

INFO 805 - Introduction à l'informatique graphique

Jacques-Olivier Lachaud

Laboratoire de Mathématiques
CNRS / Université Savoie Mont Blanc

5 février 2024

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++, Qt5/Qt6, QGLViewer)
 2. rendu par lancer de rayons (surfaces implicites, matériaux, lumières) (C++, Qt5/Qt6, QGLViewer)
 3. sites web 3D avec javascript et WebGL, un peu de shaders

Un mot sur OpenGL et le pipeline graphique

- Plusieurs bibliothèques 2D/3D pour exploiter les cartes graphiques : OpenGL, OpenGL ES, DirectX, WebGL, Vulkan
- 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 bibliothèques 2D/3D pour exploiter les cartes graphiques : OpenGL, OpenGL ES, DirectX, WebGL, Vulkan
- 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 bibliothèque 2D que 3D, notamment depuis la version 3 ! Dernière version : OpenGL 4.6

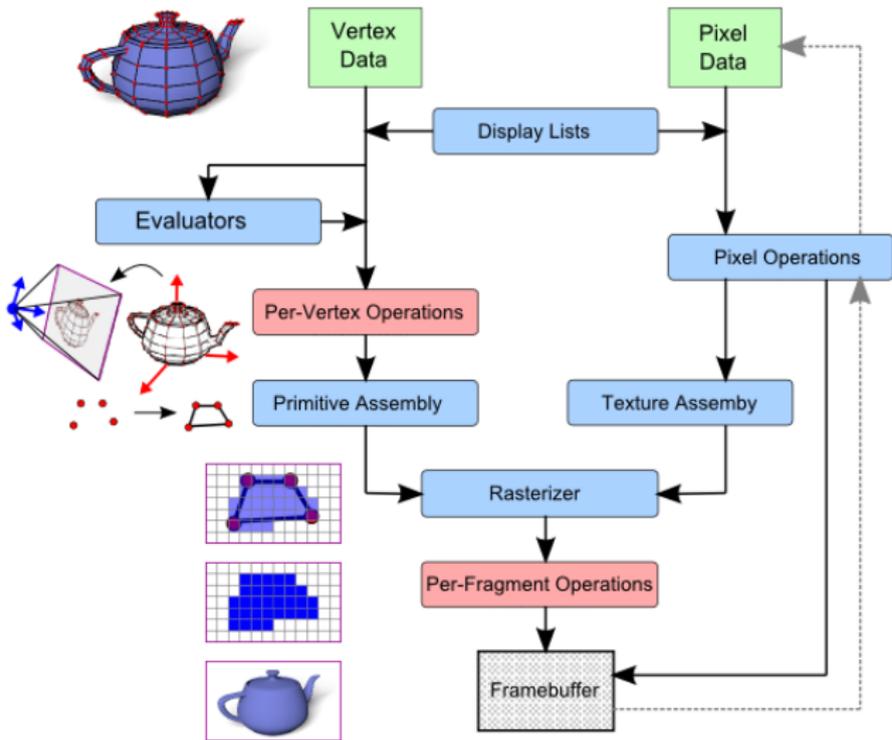
Un mot sur OpenGL et le pipeline graphique

- Plusieurs bibliothèques 2D/3D pour exploiter les cartes graphiques : OpenGL, OpenGL ES, DirectX, WebGL, Vulkan
- 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 bibliothèque 2D que 3D, notamment depuis la version 3 ! Dernière version : OpenGL 4.6
- On pourrait se passer d'OpenGL / cartes graphiques pour le rendu réaliste

WebGL et OpenGL

- WebGL est une spécification d'interface de programmation de 3D dynamique pour HTML5
- elle permet d'utiliser le standard OpenGL ES au sein d'une page web
- elle utilise javascript
- on retrouve donc l'essentiel d'OpenGL, notamment la programmation des vertex et fragment shaders en GLSL (OpenGL shading language)
- ses limites correspondent aux limites d'Open GL ES (embedded system)
 - pas de tampon de sélection
 - pas de fonctions fixes 3D à la OpenGL1/2
 - pas de textures 3D
 - pas de geometry shader

Le pipe-line OpenGL



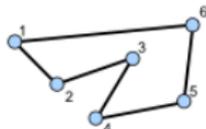
Les primitives graphiques en OpenGL



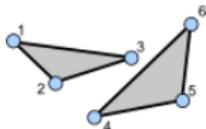
GL_POINTS



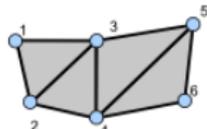
GL_LINE_STRIP



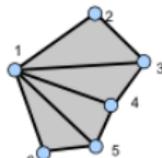
GL_LINE_LOOP



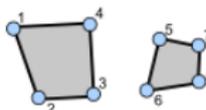
GL_TRIANGLES



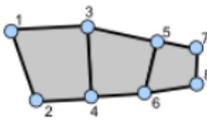
GL_TRIANGLE_STRIP



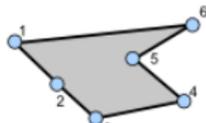
GL_TRIANGLE_FAN



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

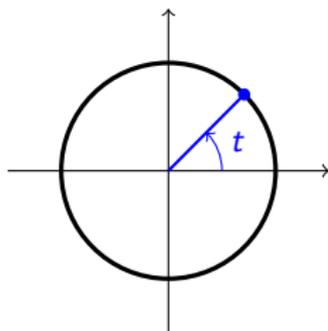
Définir la géométrie d'un objet = modélisation géométrique

