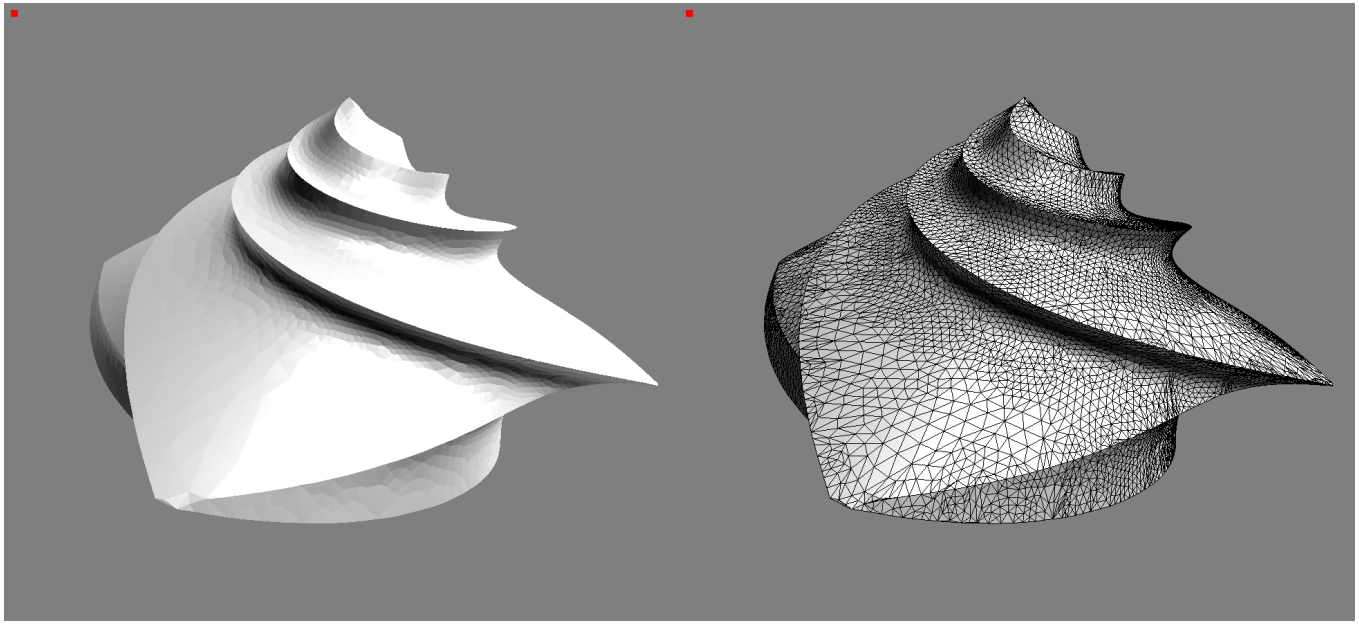
- définir ce qu'est un objet dans une scène
- placer les objets dans une scène, les tourner, les agrandir/diminuer, les déformer
- placer les éclairages dans une scène, les orienter, décider où ils éclairent
- placer l'oeil ou la caméra dans la scène, changer l'objectif de la caméra (zoom), placer les caméras si vision en relief
- calculer les couleurs d'un objet en fonction des éclairages et de la position de l'observateur
- déterminer les trajectoires des rayons lumineux pour des synthèses réalistes d'images
- calculer les intersections entre objets qui se déplacent selon des trajectoires

Définir la géométrie d'un objet = modélisation géométrique

En synthèse d'image, seule la surface de la forme nous intéresse en général : "on ne voit pas à l'intérieur des objets".
Deux grandes approches :

1. Représentations bord (B-rep) ou "explicites"
2. Représentations volumique ou "implicites"

Représentations bord (B-rep) ou “explicites”

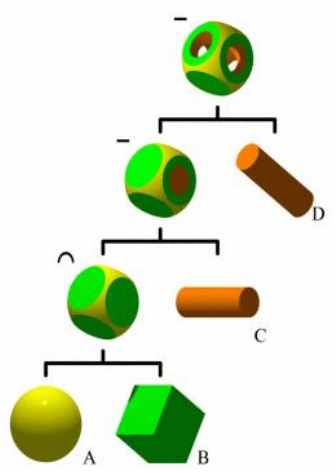


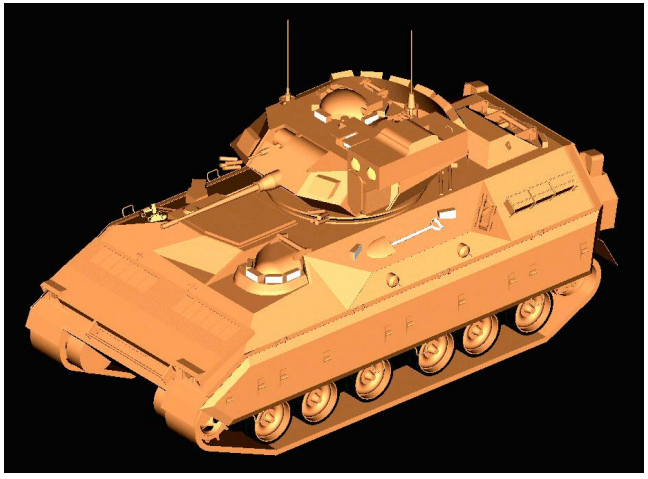
- ensemble de points
- soupe de triangles
- surfaces triangulées ou quadrangulées
- surfaces paramétriques : courbes et carreaux de Bézier, splines
- surfaces de subdivisions

\+ plus compactes, plus rapides à afficher (carte graphiques), souvent plus faciles à texturer

− pas toujours pratiques pour la conception assistée par ordinateur, “lissage” des assemblages pas toujours facile

Représentations volumique ou "implicites"



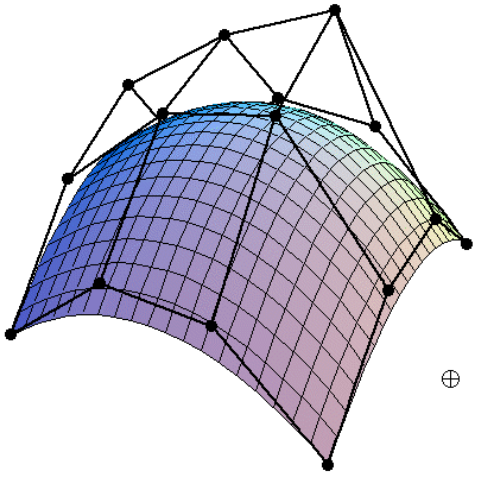


- fonctions implicites mathématiques (sphères, cubes, polynômes, etc)
- constructive solid geometry : assemblages de fonctions implicites par opérations sur ces fonctions
- images 3D (IRM, CT scans)
- feu, fumée, liquides, ...

