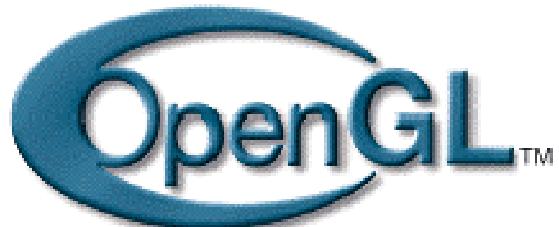


Programmation de la 3D



Alexandre Topol

Spécialité Informatique

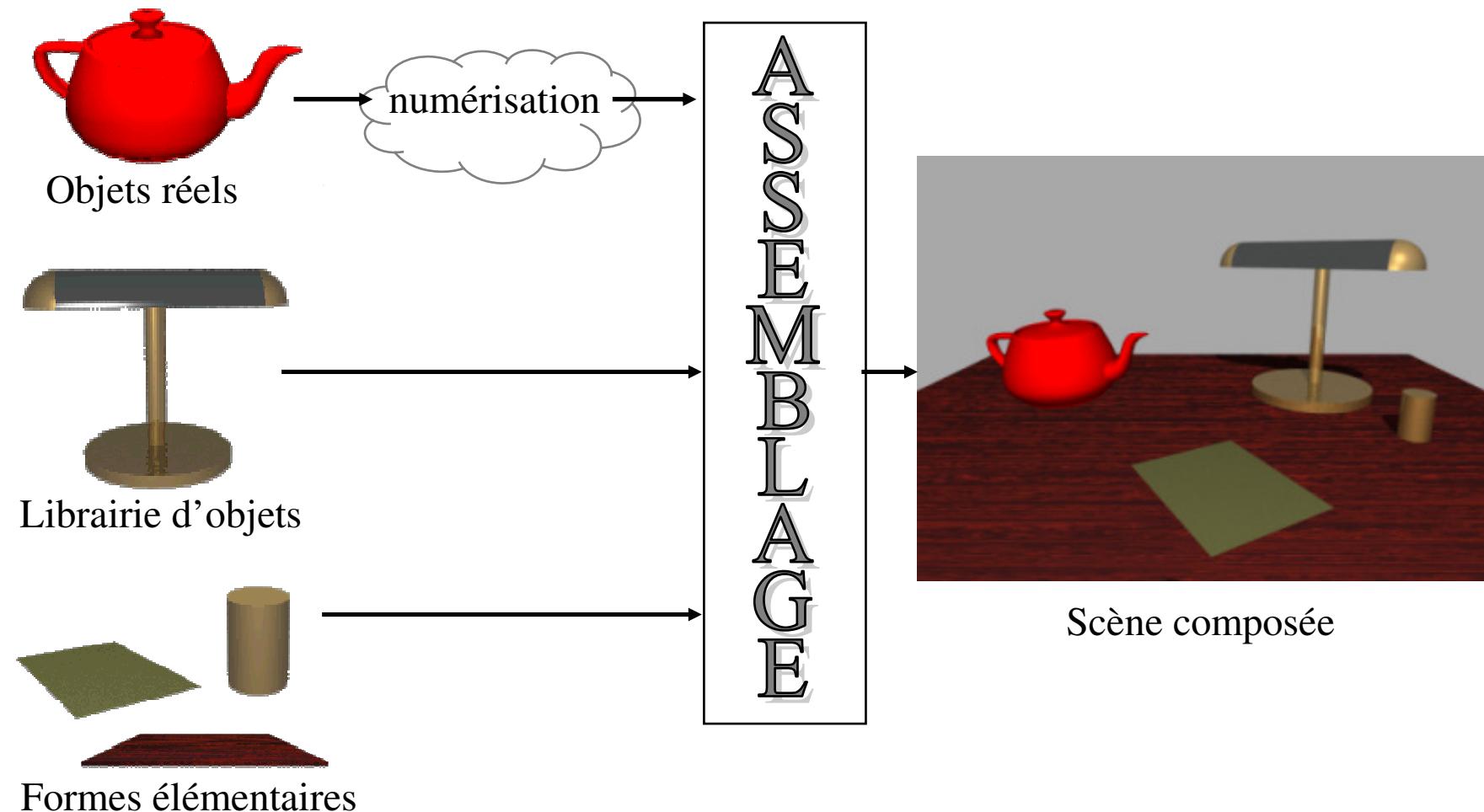
Conservatoire National des Arts & Métiers

2011-2012

Introduction

Comment ça marche ?

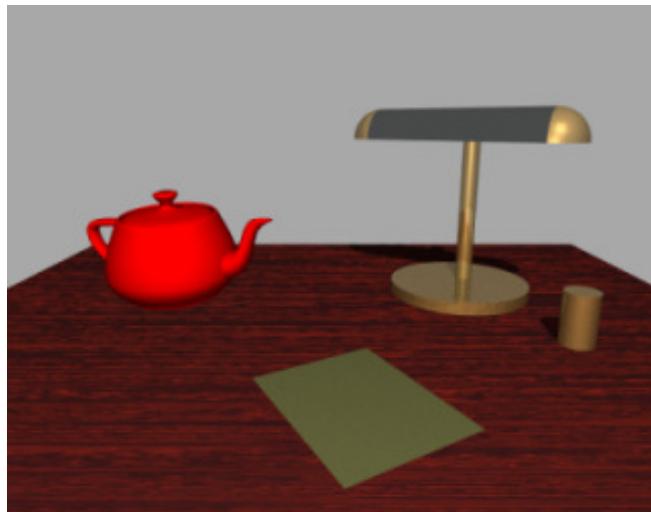
le cnam



Introduction

Comment ça marche ?

le cnam



**R
e
n
d
u
3
D**

*Raytracing
Radiosité*

"3D temps réel"

2 méthodes de rendu

le cnam

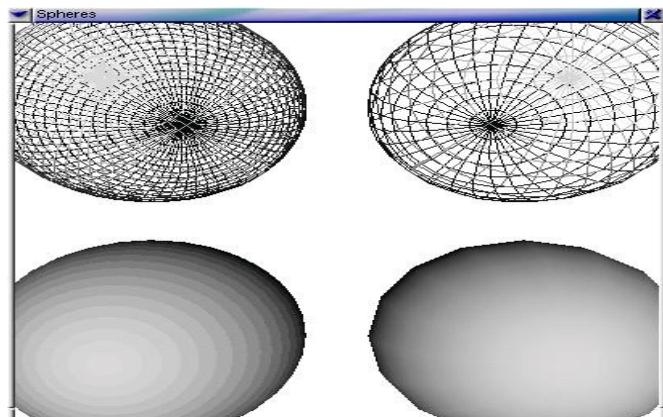
- Projection de facettes :
 - Les objets doivent être "triangulisés" (*tessaled*)
 - Et tout triangle ou polygone subit les opérations du pipeline 3D
 - Transformations, visibilité, *clipping*, projection, coloriage
- Lancé de rayons et radiosité :
 - On utilise ici les équations de surface des objets donc pas besoin de "trianguliser" les objets
 - Calcul pour chaque pixel de l'écran de l'équation partant de l'observateur et passant par ce pixel
 - Pour chaque rayon, calcul des intersections avec les objets pour déterminer la couleur du pixel