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"

Exemple de définition implicite / explicite : le cercle en 2D

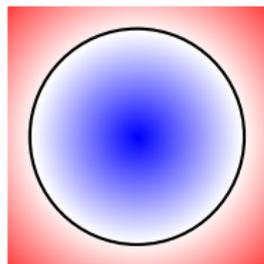


f de \mathbb{R} vers \mathbb{R}^2

$$f(t) = (r \cos t, r \sin t)$$

$$C = \{f(t), t \in [0, 2\pi[\}$$

explicite définition explicite d'un cercle de rayon r et de centre $(0, 0)$, en parcourant les points du cercle en fonction de l'angle avec l'axe des x .

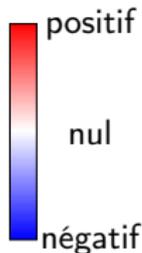


g de \mathbb{R}^2 vers \mathbb{R}

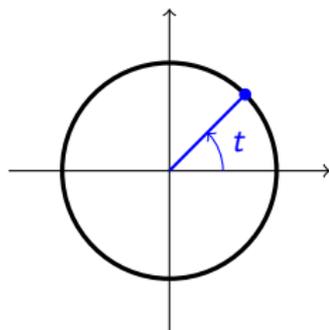
$$g(x, y) = x^2 + y^2 - r^2$$

$$C = \{(x, y) \in \mathbb{R}^2, g(x, y) = 0\}$$

implicite définition implicite d'un cercle de rayon r et de centre $(0, 0)$, en mesurant la distance entre le point (x, y) et le centre $(0, 0)$



Exemple de définition implicite / explicite : le cercle en 2D



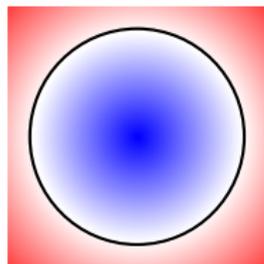
f de \mathbb{R} vers \mathbb{R}^2

$$f(t) = (r \cos t, r \sin t)$$

$$C = \{f(t), t \in [0, 2\pi[\}$$

explicite définition explicite d'un cercle de rayon r et de centre $(0, 0)$, en parcourant les points du cercle en fonction de l'angle avec l'axe des x .

Quid d'un cercle de centre (x_0, y_0) ?

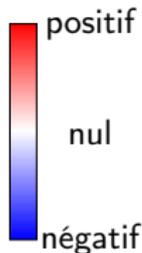


g de \mathbb{R}^2 vers \mathbb{R}

$$g(x, y) = x^2 + y^2 - r^2$$

$$C = \{(x, y) \in \mathbb{R}^2, g(x, y) = 0\}$$

implicite définition implicite d'un cercle de rayon r et de centre $(0, 0)$, en mesurant la distance entre le point (x, y) et le centre $(0, 0)$



Explicite (B-rep) vs implicite (volumique)

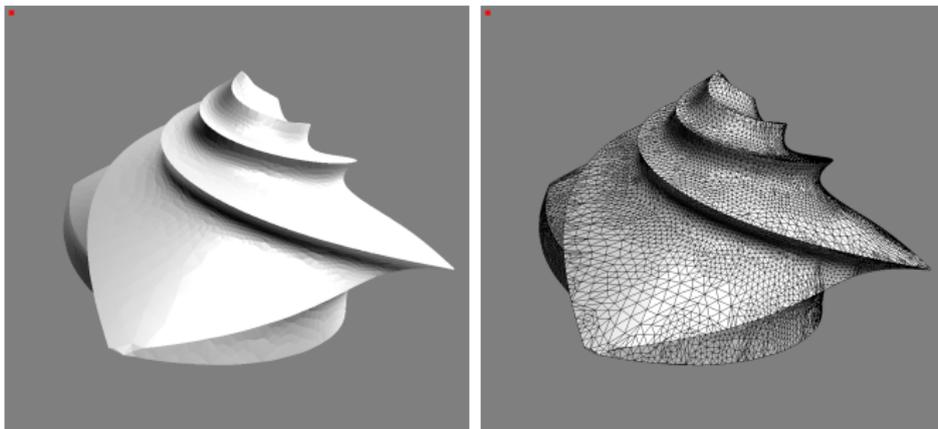
Modélisation explicite d'un objet

- on dispose ou on peut fabriquer les points de la surface de l'objet
- mathématiquement, les points ont des "coordonnées" sur la surface
- fonction d'une partie de \mathbb{R}^2 vers \mathbb{R}^3
- affichage facile, représentation compacte, texturage facile

Modélisation implicite d'un objet

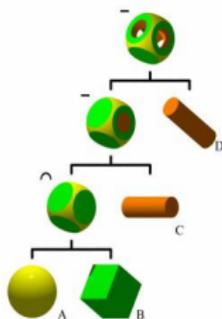
- on sait si on est dans, sur ou en dehors de l'objet
- mathématiquement, chaque point de l'espace sait sa distance à la surface
- fonction de \mathbb{R}^3 vers \mathbb{R} , négative à l'intérieur, positive à l'extérieur
- fabrication de formes par mélange facile, collisions faciles