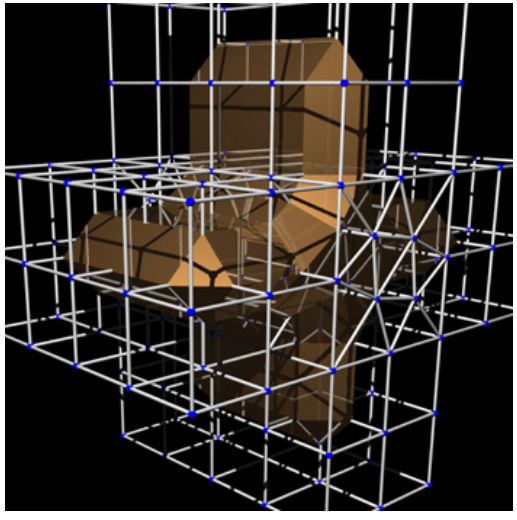
\+ formulations mathématiques facilitent certaines opérations (union, intersection, blending, etc)

– pas très compactes, pas très rapides à afficher par GPU

Transformations entre modèles



B-spline



marching cube

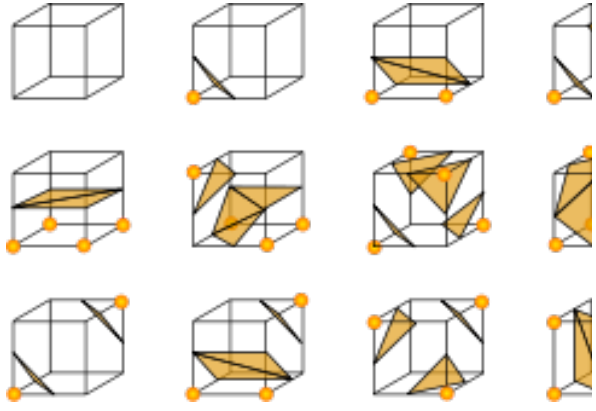
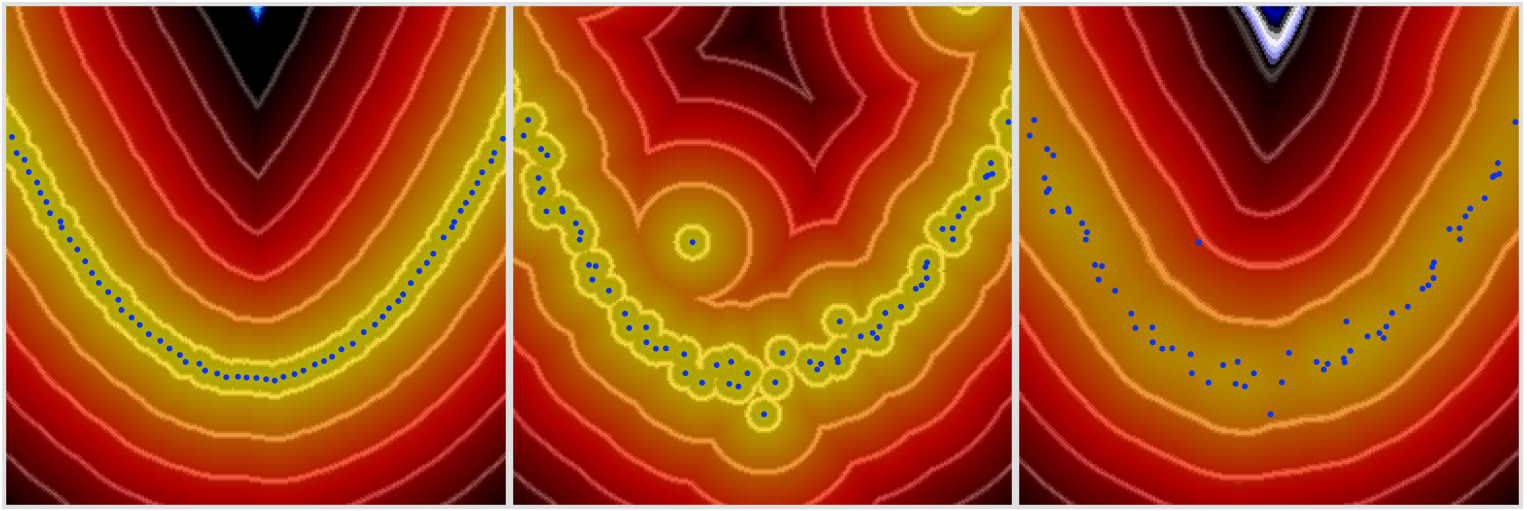


table du MC

Pour l'affichage, parfois la compression : "implicite" -> "explicite"

- B-rep type surfaces triangulées ou quadrangulées rendus directement
- Transformation B-rep générale vers ensembles de triangles par subdivision
- Représentations volumique ou "implicites" : il faut retrouver leur bord et les trianguler
- algorithme marching-cubes (et variantes : dual marching-cubes, etc)
- suivi de surface, gift wraping

Transformations entre modèles



Pour faciliter les test d'intersection / inclusion / localisation, certaines opérations : "explicite" -> "implicite"

- Ne fonctionne que si la B-rep définit un intérieur !
- Transformées en distance, avec de nombreuses variantes

Un exemple : le plan

1. implicite : 1 plan passant par a de vecteur normal $\vec{n}$.
   $\{p \in \mathbb{R}^3,$ tels que $\vec{ap} \cdot \vec{n} = 0\}$
2. explicite : 1 plan passant par trois points a, b, b
   $\{a + \alpha\vec{ab} + \beta\vec{ac},$ tels que $\alpha, \beta \in \mathbb{R}\}$.

Un exemple : le plan

1. implicite : 1 plan passant par a de vecteur normal $\vec{n}$.
   $\{p \in \mathbb{R}^3,$ tels que $\vec{ap} \cdot \vec{n} = 0\}$
2. explicite : 1 plan passant par trois points a, b, b
   $\{a + \alpha\vec{ab} + \beta\vec{ac},$ tels que $\alpha, \beta \in \mathbb{R}\}$.

Quid d'une boule ?

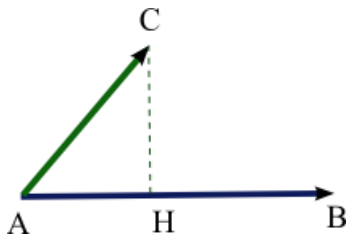
Points et vecteurs

- points et vecteurs sont des éléments de $\mathbb{R}^3$, et se représentent avec 3 valeurs réelles
- mais points et vecteurs n'ont pas les mêmes opérations
- $p, q \in \mathbb{R}^3$, $\vec{u}, \vec{v} \in \mathbb{R}^3$
- vecteurs : addition (comp.), soustraction (comp.), produit externe. Ex : $0,5\vec{u}$ est la moitié du vecteur $\vec{u}$.
- points : différence $q - p$ donne le vecteur $\vec{pq}$, point + vecteur donne un point. Ex : $p + 0,5\vec{pq}$ est le milieu du segment $[pq]$
- 2 opérations essentielles sur les vecteurs : produit scalaire, produit vectoriel.

Produit scalaire

Dans l'espace Euclidien, le *produit scalaire* entre $\vec{u}$ et $\vec{v}$ est le **réel** qui est la somme des produits de leurs composantes :

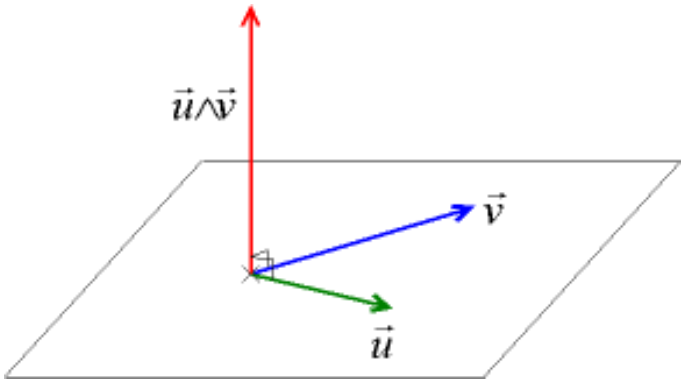
$$\vec{u} \cdot \vec{v} = u_x * v_x + u_y * v_y + u_z * v_z. \quad (1)$$



- norme euclidienne de $\vec{u}$: $\|\vec{u}\|^2 = \vec{u} \cdot \vec{u}$.
- Géométriquement, c'est le produit de la longueur de $\vec{u}$ avec la longueur (signée) du projeté de $\vec{v}$ sur $\vec{u}$.
- $\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos \alpha$, avec α angle entre vecteurs.
- Symétrique, bilinéaire
- **Orthogonalité** entre $\vec{u}$ et $\vec{v}$ ssi $\vec{u} \cdot \vec{v} = 0$.
- **Distance** entre A et B : $d(A, B) = \|\vec{AB}\| = \sqrt{\vec{AB} \cdot \vec{AB}}$.
- **Longueur du projeté** de $\vec{v}$ sur $\vec{u}$ = $\frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|}$.