Visualisation des scènes : deux méthodes

le cnam

Par facettage

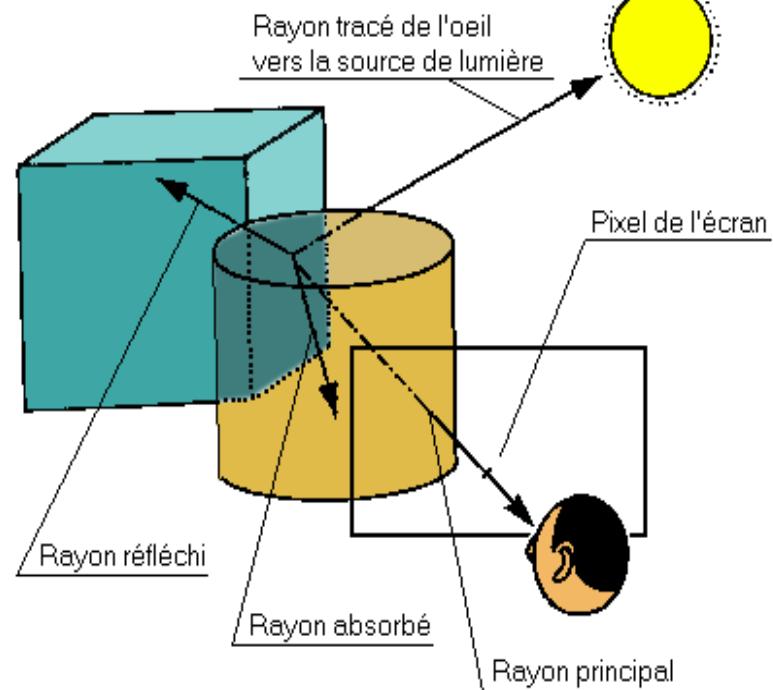
1. Transformation des formes complexes en ensemble de triangles



2. Transformation, illumination, projection et coloriage de chacun des triangles → pipeline 3D

« quantité plutôt que qualité »

Raytracing



« qualité plutôt que quantité »

Visualisation des scènes : deux méthodes

le cnam

Par facetage



© Activision

« quantité plutôt que qualité »

Raytracing



© Dreamworks

« qualité plutôt que quantité »

Description des objets

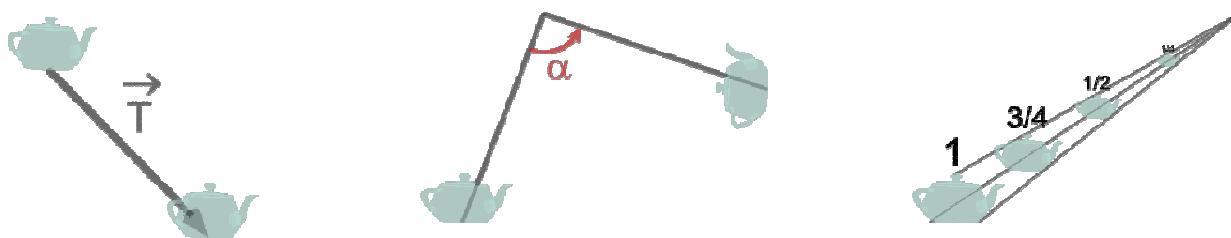
le cnam

- Elle est quasi identique quelque soit la méthode employée pour les visualiser
- La seule exception est l'utilisation des opérations booléennes (surtout dans le lancé de rayons)
- Les deux étapes :
 - Décrire les objets
 - Par liste de sommets et de faces, par profil, par fractales, ...
 - Composer la scène
 - Par positionnement des objets les uns par rapport aux autres

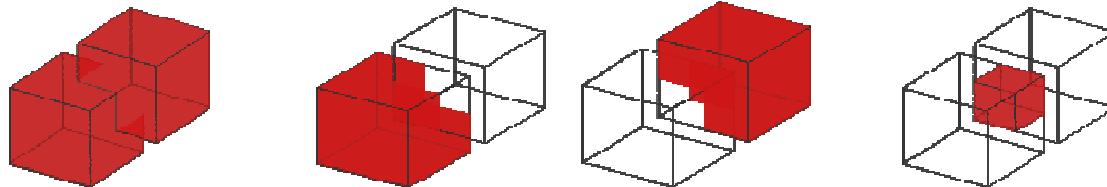
Composition de la scène

le cnam

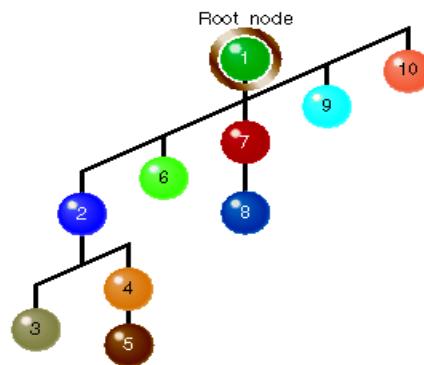
- Par transformations géométriques



- Par opérations booléennes



- Dans un graphe de scène



Visualisation des scènes

Projection de facettes

le cnam

■ L'algorithme :

Initialisation()

 Chargement de la scène

 Pour chaque objet Faire

 Décomposer en polygones (triangles) si nécessaire
(si l'objet n'est pas déjà constitué de faces)

 Fin Pour

Rendu()

 Initialiser Z-Buffer et FrameBuffer (l'image)

 Pour chaque triangle Faire

1. Transformations - se placer dans le repère de l'observateur
2. Test de visibilité - rejet des faces non visibles
3. Clipping - rejet des parties du hors du volume de vue
4. Calcul de l'intensité lumineuse
5. Projection
6. Z-Buffer et Coloriage

 Fin Pour

 Afficher l'image

Visualisation des scènes

Lancé de rayons

le cnam

■ L'algorithme (de base) :

```
Pour chaque pixel de l'écran Faire
    Calculer l'équation de la droite (point de vue, pixel)
    maxlocal := -oo
    Pour chaque objet de la scène Faire
        pz = intersection de valeur Z minimum entre le rayon et l'objet
        Si pz existe et pz >= maxlocal Alors
            maxlocal = pz
            pixel (i, j) = couleur de la face
        Fin Si
    Fin Pour
Fin Pour
```

Exemple

Accès à ces méthodes de rendu 3D décrite ou programmée ?

le cnam

■ 2 manières de mettre en œuvre ces 2 méthodes :

- La 3D programmée : utilisation de fonctions dans un programme pour donner à la fois :
 - Les caractéristiques des objets
 - L'ordre de rendu
- La 3D décrite :
 - Spécification de la scène dans un fichier
 - Interprétation et rendu de la scène par un programme spécialisé
 - Choix d'un format cible (VRML, POV, 3DS ...)

La 3D décrite ou programmée ?

La 3D programmée

- Utilisation d'une API 3D : OpenGL, Direct3D, Java3D, ...
- Avantages :
 - Accélération graphique
 - Contrôle sur le rendu - application "sur mesure"
- Inconvénients :
 - Nécessite la connaissance d'un langage de programmation
 - Difficulté de maintenance
 - Pauvre "réutilisabilité"
 - Temps de développement

La 3D décrite ou programmée ?