Représentations bord (B-rep) ou “explicites”



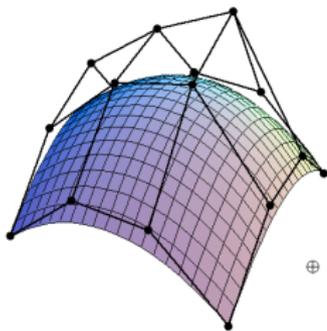
- ensemble de points
 - soupe de triangles
 - surfaces triangulées ou quadrangulées
 - surfaces paramétriques : courbes et carreaux de Bézier, splines
 - surfaces de subdivisions
- + plus compactes, plus rapides à afficher (carte graphiques), souvent plus faciles à texturer
- pas toujours pratiques pour la conception assistée par ordinateur, “lissage” des assemblages pas toujours facile

Représentations volumique ou “implicites”

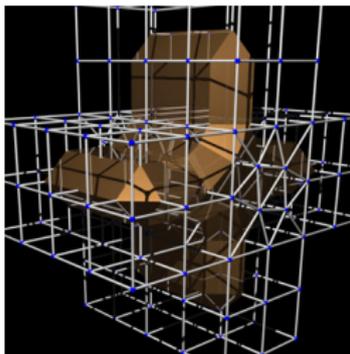


- fonctions implicites mathématiques (sphères, cubes, polynômes, etc)
- constructive solid geometry : assemblages de fonctions implicites par opérations sur ces fonctions
- images 3D (IRM, CT scans)
- feu, fumée, liquides, ...
- + formulations mathématiques facilitent certaines opérations (union, intersection, blending, etc)
- pas très compactes, pas très rapides à afficher par GPU

Transformations entre modèles



B-spline



marching cube

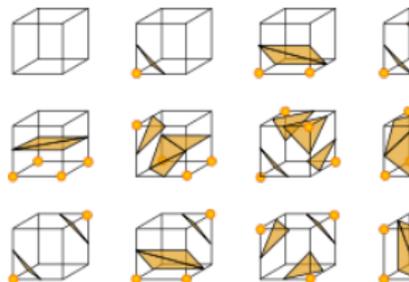
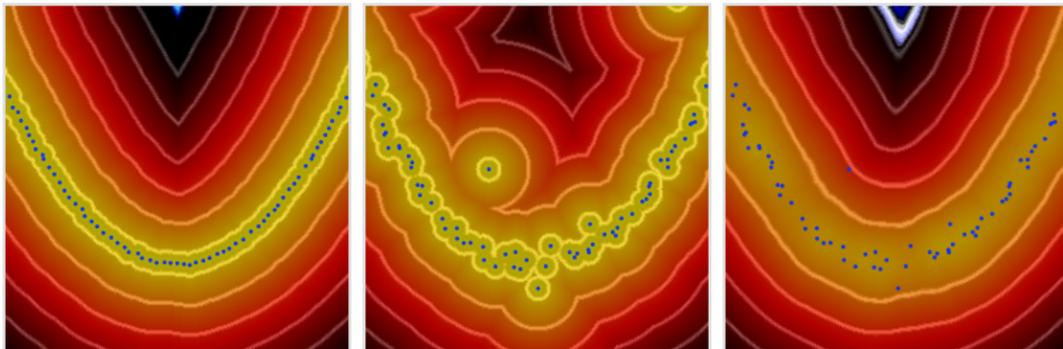


table du MC

Pour l'affichage, parfois la conversion : "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 wrapping

Transformations entre modèles



Pour faciliter les test d'intersection / inclusion / localisation, et des opérations de mélange : conversion "explicite" -> "implicite"

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

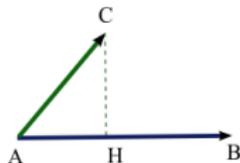
Points et vecteurs

- 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} : $\|\vec{u}\|^2 = \vec{u} \cdot \vec{u}$.
- vecteur \vec{u} **unitaire** ssi $\vec{u} \cdot \vec{u} = 1$
- 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

Propriétés du produit scalaire

Propriétés algébriques

- vecteur nul = longueur nulle : $\vec{u} = \vec{0}$ si et seulement si $\vec{u} \cdot \vec{u} = 0$
- symétrie : $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$
- bilinéarité : $\vec{u} \cdot (\vec{v} + \vec{w}) = \vec{u} \cdot \vec{v} + \vec{u} \cdot \vec{w}$
 $(\vec{u} + \vec{v}) \cdot \vec{w} = \vec{u} \cdot \vec{w} + \vec{v} \cdot \vec{w}$
 $(\alpha\vec{u}) \cdot (\beta\vec{v}) = (\alpha\beta)\vec{u} \cdot \vec{v}$

Propriétés géométriques

- **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 $\vec{u} = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|}$.

Projection sur un axe \vec{n}

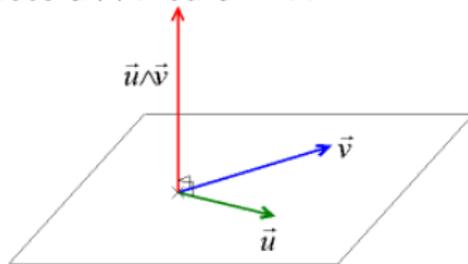
$\text{Proj}_{\vec{n}}(\vec{u}) := (\vec{u} \cdot \vec{n})\vec{n}$ si $\|\vec{n}\| = 1$.