Produit vectoriel

Contrairement au produit scalaire, le *produit vectoriel* entre 2 vecteurs $\vec{u}$ et $\vec{v}$ est un **vecteur** noté $\vec{u} \wedge \vec{v}$ ou $\vec{u} \times \vec{v}$.



C'est le seul vecteur $\vec{w}$ tel que :

1. $\vec{w}$ est orthogonal à $\vec{u}$ et à $\vec{v}$, i.e. $\vec{w} \cdot \vec{u} = \vec{w} \cdot \vec{v} = 0$,
2. la longueur de $\vec{w}$ est : $\|\vec{w}\| = \|\vec{u}\|\|\vec{v}\| \sin \alpha$,
3. $(\vec{u}, \vec{v}, \vec{w})$ est de sens direct.

$$\vec{u} \times \vec{v} = \begin{pmatrix} u_y * v_z - u_z * v_y \\ u_z * v_x - u_x * v_z \\ u_x * v_y - u_y * v_x \end{pmatrix} \quad (2)$$

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

Matériaux, illumination, textures

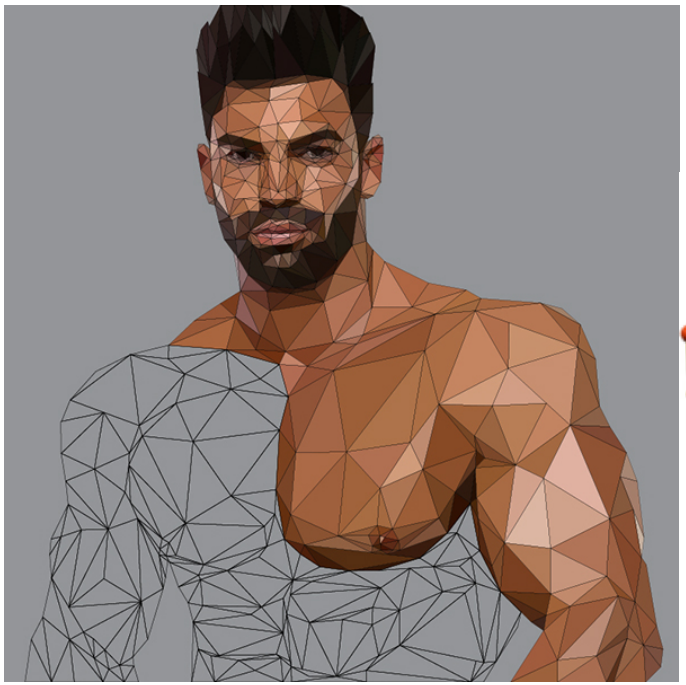
Caméra et perspective

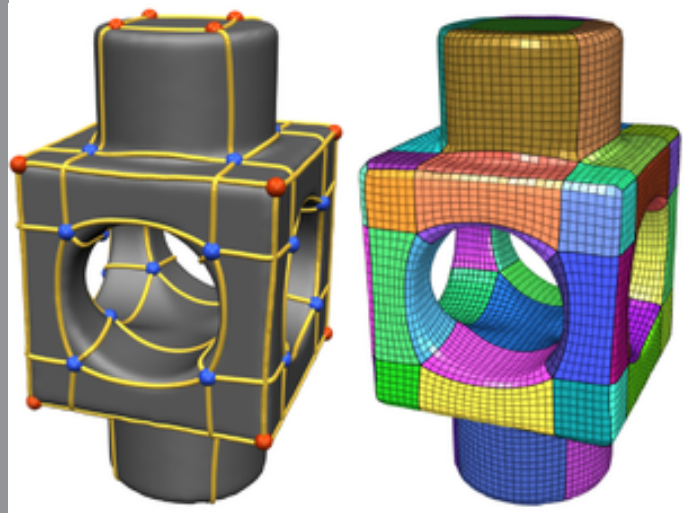
Algorithmes de rendu

Pipe-line graphique et shaders

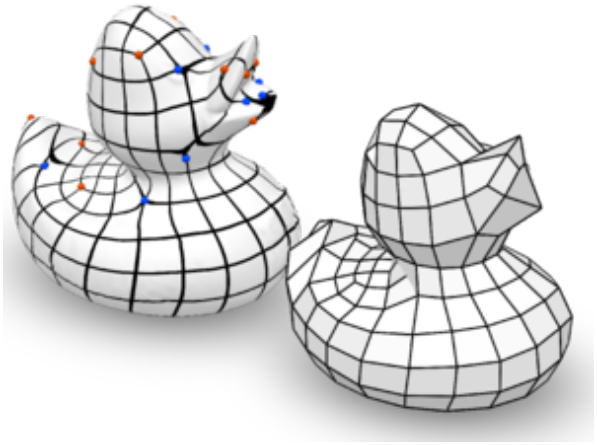
Représentations B-rep

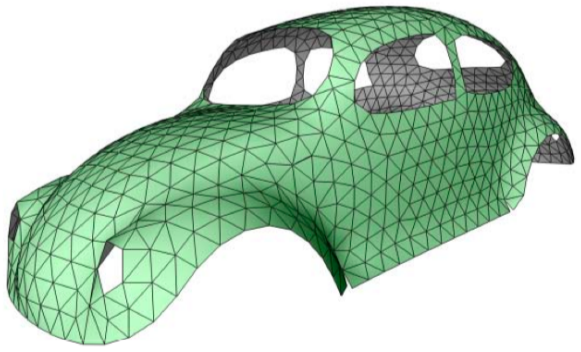
- représentation explicite de la surface de l'objet
- assemblage d'éléments simples, cousus entre eux
- *topologie* : comment les éléments sont cousus entre eux
- *géométrie* : comment chacun des éléments occupe l'espace
- *matériaux* : quelles propriétés/texture/couleurs ont chaque élément





Topologie des B-rep



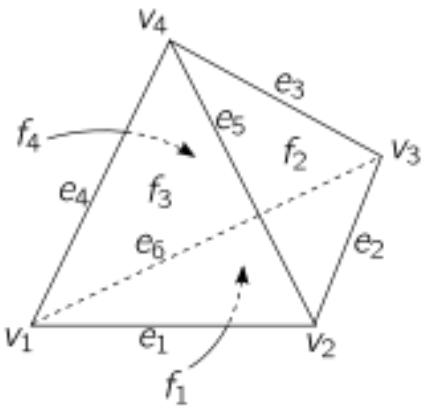


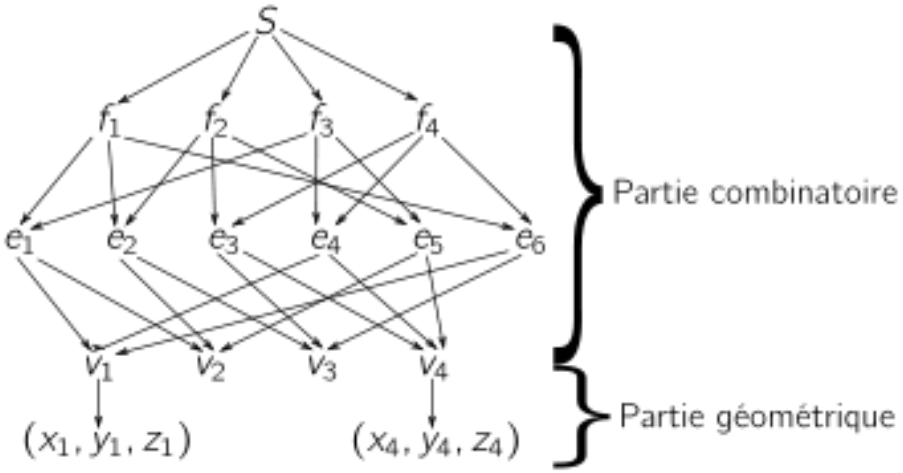
Mesh with boundary

- décomposition cellulaire de surfaces
- assemblage de faces (triangles, quadrilatères, ...) le long d'arêtes terminées par des sommets
- contraintes combinatoires pour avoir la topologie d'une surface
  - l'intersection de deux faces est vide, ou un sommet, ou une arête
  - une arête n'est incidente qu'à une ou deux faces
  - autour d'un sommet, les faces incidentes forment une ombrelle
- et encore plus de contraintes si la surface est le bord d'un solide :
  - une arête est incidente à exactement deux faces
  - le graphe d'arête est connexe si le solide est connexe sans cavités