La 3D décrite

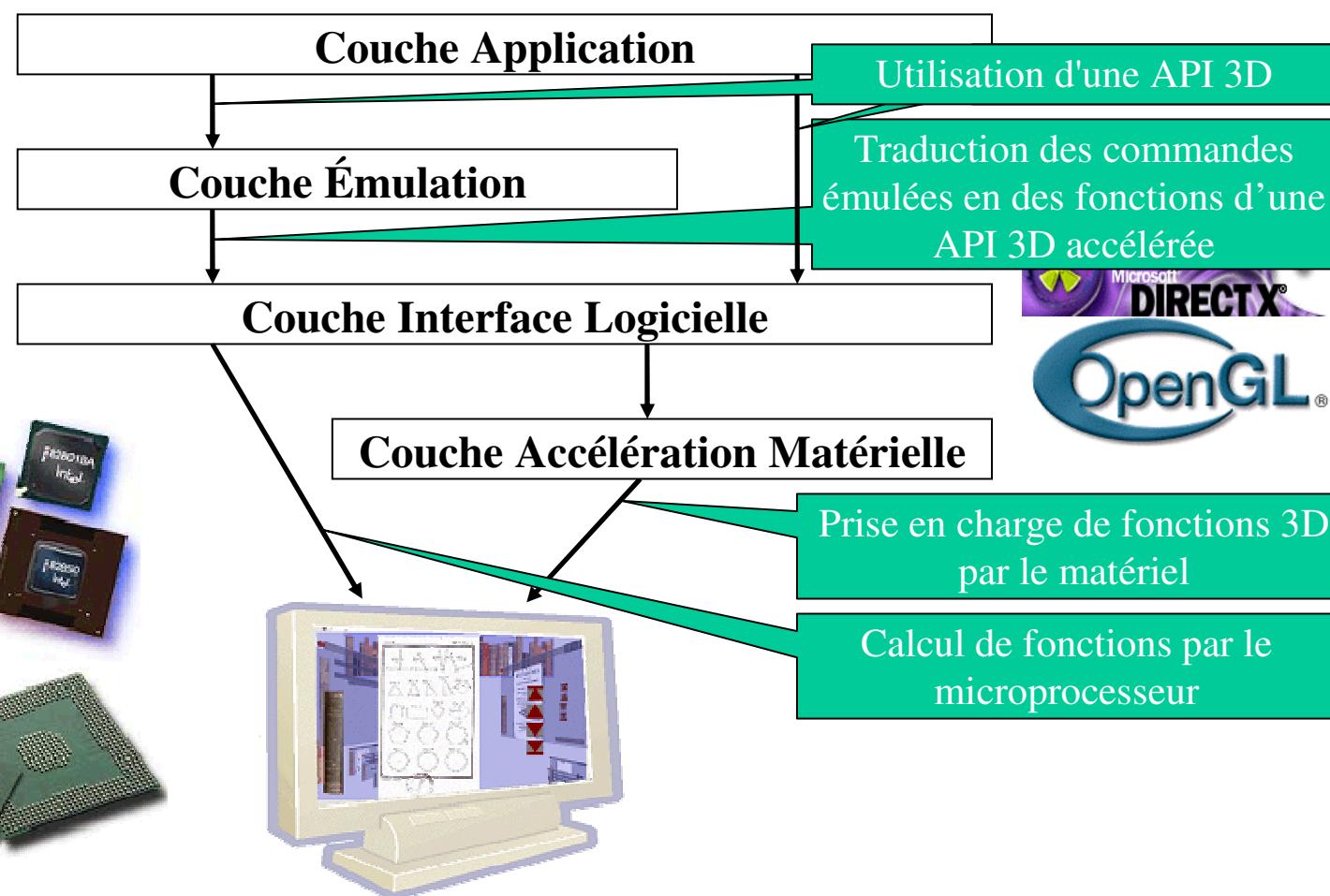
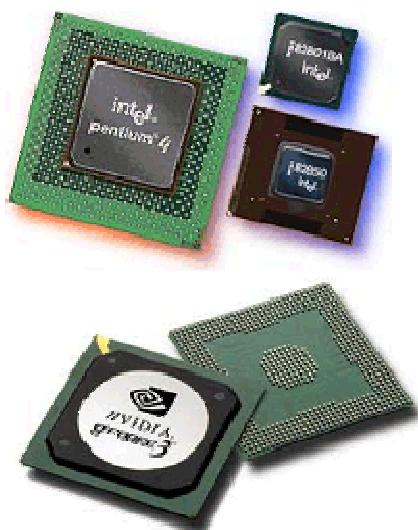
le cnam

- Utilisation d'un langage de description et d'un programme ou plugin associé : VRML, Collada, ...
- Avantages :
 - Bonne description de l'apparence des objets
 - Outils pour la création, l'édition et le rendu
 - Diffusion sur internet – génération par scripts CGI
 - Facilité pour la maintenance
 - "Réutilisabilité" - partage
- Inconvénients :
 - Pauvre description des interactions
 - N'inclut pas les dernières techniques réalistes

La 3D décrite ou programmée ?

Les couches d'une architecture 3D

le cnam



La 3D décrite Lancé de rayons

- Des modeleurs (3DS Max, Maya, Blender, ...)
 - Construction des scènes par clics de souris
 - Pré-visualisation en 3D par facettage
 - Calcul de l'image ou de l'animation finale par lancer de rayons
 - POV (*Persistance Of Vision*)
 - Construction des scènes par langage
 - Pas de pré-visualisation
 - Mais ... gratuit et sources disponibles
- <http://www.povray.org>

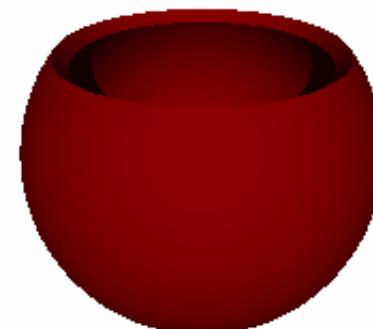
La 3D décrite Lancé de rayons – POV

le cnam

■ Tout y est et bien plus ...

- Formes élémentaires : polygon, sphere, cone, torus ...
- Formes complexes : splines, surfaces d'élévation, ...
- Opérations booléennes : union, difference, intersection
- Modèles de coloriage : flat, phong, bump mapping, ...

```
camera {  
    location <0.0, 0.0, -3.0>  
    look_at <0.0, 0.0, 0.0>  
}  
  
light_source { <0, 0, -100> rgb <1.0, 1.0, 1.0>  
}  
  
background { rgb <1.0, 1.0, 1.0> }  
  
difference {  
    difference {  
        sphere { <0, 0, 0>, 1 }  
        sphere { <0, 0, 0>, 0.9 }  
    }  
    cone { <0, 1.1, 0>, 1.3, <0, 0, 0>, 0 }  
    pigment { rgb <0.8, 0.0, 0.0> }  
    rotate <-30, 0, 0>  
}
```



La 3D décrite 3D par facettage – VRML, X3D, Collada

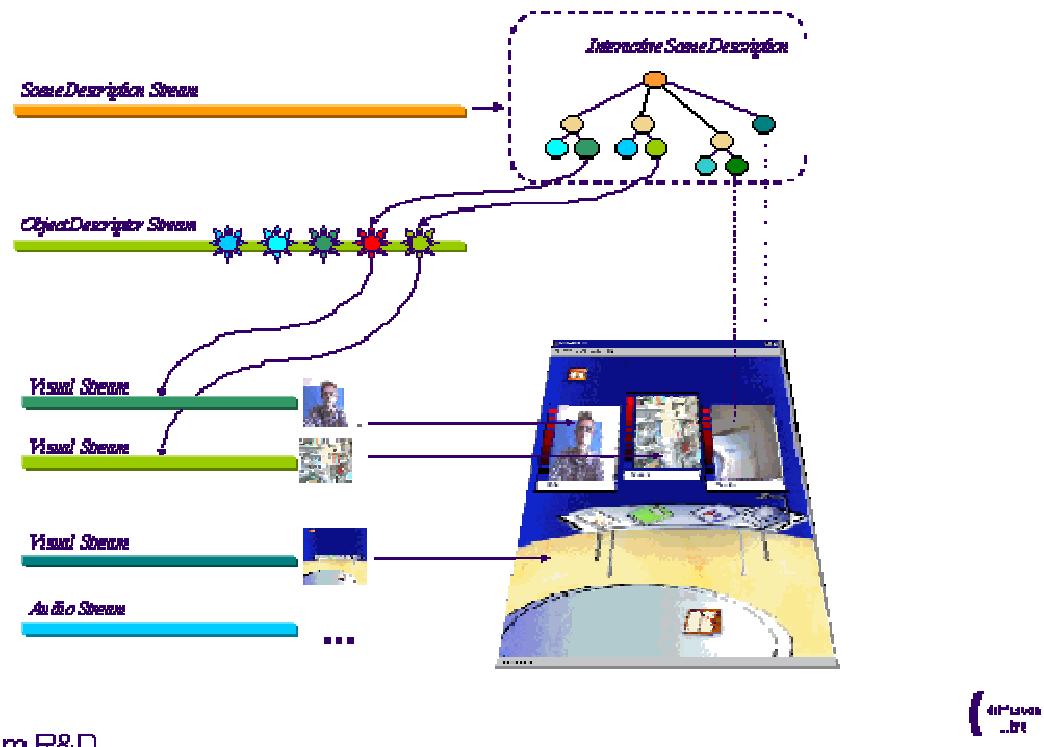
- Ce sont des formats de fichier et non des langages !
 - Un *plugin* associé permet de les visualiser
 - Organisation hiérarchique d'objets contenant des attributs
 - Les nœuds fils et leur descendance subissent tous le comportement défini par les attributs de leur nœud père
⇒ cascade de comportements
- 