Projection sur un plan de normal \vec{n}

$\text{Proj}_{\text{Plan}_{\vec{n}}}(\vec{u}) := \vec{u} - (\vec{u} \cdot \vec{n})\vec{n}$ si $\|\vec{n}\| = 1$.

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.

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

Propriétés du produit vectoriel

Propriétés algébriques

- antisymétrie : $\vec{u} \times \vec{v} = -\vec{v} \times \vec{u}$
- bilinéarité : $\vec{u} \times (\vec{v} + \vec{w}) = \vec{u} \times \vec{v} + \vec{u} \times \vec{w}$
 $(\vec{u} + \vec{v}) \times \vec{w} = \vec{u} \times \vec{w} + \vec{v} \times \vec{w}$
 $(\alpha\vec{u}) \times (\beta\vec{v}) = (\alpha\beta)\vec{u} \times \vec{v}$

Propriétés géométriques

- **Orthogonalité** : Si $\vec{w} = \vec{u} \times \vec{v}$ alors $\vec{w} \perp \vec{u}$ et $\vec{w} \perp \vec{v}$, i.e. $\vec{w} \cdot \vec{u} = \vec{w} \cdot \vec{v} = 0$
- **Colinéarité** : $\vec{u} \times \vec{v} = 0$ ssi \vec{u} et \vec{v} sont colinéaires

Construire un vecteur normal à un plan

Si \vec{u} et \vec{v} sont deux vecteurs d'un plan non colinéaires, alors $\vec{u} \times \vec{v}$ est un vecteur normal à ce plan.

Un exemple : la droite en 2D

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

Un exemple : la droite en 2D

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

Quid d'un cercle ?

Un exemple : le plan

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

Un exemple : le plan

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

Quid d'une boule ?

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

Matériaux, illumination, textures

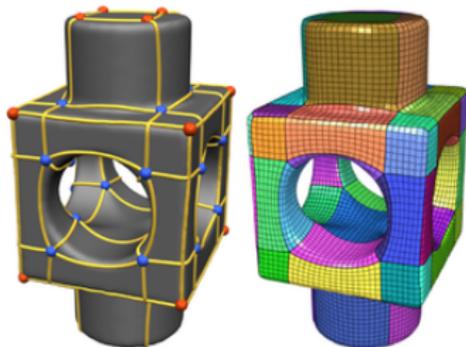
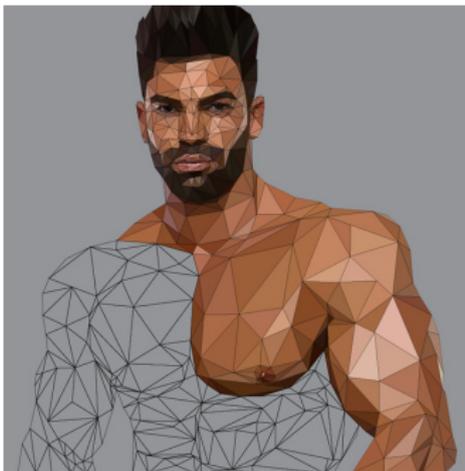
Caméra et perspective

Algorithmes de rendu

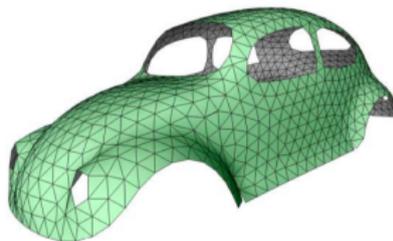
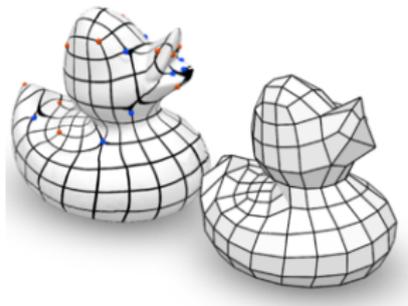
Pipe-line graphique et shaders

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

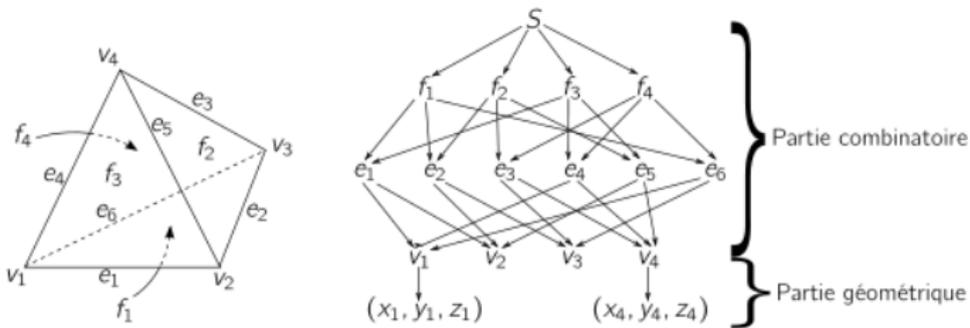
- décomposition cellulaire de surfaces
- assemblage de faces (triangles, quadrilatères, ...) le long d'arêtes terminées par des sommets
- contraintes combinatoires pour avoir la topologie d'une surface
 - l'intersection de deux faces est vide, ou un sommet, ou une arête
 - une arête n'est incidente qu'à 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