Structures de données

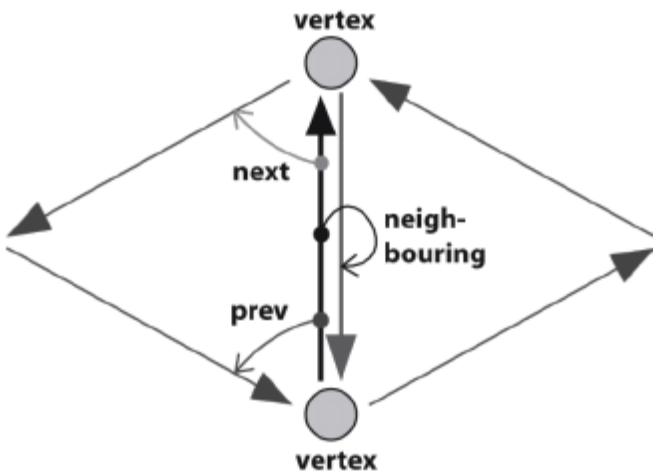
- soupe de triangles
- "graphe" : sommets, liste circulaire des voisins
- graphe d'incidence
- *winged-edge* structure (ou doubly-linked cell list)
- quad-edge data structure
- cartes combinatoires





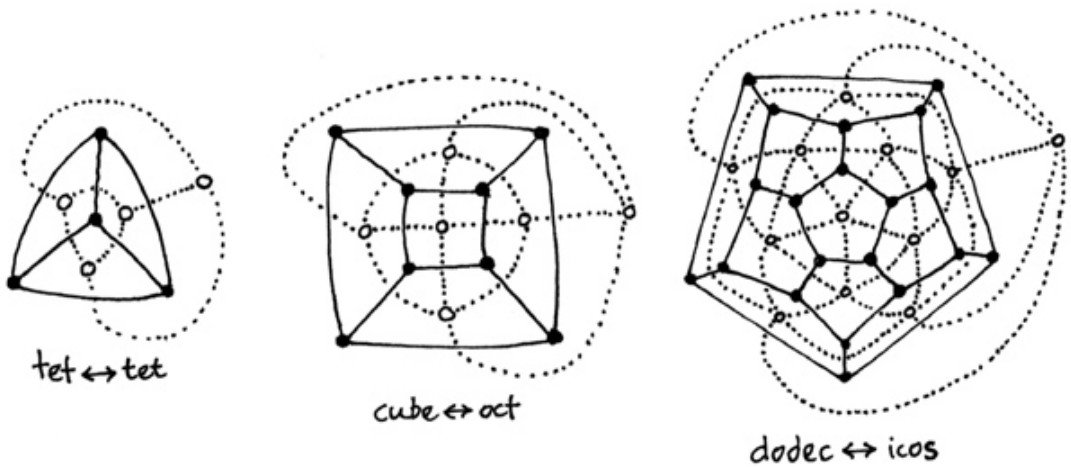
Structures de données

- soupe de triangles
- "graphe" : sommets, liste circulaire des voisins
- graphe d'incidence
- *winged-edge* structure (ou doubly-linked cell list)
- quad-edge data structure
- cartes combinatoires



Structures de données

- soupe de triangles
- "graphe" : sommets, liste circulaire des voisins
- graphe d'incidence
- *winged-edge* structure (ou doubly-linked cell list)
- quad-edge data structure
- cartes combinatoires



Géométrie affine par morceaux

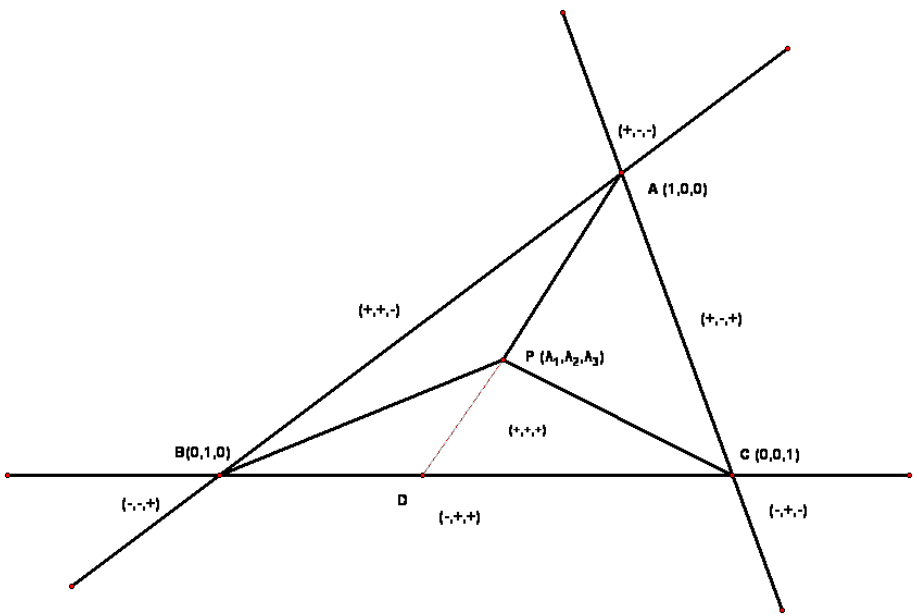
Très souvent, surfaces polygonales avec faces planes

- géométrie se réduit à placer les sommets (S_i) dans l'espace
- géométrie d'une arête $[S_i S_j]$ se déduit par *convexité*

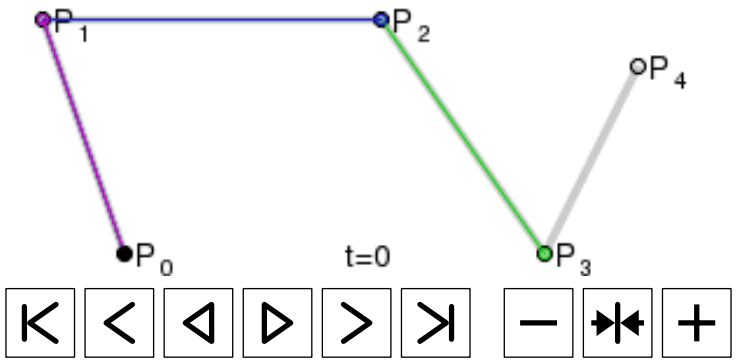
$$[S_i S_j] = \{(1-t)S_i + tS_i, \text{ t.q. } t \in [0,1]\}$$

- géométrie des faces se déduit par convexité, e.g. triangle S_i, S_j, S_k :

$$[S_i S_j S_k] = \{\alpha S_i + \beta S_j + \gamma S_k, \text{ t.q. } \alpha, \beta, \gamma \in [0,1] \text{ et } \alpha + \beta + \gamma = 1\}$$