La 3D décrite 3D par facettage – MPEG4

le cnam

MPEG-4 Systems Principles

&



France Télécom R&D

La communication des flux et sources d'images de France Télécom R&D
(Réf. du 1/10) - 03/01/2000

La 3D programmée

Interface de programmation (API) 3D

le cnam

- Gestion du pipeline 3D par appels de fonctions
- Interface avec un langage (C++, Java, Ada...)
- Utilisation des drivers de la carte 3D pour accélérer le rendu
- Différentes architectures cibles

Retour à l'algorithme de base

le cnam

■ L'algorithme :

Initialisation()

 Chargement de la scène

 Pour chaque objet Faire

 Décomposer en polygones (**triangles**) si nécessaire
(si l'objet n'est pas déjà constitué de faces)

 Fin Pour

Rendu()

 Initialiser **Z-Buffer** et **FrameBuffer** (l'image)

 Pour chaque triangle Faire

1. Transformations - se placer dans le repère de l'observateur
2. **Test de visibilité** - rejet des faces non visibles
3. **Cloturage** - rejet des parties du hors du volume de vue
4. Calcul de l'intensité lumineuse
5. **Projection**
6. **Z-Buffer et Coloriage**

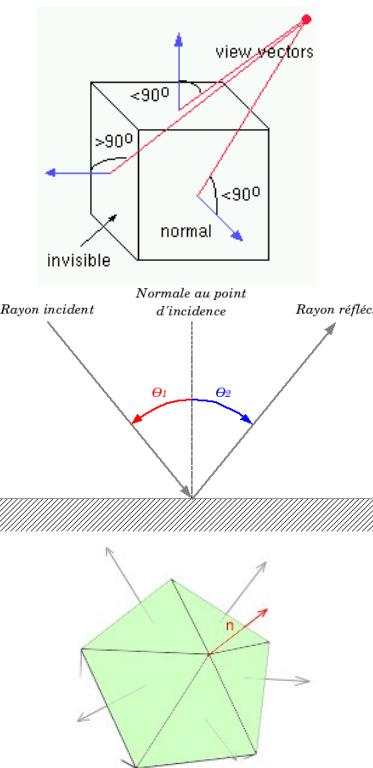
 Fin Pour

 Afficher l'image

Pourquoi des triangles ?

le cnam

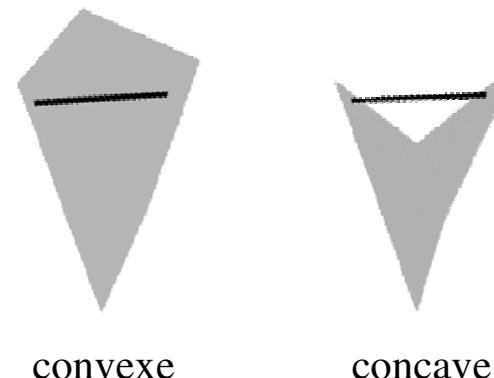
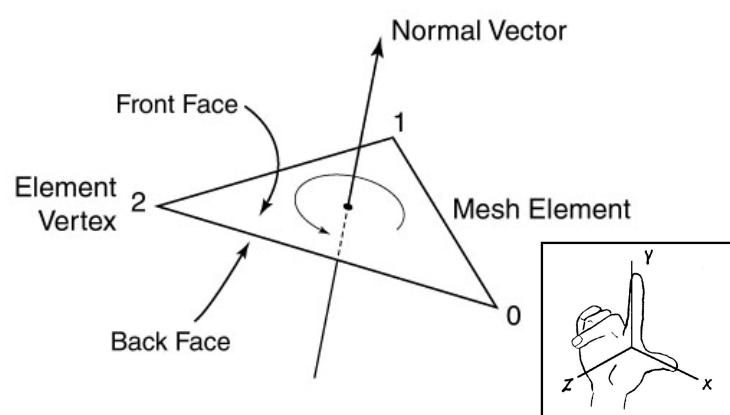
- On a besoin des normales aux faces pour pas mal de calculs :
 - détermination des faces vues de dernière (backface culling)
 - d'illumination des faces (basés sur la loi de Descartes)
 - les calculs des normales aux sommets pour le coloriage Gouraud ou Phong



Pourquoi des triangles ?

le cnam

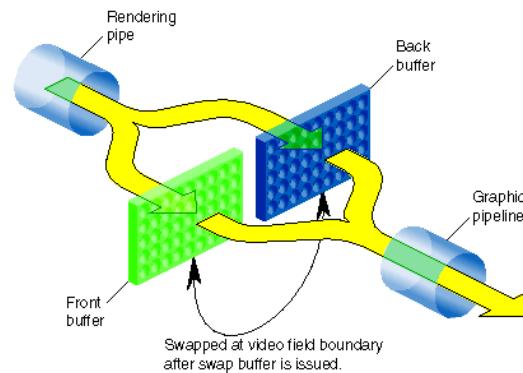
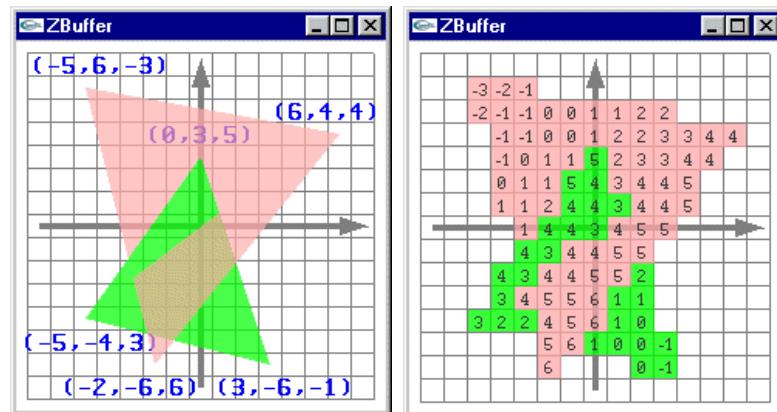
- Or, les sommets d'un triangle sont nécessairement coplanaires
 - donc une seule normale pour tout point sur la face
 - les autres polygones peuvent être vrillés
- De plus, un triangle est nécessairement convexe => plus facile à colorier



FrameBuffer et ZBuffer

le cnam

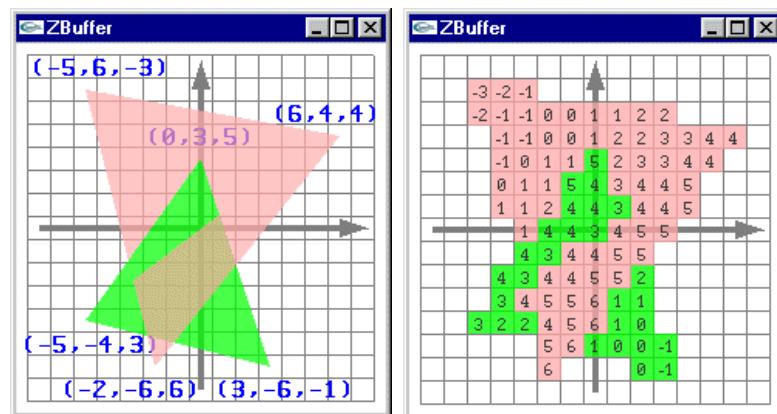
- Les deux tableaux de valeurs les plus importants pour le calcul d'une image
 - Framebuffer = valeur des pixels à afficher
 - En pratique 2 framebuffers (celui affiché et celui dans lequel on calcule l'image suivante)
 - ZBuffer = profondeur de ces pixels



FrameBuffer et ZBuffer

le cnam

- Le ZBuffer permet l'élimination des faces cachées par d'autres (ce que ne fait pas le backface culling)
 - Après projection d'un nouveau triangle, on regarde dans le ZBuffer pour chacun de ces pixels (par balayage) si ils sont plus proches de l'observateur que les pixels précédemment calculés.
 - Si oui, on stocke les valeurs de pixel et de profondeur dans les buffers
 - Sinon, on ne fait rien.