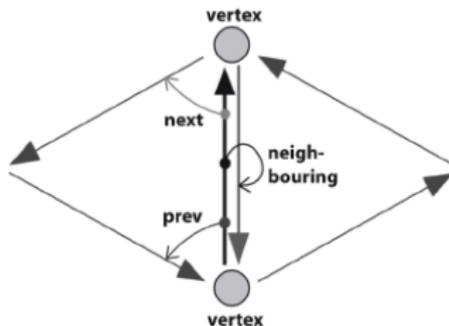
Structures de données

- soupe de triangles
- “graphe” : sommets, liste circulaire des voisins
- graphe d'incidence



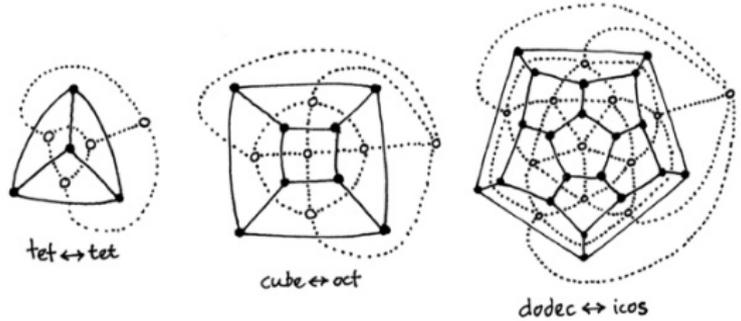
Structures de données

- soupe de triangles
- “graphe” : sommets, liste circulaire des voisins
- graphe d'incidence
- *winged-edge* structure (ou doubly-linked cell list)



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



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

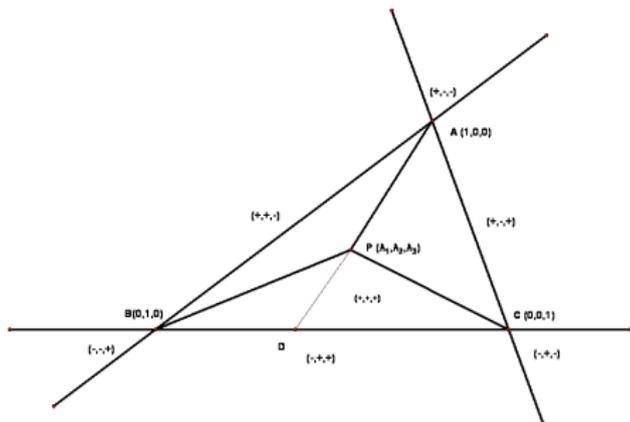
Très souvent, surfaces polygonales avec faces planes

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

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

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

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

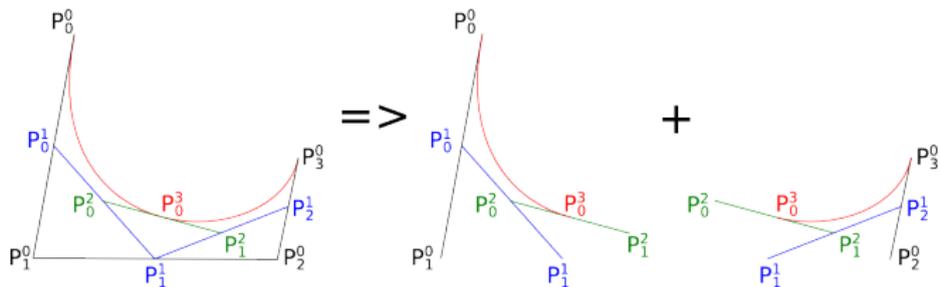


Géométrie lisse (par morceaux)

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

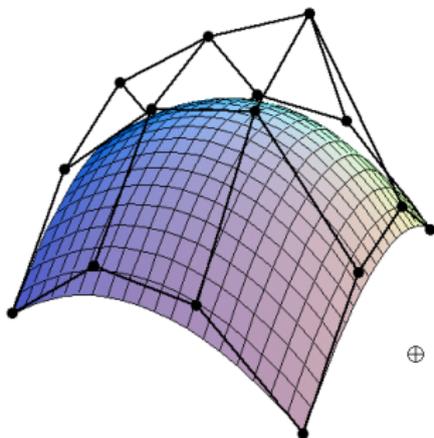
- Pour $n+1$ points de contrôle (P_0, \dots, P_n) , on définit une courbe de Bézier par l'ensemble des points $\sum_{i=0}^n B_i^n(t) \cdot P_i$, avec $t \in [0, 1]$ et où les B_i^n sont les polynômes de Bernstein.
- Si $n=1$, 2 points de contrôle : $B_0^1(t) = (1 - t)$ et $B_1^1(t) = t$
- Si $n=2$, 3 points de contrôle : $B_0^2(t) = (1 - t)^2$, $B_1^2(t) = 2(1 - t)t$, $B_2^2(t) = t^2$
- les points P_0, \dots, 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 $1/2$.