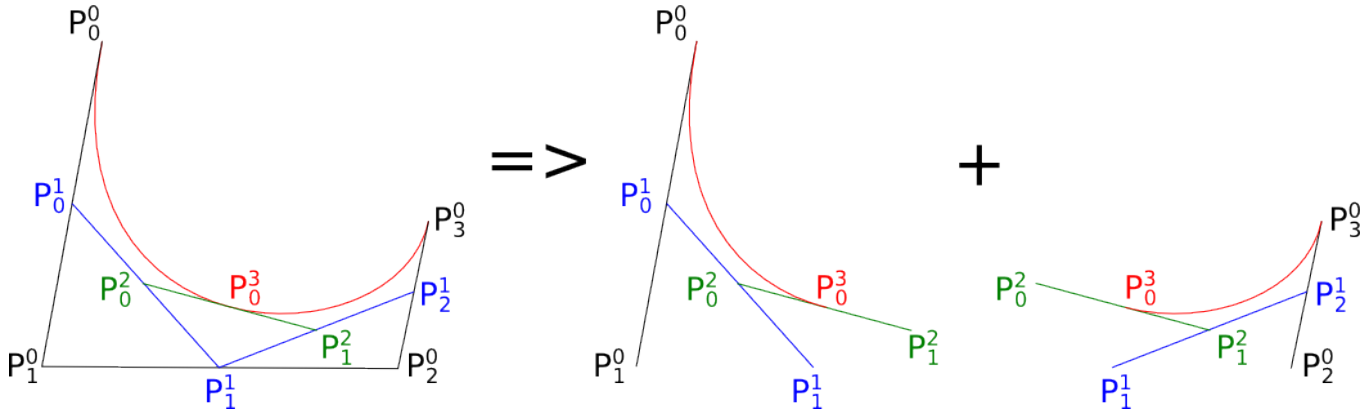
Géométrie lisse (par morceaux)



Courbes ou surfaces plus lisses obtenues par des polynômes de degré supérieur

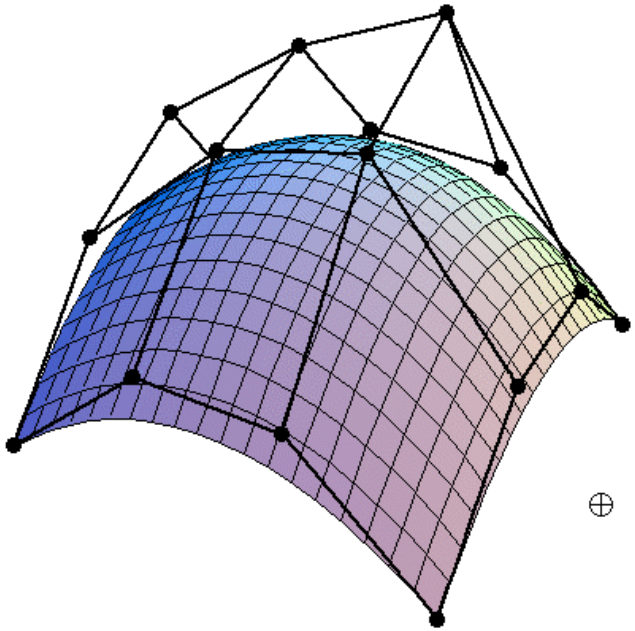
- Pour n+1 points de contrôle ($\mathbf{P}_0, \ldots, \mathbf{P}_n$), on définit une courbe de Bézier par l'ensemble des points $\sum_{i=0}^{n} B_i^n(t).\mathbf{P}_i$, avec $t \in [0,1]$ et où les B_i^n sont les polynômes de Bernstein.
- Si n=1, 2 points de contrôle : $B_0^1(t) = (1-t)$ et $B_1^1(t) = t$
- Si n=2, 3 points de contrôle : $B_0^2(t) = (1-t)^2$, $B_1^2(t) = 2(1-t)t$, $B_1^2(t) = t^2$
- les points $\mathbf{P}_0, \ldots, \mathbf{P}_n$ forment le « polygone de contrôle de Bézier ».

Algorithme de de Casteljau pour les courbes de Bézier



- on peut découper la courbe de Bézier n'importe où en 2 parties qui sont chacune des courbes de Bézier.
- on peut continuer récursivement
- le plus efficace est de découper à 1/2.

Surfaces lisses et carreaux de Bézier

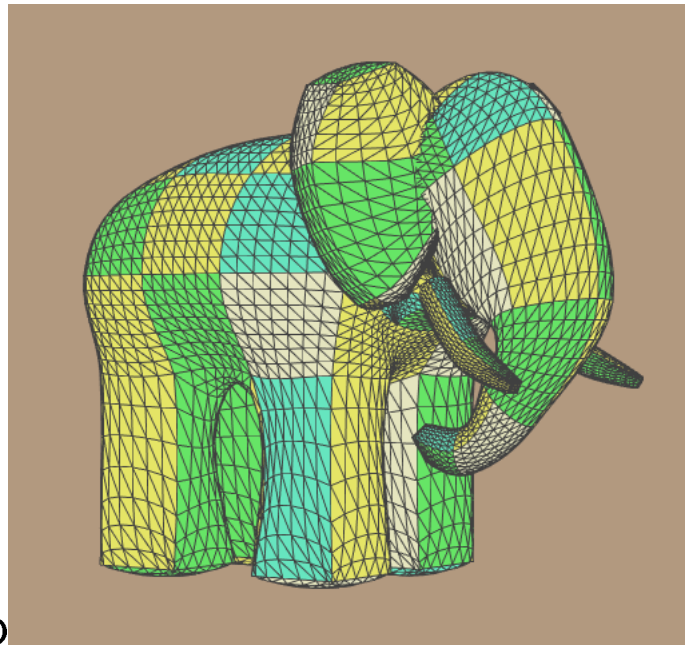


- le "produit" de deux courbes de Bézier paramétrées par u et v.
- Pour $(n+1) \times (m+1)$ points de contrôle $\mathbf{P}_{i,j}$
- $S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u) B_j^m(v) \mathbf{P}_{i,j}$
- $u, v \in [0,1]$ et les B_i^n, B_j^m sont les polynômes de Bernstein.

$$B_i^n(u) := \binom{n}{i} (1-u)^{n-i} u^i$$

Continuités (et plus) entre morceaux de surface

Catmull's Gumbo



- Contraintes sur les points de contrôles de part et d'autre
- continuité simple (C^0 ou G^0) avec même points de contrôles aux bords
- En général, continuité de tangente (dite G^1) suffisante, en rendant les points de contrôle symétrique de part et d'autre