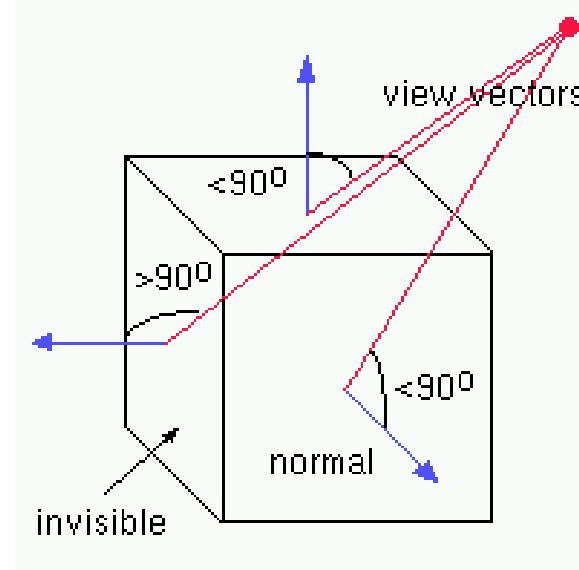
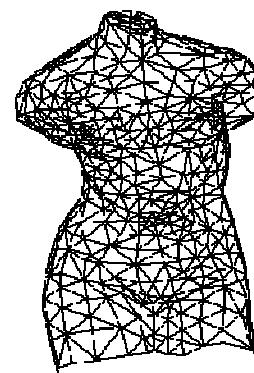
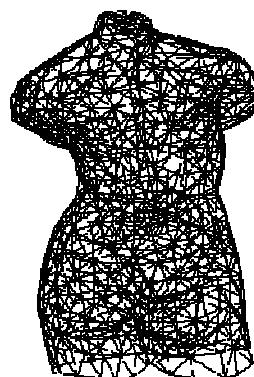


Exemple

Test de visibilité – *Backface Culling*

le cnam

- Étape optionnelle
- Élimination du pipeline 3D des faces vues de derrière :
 - Si la face est dirigée vers l'observateur (la caméra) on la garde
 - Sinon, on la rejette et cette face n'est pas projetée
- ⑩ → Utile pour les mondes clos,
pour les objets transparents
ou les objets fil de fer

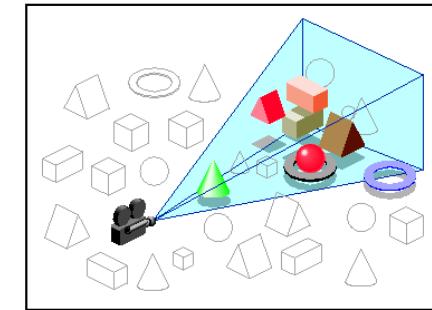
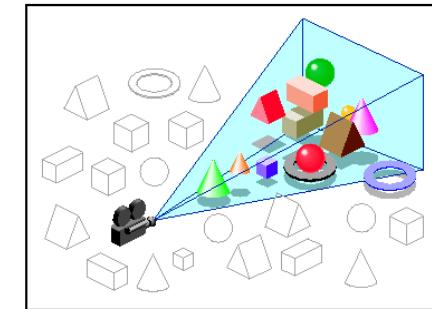
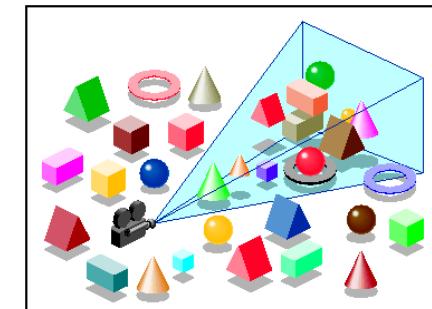
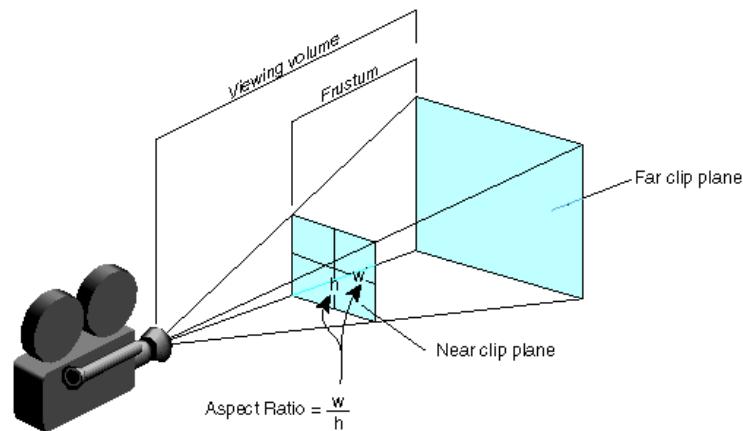


Cloturage – Clipping

le cnam

■ Volume de vue et *View Frustum*

- C'est le volume donnant la portion de l'espace dans laquelle les objets sont visibles (en fonction de la caméra)