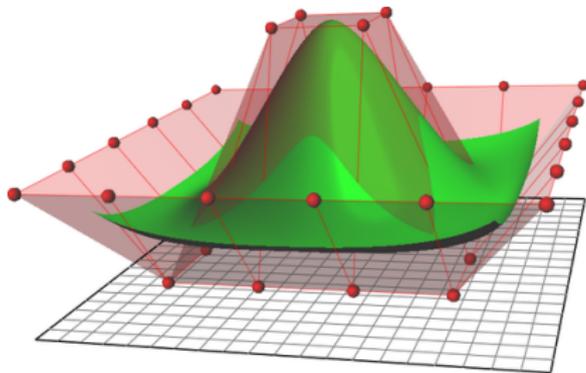
Surfaces lisses et carreaux de Bézier



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

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

NURBS (Non uniform rational B-splines)



- degré de NURBS choisi $N = \text{nb de points de contrôle par segment} = \text{degré polynôme} - 1$
 - séquence de points de contrôle $(p_i)_{i=1, \dots, n}$
 - vecteur de noeuds $(t_i)_{i=1, \dots, n}$, séquence croissante des “moments” où le point p_i est utilisé.
 - définition récursive par mélange de fonctions triangulaires de base.
 - poids sur les noeuds
- ⇒ permet de construire exactement des cercles, coniques, etc.

INFO 805 - Introduction à l'informatique graphique

Géométrie dans l'espace

Modélisation géométrique

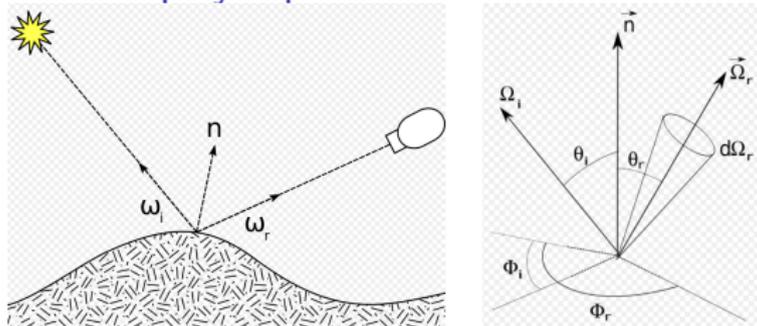
Matériaux, illumination, textures

Caméra et perspective

Algorithmes de rendu

Pipe-line graphique et shaders

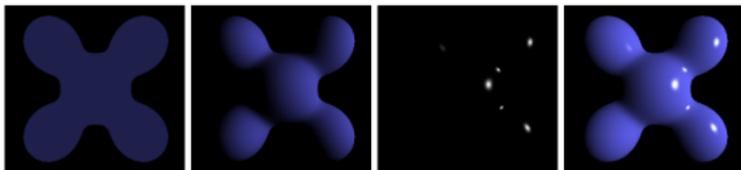
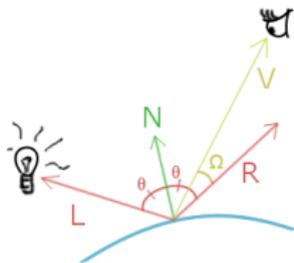
Modèle physique de couleurs : BRDF



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

$$\begin{aligned}
 L_\nu^r(\Omega_r) &= \int_{2\pi} f_\nu(\Omega_i, \Omega_r) L_\nu^i(\Omega_i) d\Omega_i \\
 &= \int_0^\pi \int_0^{\frac{\pi}{2}} f_\nu(\theta_i, \phi_i, \theta_r, \phi_r) L_\nu(\theta_i, \phi_i) \sin\theta_i d\theta_i d\phi_i
 \end{aligned}$$