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

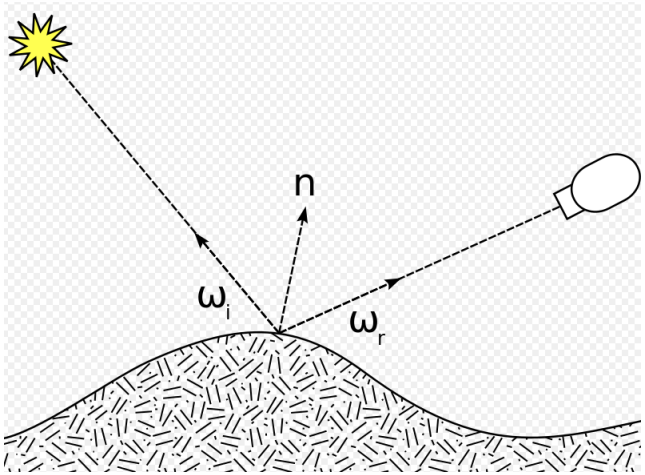
Matériaux, illumination, textures

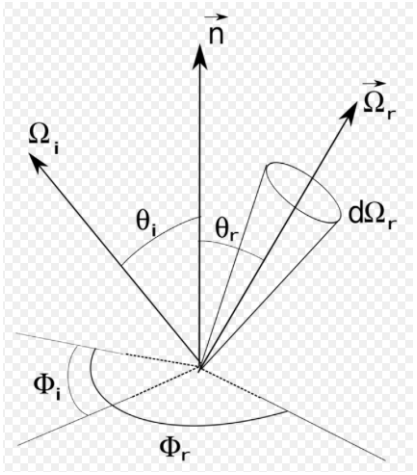
Caméra et perspective

Algorithmes de rendu

Pipe-line graphique et shaders

Modèle physique de couleurs : BRDF



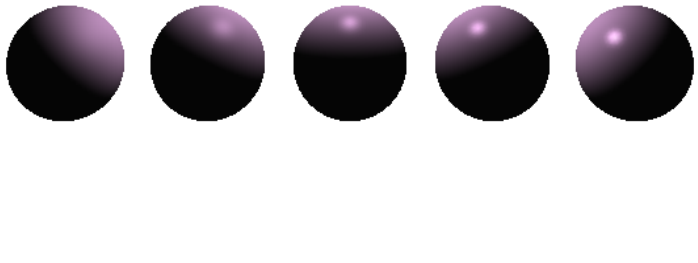


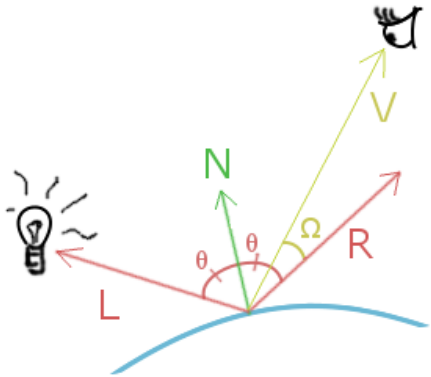
- lien luminance incidente $L^i_\nu(\Omega_i)$ et luminance réfléchie $L^r_\nu(\Omega_r)$ pour la fréquence ν
- BRDF = Bidirectional Reflectance Distribution Function
- fonction de distribution $f_\nu(\theta_i, \theta_r, \phi_i, \phi_r)$
- Luminance sortante :

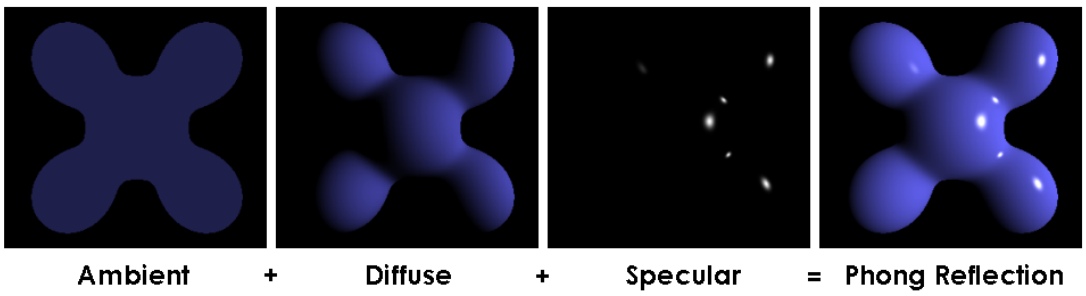
$$L_\nu(\Omega_r) = \int_{2\pi} f_\nu(\Omega_i, \Omega_r)\ L_\nu(\Omega_i)\ \mathrm{d}\Omega_i$$

$$= \int_0^{\pi} \int_0^{\frac{\pi}{2}} f_\nu(\theta_i, \phi_i, \theta_r, \phi_r)\ L_\nu(\theta_i, \phi_i)\ \sin\theta_\mathrm{i}\ \mathrm{d}\theta_\mathrm{i}\ \mathrm{d}\phi_\mathrm{i}$$

Modèle de Phong





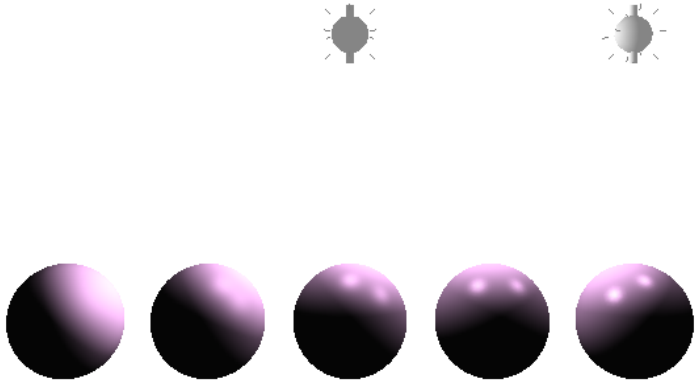


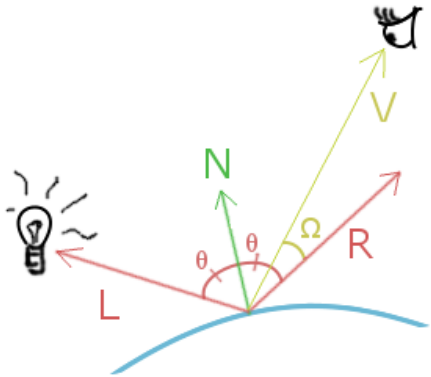
ambient 1 couleur sans lumière

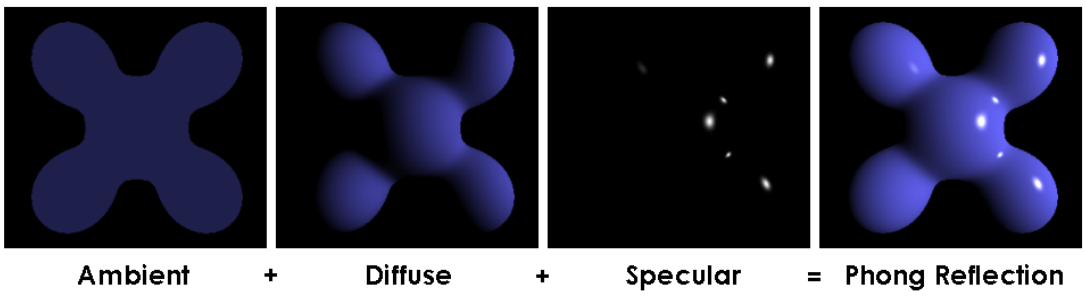
diffuse 1 couleur qui dépend des lumières (donc de θ)

spéculaire 1 couleur qui dépend des lumières et de l'œil (de θ et Ω)

Modèle de Phong





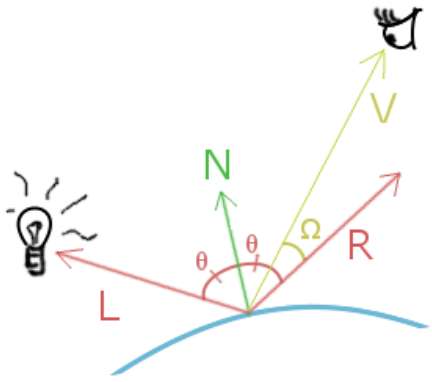


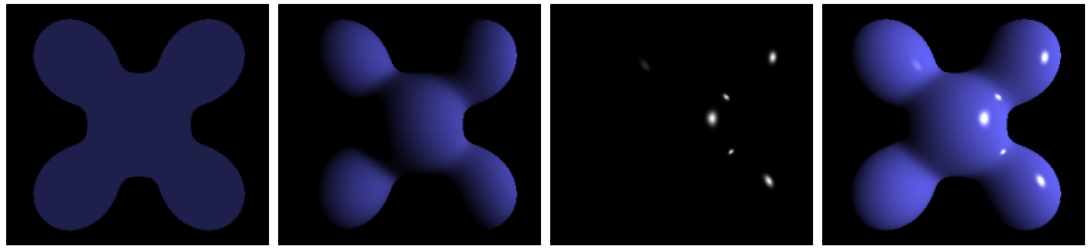
ambient 1 couleur sans lumière

diffuse 1 couleur qui dépend des lumières (donc de θ)

spéculaire 1 couleur qui dépend des lumières et de l'œil (de θ et Ω)