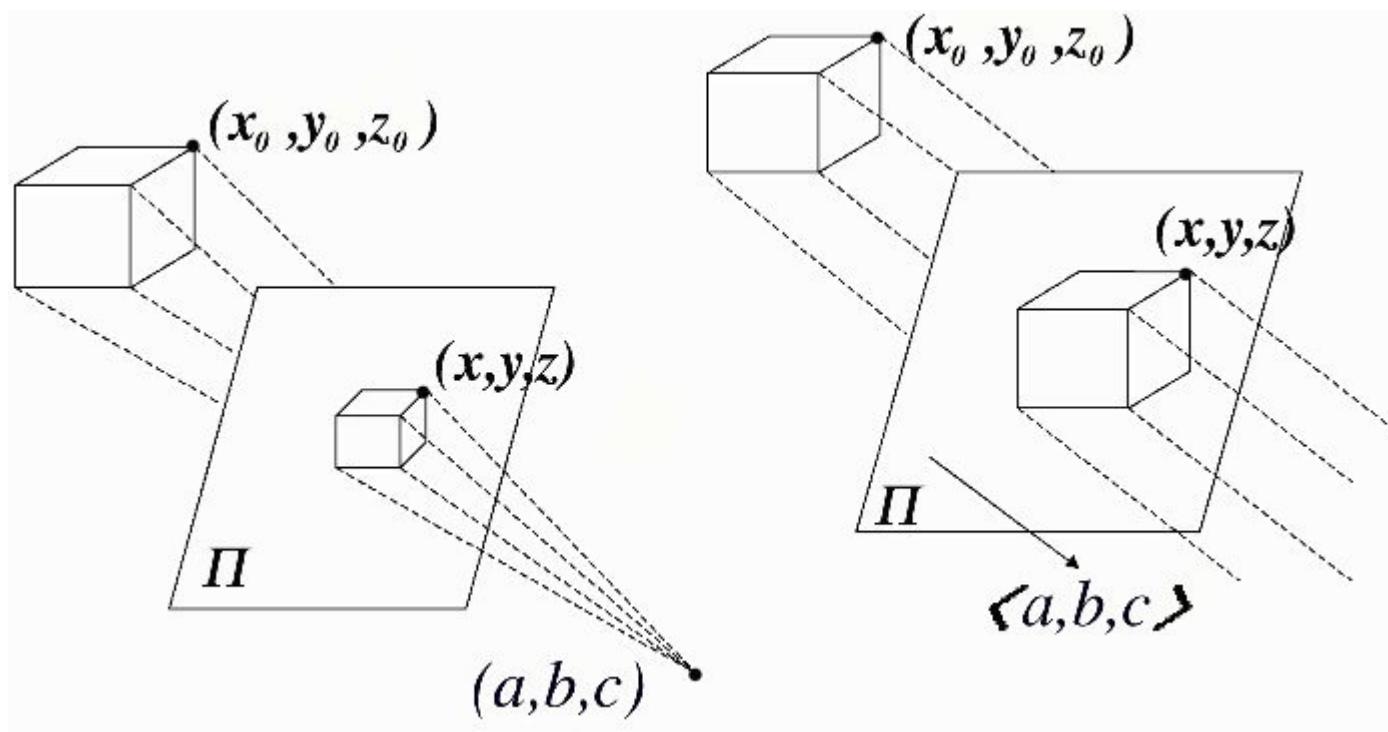


Effets combinés du clipping et du ZBuffer

Projection

le cnam

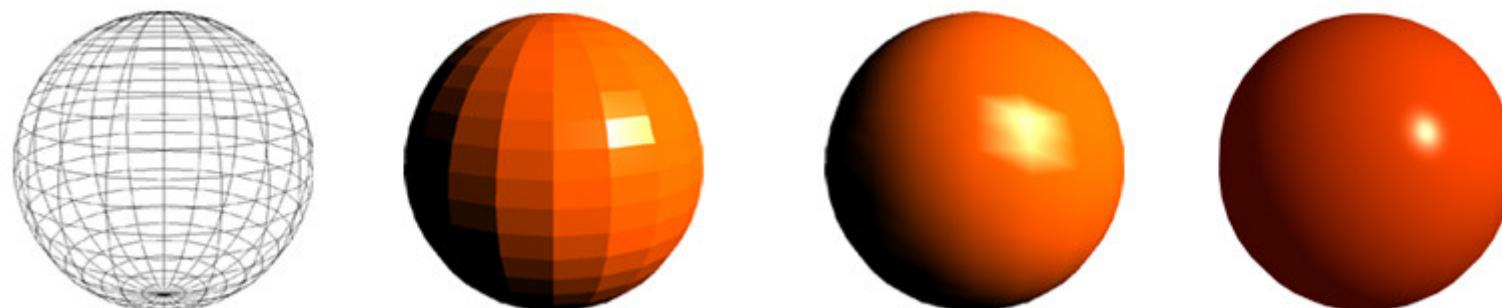
- Calcul de la position d'un point 3D sur l'écran 2D simulant l'image résultante



Coloriage – *Rasterization*

le cnam

- Balayage de la face projetée en vue de calculer une couleur pour chacun de leur pixel visible (qui réussit le test du ZBuffer)
- Beaucoup de techniques de coloriages
- Les plus simples : wireframe, flat shading, gouraud shading et phong shading



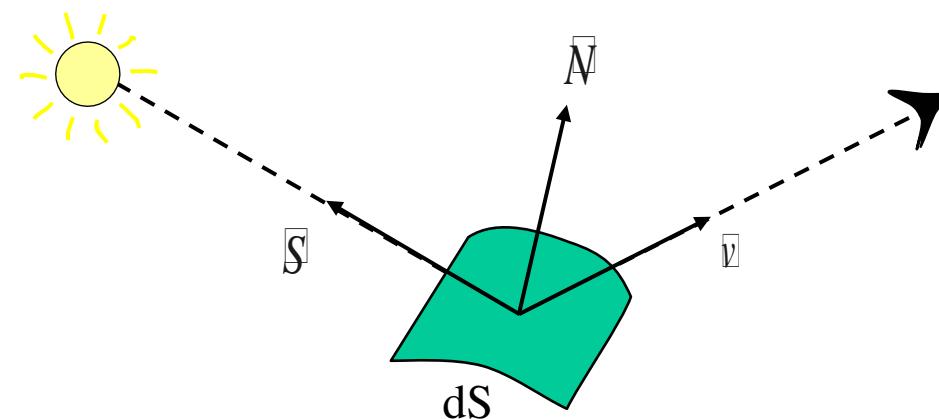
Lumière et réflexion

le cnam

■ Interaction lumière / objet

- une partie de la puissance lumineuse reçue est absorbée par l'objet et convertie en chaleur
- une partie est réfléchie par la surface
- le reste est transmis à l'intérieur de l'objet (réfraction)

■ Première approximation en 3D : la lumière ambiante est le produit des multiples réflexions dans la scène

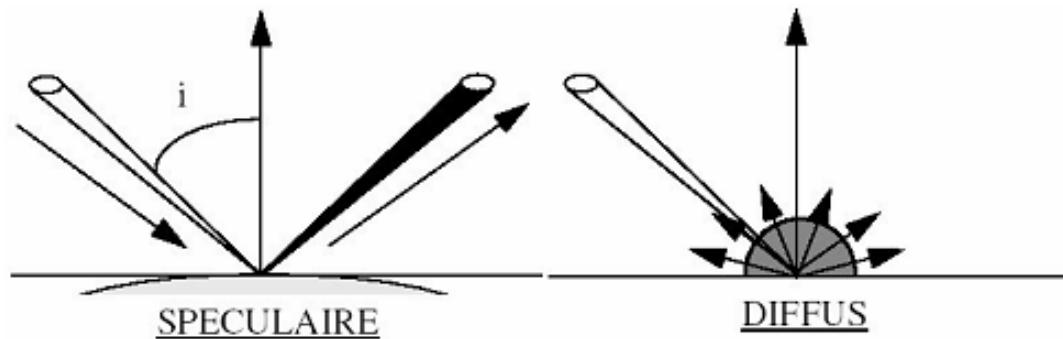


Lumière et réflexion

le cnam

■ Deux types de réflexion :

- Diffuse : La surface de l'objet "réagit". La lumière est ré-émise dans toutes les directions. Sa couleur est affectée.
- Spéculaire : La surface de l'objet ne "réagit" pas. La lumière est ré-émise selon l'angle d'incidence. Sa couleur n'est pas affectée.

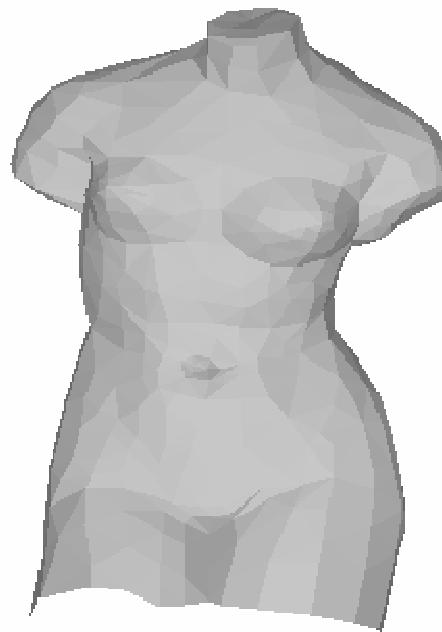


Exemple

Le coloriage à plat (*flat shading*)

le cnam

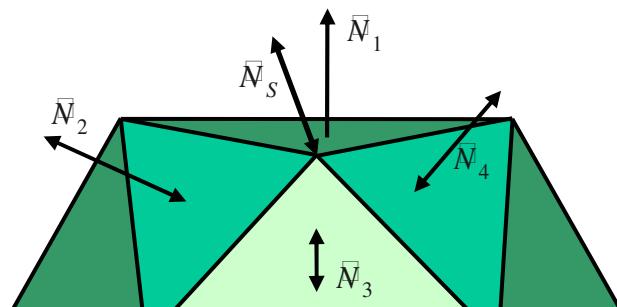
- Bouknight 1970
- On utilise la loi de lambert. L'intensité réfléchie est calculée pour chaque face et supposée constante pour toute la surface du polygone
- La face est plus ou moins éclairée en fonction de son orientation par rapport aux sources lumineuses
- L'approximation est valide si
 - la source est infiniment loin
 - l'observateur aussi
 - la surface n'est pas réfléchissante