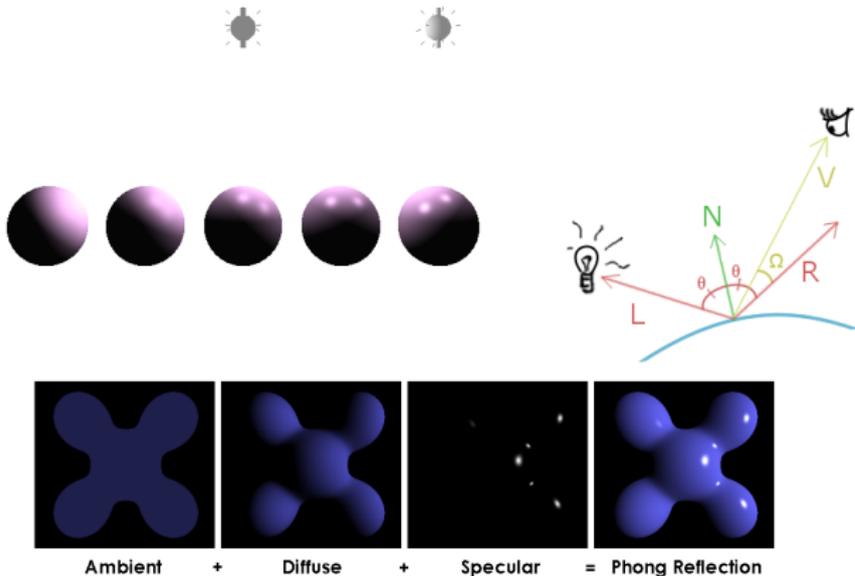
Modèle de Phong



Ambient + Diffuse + Specular = Phong Reflection

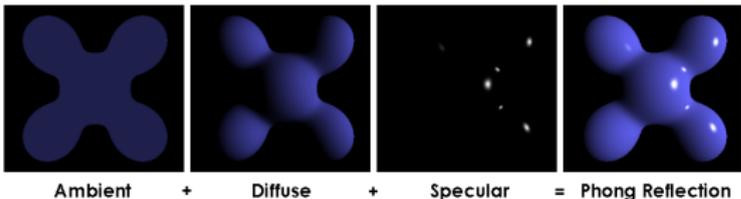
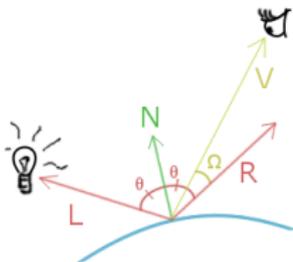
- ambient** 1 couleur sans lumière
- diffuse** 1 couleur qui dépend des lumières (donc de θ)
- spéculaire** 1 couleur qui dépend des lumières et de l'œil (de θ et Ω)

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

C_a

couleur émissive

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

$$\sum_j \underbrace{(\vec{N} \cdot \vec{L}_j)}_{\text{irradiance}} C_d * C_{lightj}$$

couleur diffuse

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

$$\sum_j \underbrace{(\vec{N} \cdot \vec{L}_j)}_{\text{irradiance}} \underbrace{(\vec{V} \cdot \vec{R}_j)^t}_{\text{tache spéculaire}} C_s * C_{lightj}$$

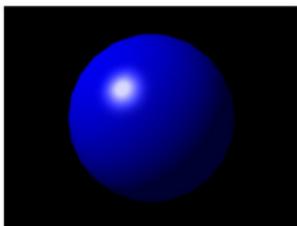
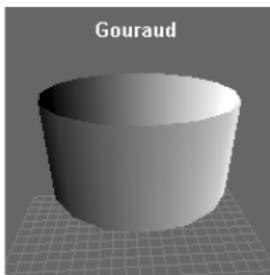
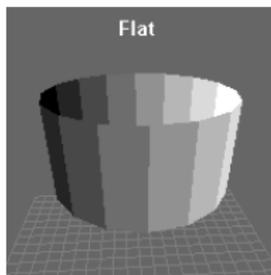
couleur spéculaire

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

Matériaux

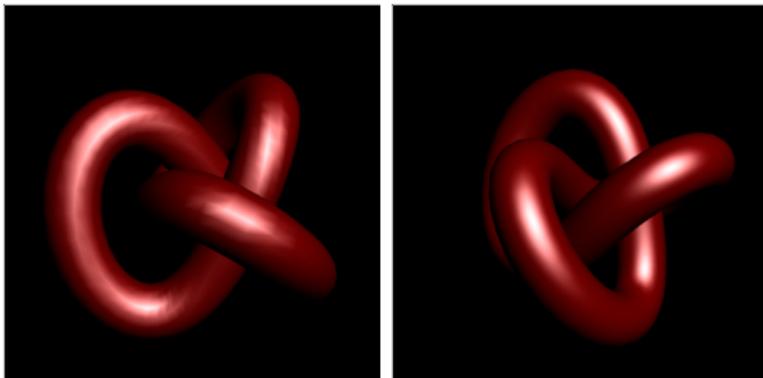


Ombrage / shading



- 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>

Displacement maps, normal maps



displacement map Texture modifiant la hauteur au-dessus de chaque point d'une facette

normal map Texture modifiant le vecteur normal de chaque point d'une facette

⇒ Normales géométriques sont modifiées par ces textures

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

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

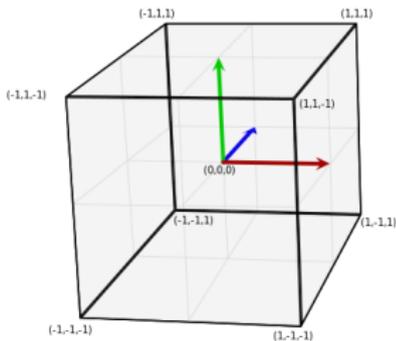


- matrice V : transforme coordonnées “camera” vers “monde”

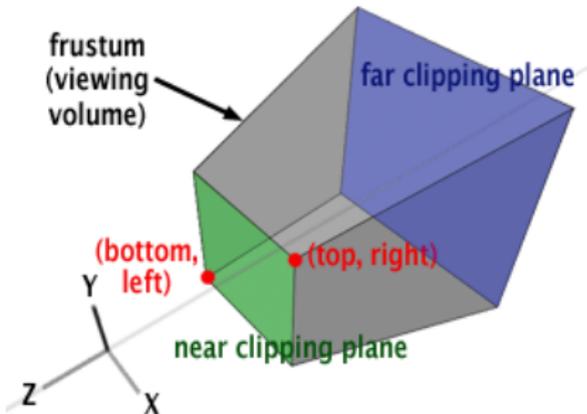
$$\begin{bmatrix} R_x & U_x & F_x & T_x \\ R_y & U_y & F_y & T_y \\ R_z & U_z & F_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Transformation en coordonnées écran : la perspective