Modèle de Phong





ambient 1 couleur sans lumière

$\mathbf{C}_a$ couleur émissive

diffuse 1 couleur qui dépend des lumières (donc de θ_i)

$\sum_j \vec{N} \cdot \vec{L_j} \mathbf{C}_d * \mathbf{C}_{light_j}$ couleur diffuse

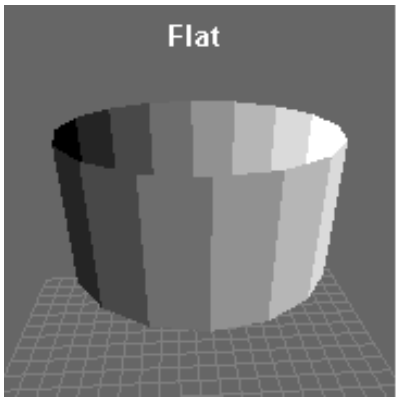
spéculaire 1 couleur qui dépend des lumières et de l'œil (de $\theta_i, \phi_i, \theta_r, \phi_r$)

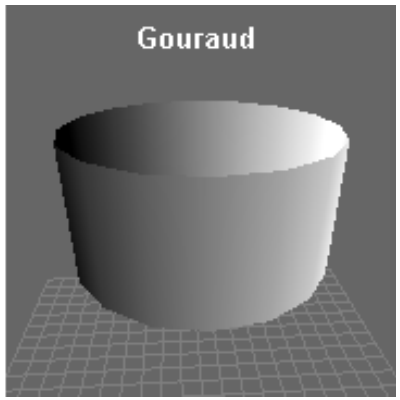
$\sum_j (\vec{V} \cdot \vec{R_j})^t \mathbf{C}_s * \mathbf{C}_{light_j}$ couleur spéculaire

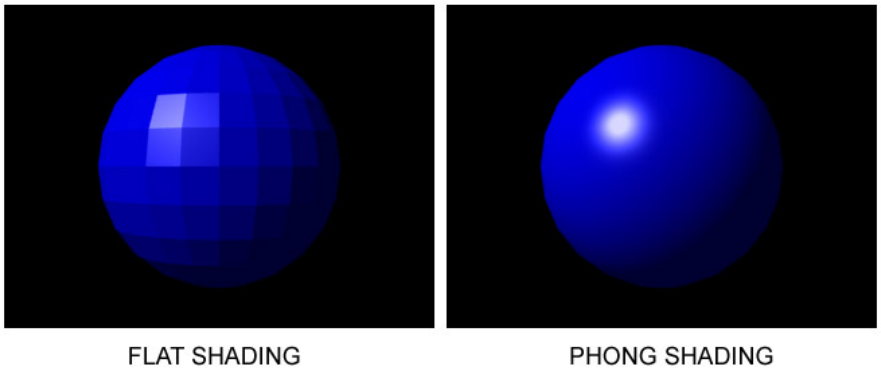
t est la brillance (1 : faible, 100 : très forte)

Matériaux

Ombrage / shading





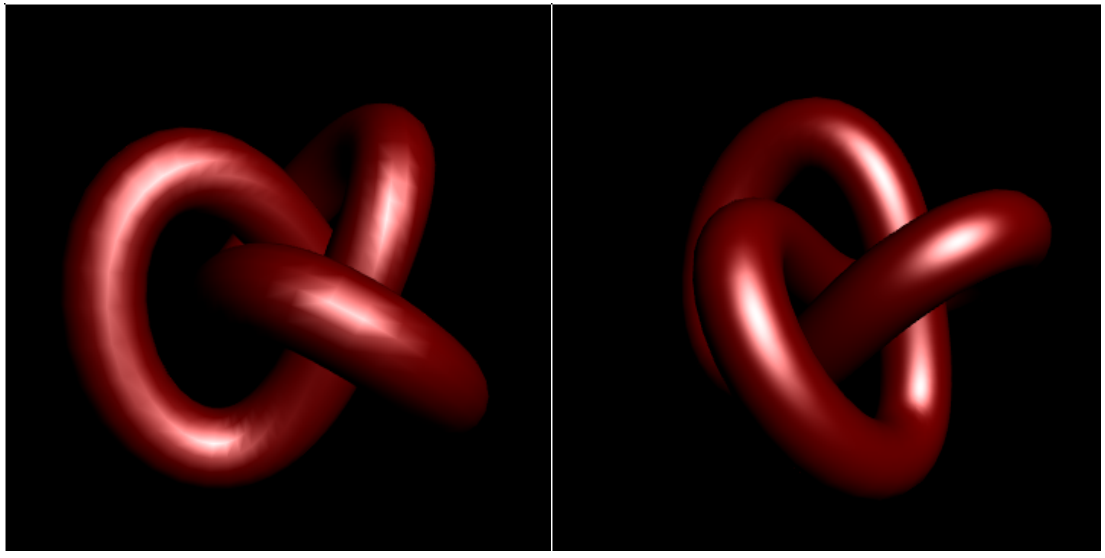


flat 1 normale par face, 1 couleur par face

Gouraud 1 normale par sommet, couleurs calculées au sommets et interpolées en chaque pixel

Phong 1 normale par sommet, normales interpolées en chaque pixel, couleurs calculées en chaque pixel

Ombrage / shading avec les shaders



- prog. GPU permet de faire des calculs **par sommet** et **par pixel**
- code WebGL ci-dessous montre comment faire du Phong shading temps réel

`http://www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/code/WebGLShaderLightMat/ShaderLightMat.html`

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

Matériaux, illumination, textures

Caméra et perspective

Algorithmes de rendu

Pipe-line graphique et shaders

Caméra, perspective... matrices !

- modélisation de la caméra (position, orientation), de la transformation perspective, du placement des objets dans une scène $\Rightarrow$ matrices
- coordonnées homogènes + matrices 4×4 donnent toutes les transformations usuelles
- Calcul matriciel : seulement des $\times$ et des $+$!

$$\begin{pmatrix} 1 & & & T_x \\ & 1 & & T_y \\ & & 1 & T_z \\ & & & 1 \end{pmatrix}$$

Translation

$$\begin{pmatrix} a & b & & \\ b & c & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$$

Rotation selon Z

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & f' & \end{pmatrix}$$

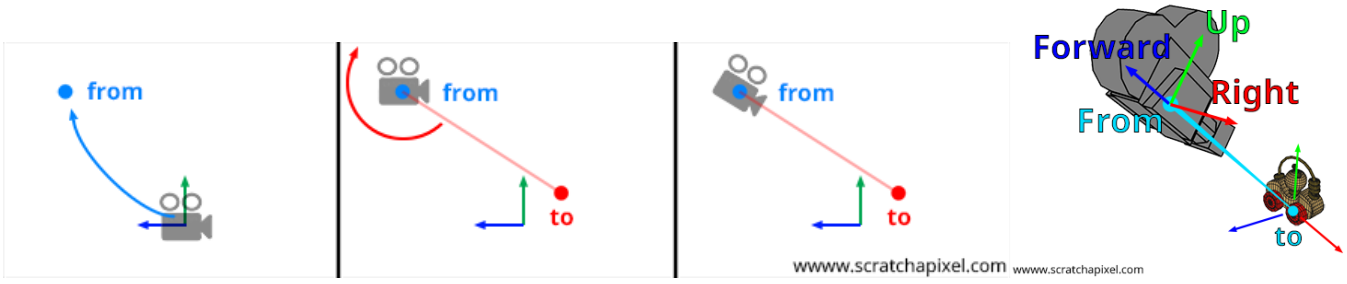
Projection sur (X,Y)

...