Gouraud (*smooth shading*)

le cnam

- Principe : interpolation linéaire de l'intensité pour éliminer les discontinuités visibles de face en face
- Suppose la connaissance de l'intensité lumineuse des sommets **donc** de leur orientation par rapport à la lumière **donc** de leur normale

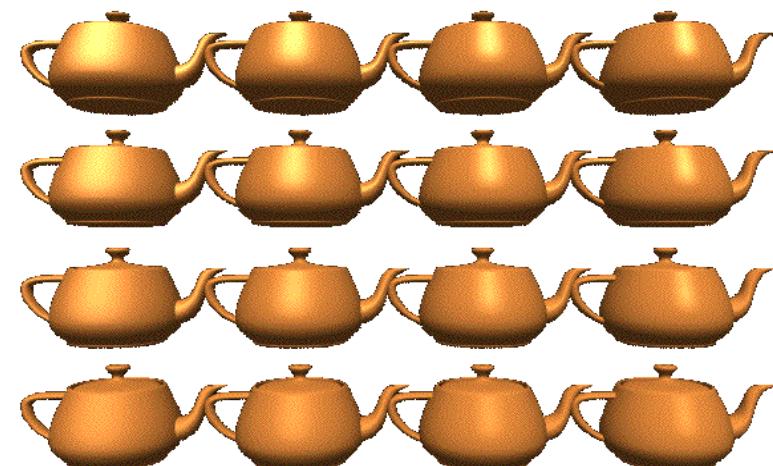


Modèle de Phong Bui-Tuong

le cnam

■ *Phong Shading* (1975), le principe :

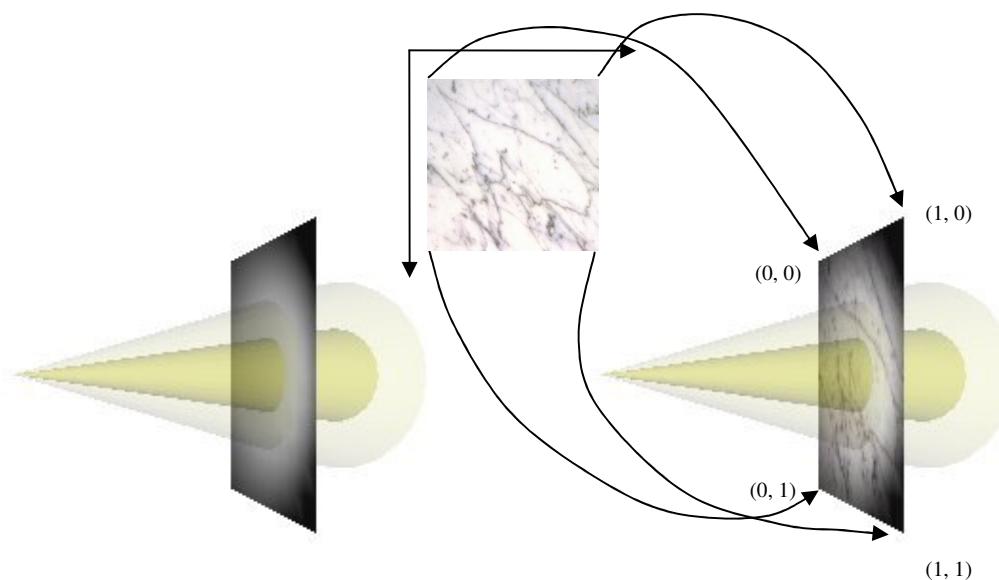
- Calcul d'une réflexion spéculaire en chaque point de la face, variable selon le matériau
- Donne un objet avec une réflexion spéculaire mais pas entièrement, d'où l'aspect de brillance locale



Placage de textures

le cnam

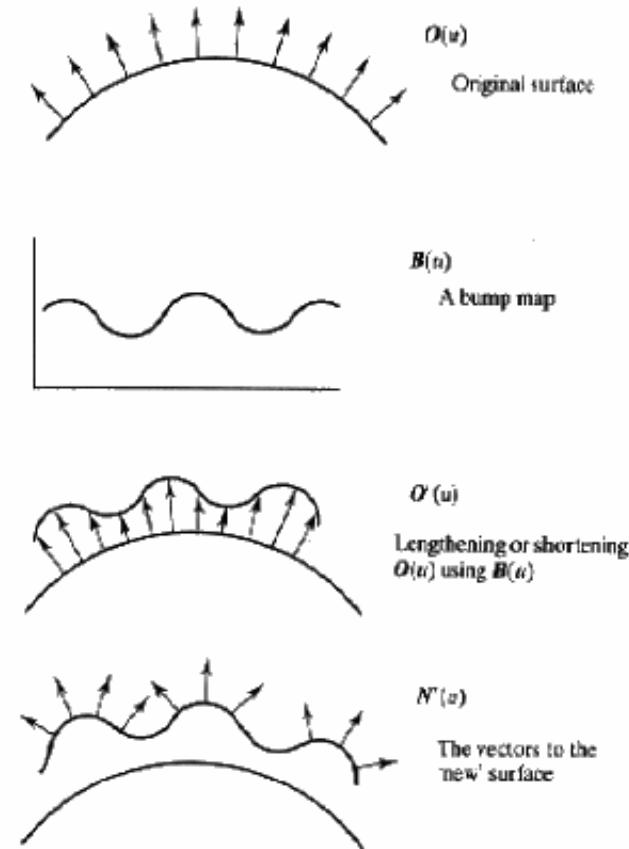
- Les algorithmes précédents ne retirent pas complètement le caractère "artificiel" du coloriage.
- Une solution souvent employée : le plaqué de textures sur les faces



Bump mapping

le cnam

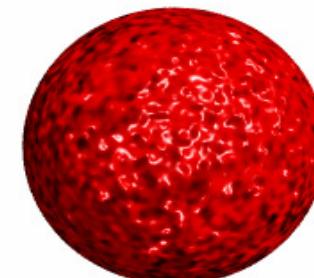
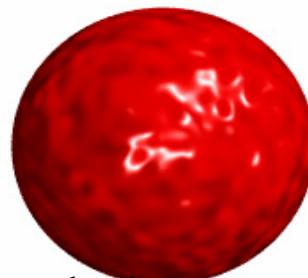
- Modèles de Davies (1954) et de Torrance & Sparrow (1966)
- Perturber la normale à la surface
- ⑩ → la lumière est donc réfléchie différemment d'un pixel à l'autre
- ⑩ → effet de relief créé par l'éclairage
- Plusieurs techniques :
 - plus simple : perturbation aléatoire
 - plus générale : un tableau indiquant pour chaque point de la surface comment se fait la perturbation (bump map)