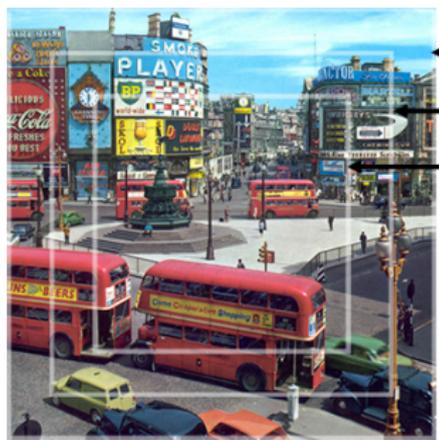
Clipspace



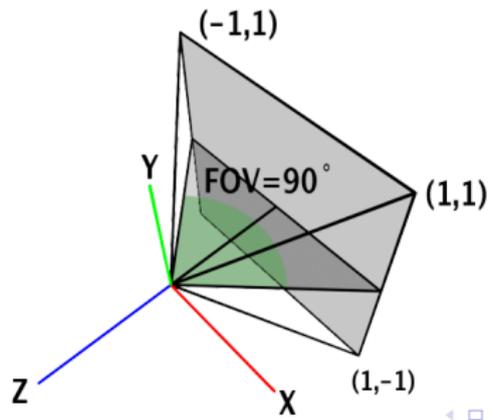
$$P := \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

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

Champ de vue de la caméra



← FOV = 110°
← FOV = 90°
← FOV = 60°



Résumé

- Suite de transformations pour amener un point X dans l'espace "pixels"
 1. model space \rightarrow model matrix $M \rightarrow$ world space
 2. world space \rightarrow view matrix $V^{-1} \rightarrow$ view space
 3. view space \rightarrow projection matrix $P \rightarrow$ 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 $X' := PV^{-1}MX$ 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

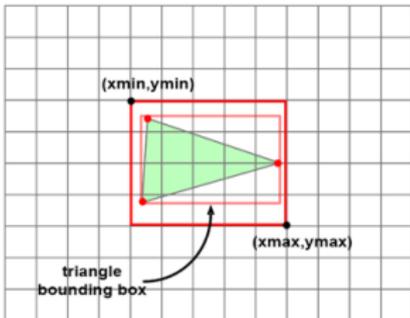
Pipe-line graphique et shaders

Principaux algorithmes de rendu

- Rendus dits “non-réalistes”
 - algorithme du peintre (historique)
 - algorithme Z-buffer (sur les cartes graphiques)
- Rendus dits “réalistes”
 - ray-tracing
 - radiosité
 - path-tracing

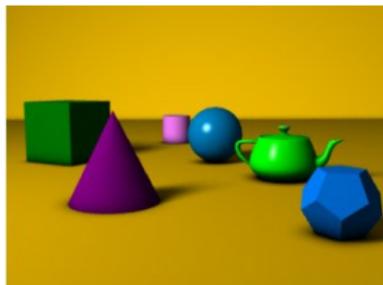
Algorithme Z-buffer

- Sommets placés dans les coordonnées écran :
 $X' := (x', y', z') = PV^{-1}MX$
- Dans ce cube : (x', y') coordonnées écran, z' profondeur
- Chaque polygone est rasterisé :



© www.scratchapixel.com

- couleur/profondeur interpolées
- le pixel est affiché si $z' < Z[x'][y']$



Une scène 3d simple



Le Tampon de profondeur

Tableau Z en bas

Coordonnées barycentriques, rasterisation

- rastérisation + interpolation
- coordonnées barycentriques (u, v, w)

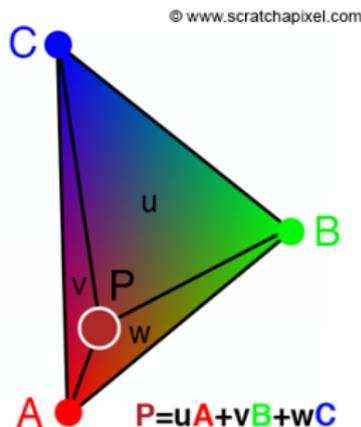
$$u = \frac{\text{Area}(PBC)}{\text{Area}(ABC)}$$

$$v = \frac{\text{Area}(PCA)}{\text{Area}(ABC)}$$

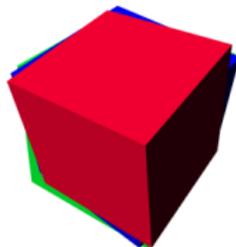
$$w = \frac{\text{Area}(PAB)}{\text{Area}(ABC)}$$

$$\text{position } P = uA + vB + wC$$

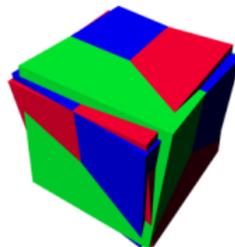
- pour tout pixel $\text{bbox}(ABC)$
- pixel $P \in ABC$ ssi $u, v, w \geq 0$
- couleur $C = uC_A + vC_B + wC_C$
- profondeur
 $Z = uZ_A + vZ_B + wZ_C$



WITHOUT Z-TEST



WITH Z-TEST



Algorithme ray-tracing

- Envoi des rayons lumineux de l'observateur dans la scène
- Les rayons rebondissent jusqu'à atteindre une lumière

Algorithme ray-tracing

- Envoi des rayons lumineux de l'observateur dans la scène
- Les rayons rebondissent jusqu'à atteindre une lumière

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