Placement des objets

- on associe une matrice de transformation M à chaque objet, souvent appelée *model matrix*
  `http://voxelent.com/html/beginners-guide/chapter_4/ch4_ModelView.html`
- typiquement, multiplication d'une translation, d'une rotation et d'un scaling.
- attention à la convention multiplication à gauche ou à droite de la position

Placement de la caméra

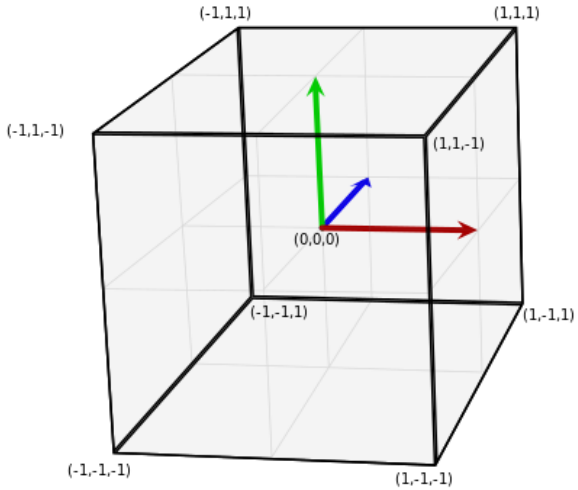


- matrice V : transforme coordonnées "camera" vers "monde"

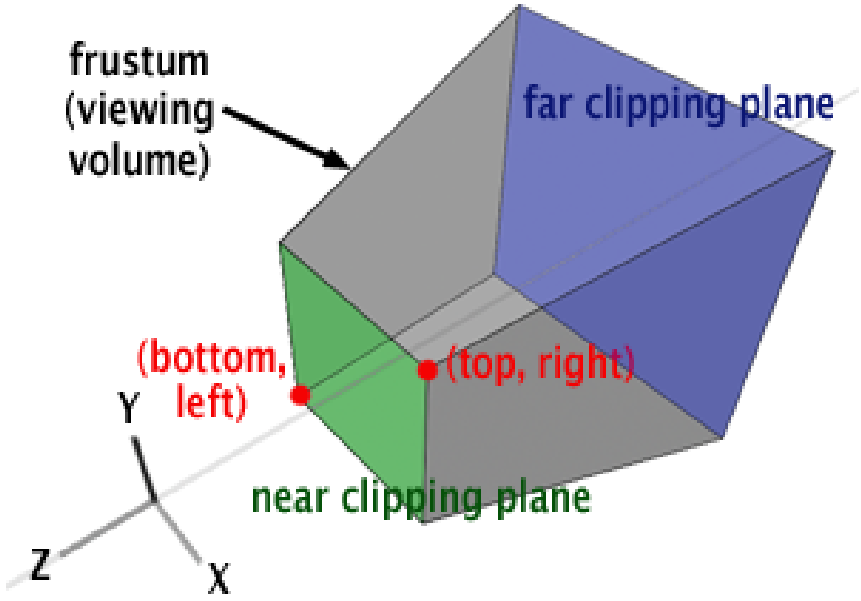
$$\begin{bmatrix} R_x & U_x & F_x & T_x \\ R_y & U_y & F_y & T_y \\ R_z & U_z & F_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- matrice V^{-1} : transforme coordonnées "monde" vers "camera", souvent appelée *view matrix*

Transformation en coordonnées écran : la perspective



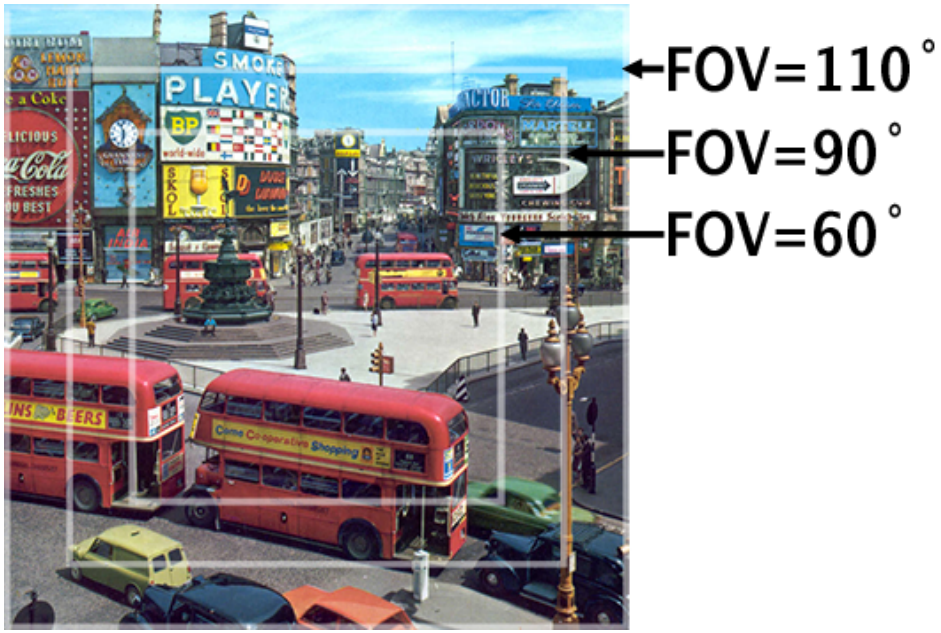
Clipspace

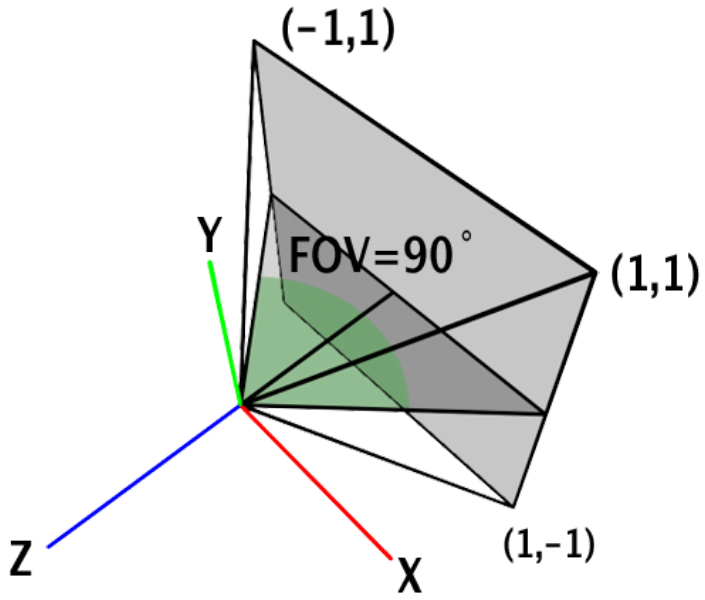


$$P := \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

n	near clipping plane
f	far clipping plane
t	top
b	bottom
l	left
r	right

Champ de vue de la caméra





- field of view *fov* : détermine avec l'aspect-ratio de l'écran les paramètres t, b, l, r.

$$S = \frac{1}{\tan(\frac{fov}{2} * \frac{\pi}{180})}$$

Résumé

- Suite de transformations pour amener un point X dans l'espace "pixels"
  1. model space $\longrightarrow$ model matrix M $\longrightarrow$ world space
  2. world space $\longrightarrow$ view matrix V^{-1} $\longrightarrow$ view space
  3. view space $\longrightarrow$ projection matrix P $\longrightarrow$ clip space
- WebGL et OpenGL
  1. Java script / C / C++ : on calcule M par objet, puis globalement P et V^{-1}
  2. vertex shader (GPU) calcule $PV^{-1}MX'$ pour caméra + projection
  3. fragment shader (GPU) s'occupe des interpolations normales/positions/couleurs

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

Matériaux, illumination, textures

Caméra et perspective

Algorithmes de rendu

Pipe-line graphique et shaders

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

Matériaux, illumination, textures

Caméra et perspective

Algorithmes de rendu

Pipe-line graphique et shaders