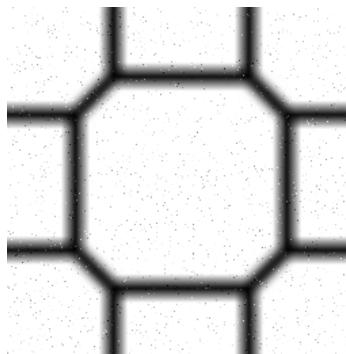
Bump mapping

le cnam

■ Perturbation aléatoire



■ Utilisation d'une texture de bump



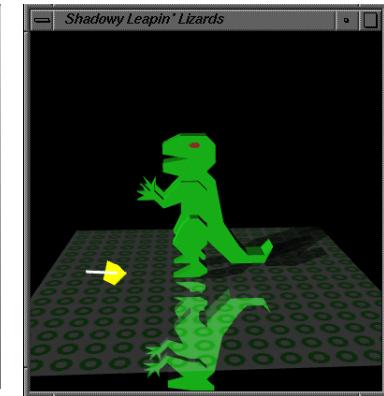
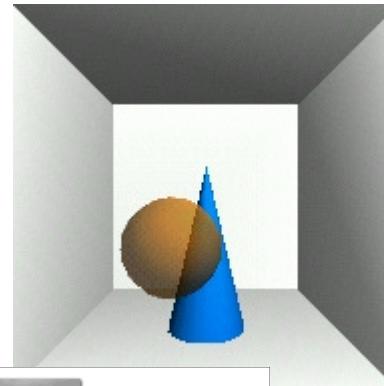
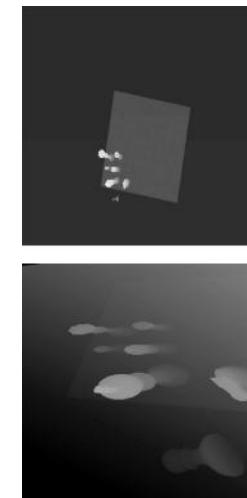
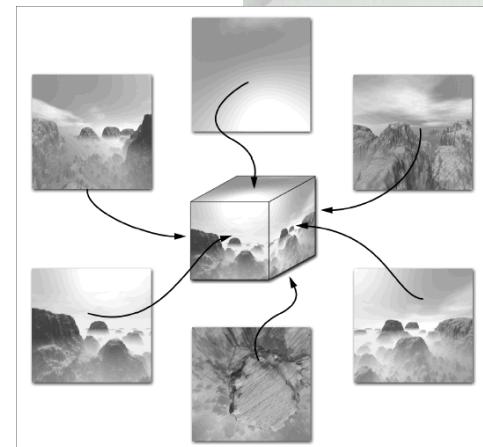
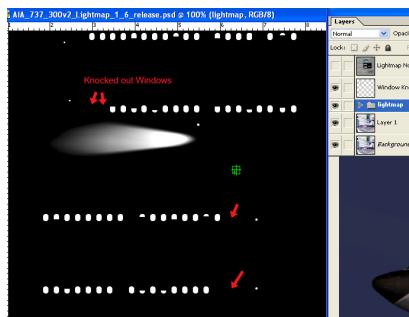
From Computer Desktop Encyclopedia
Reproduced with permission.
© 2001 Intergraph Computer Systems



Autres techniques de coloriage employant les textures

le cnam

- Par rendu multi passes :
 - La transparence
 - La réflexion sur un sol
 - Les ombres dynamiques
- Par multitexturage :
 - Lumières fixes
 - Ombres fixes
 - Environment mapping



Et pour aller plus loin ...

le cnam

- Les shaders !!



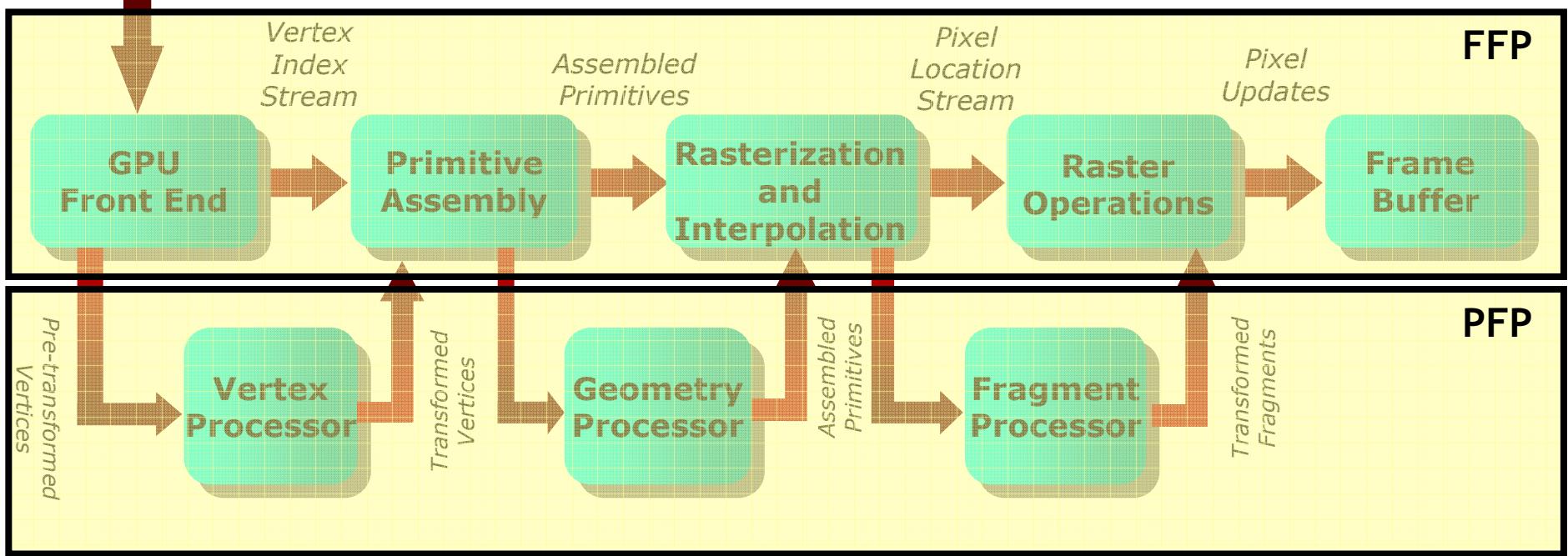
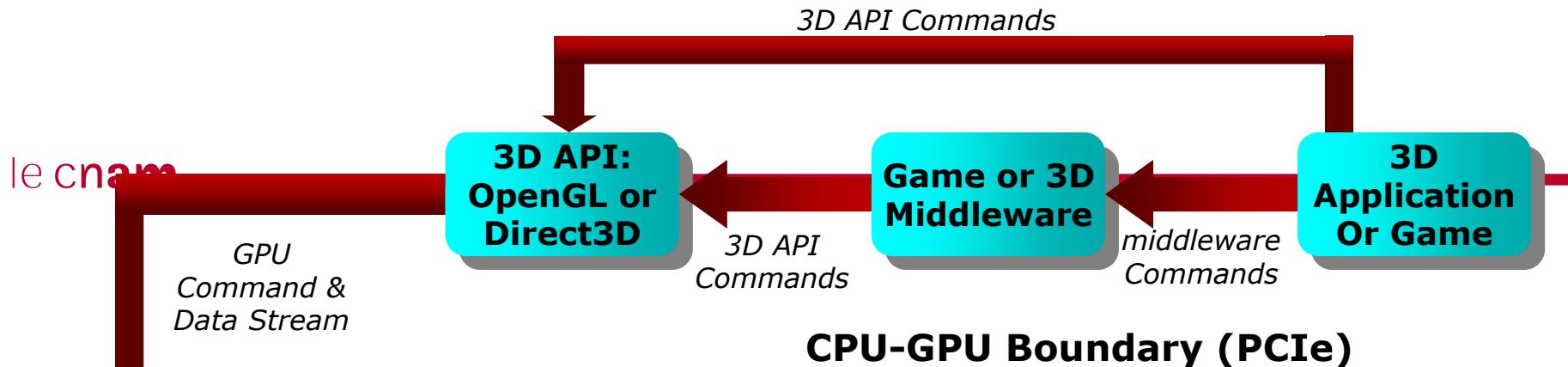
Doom III

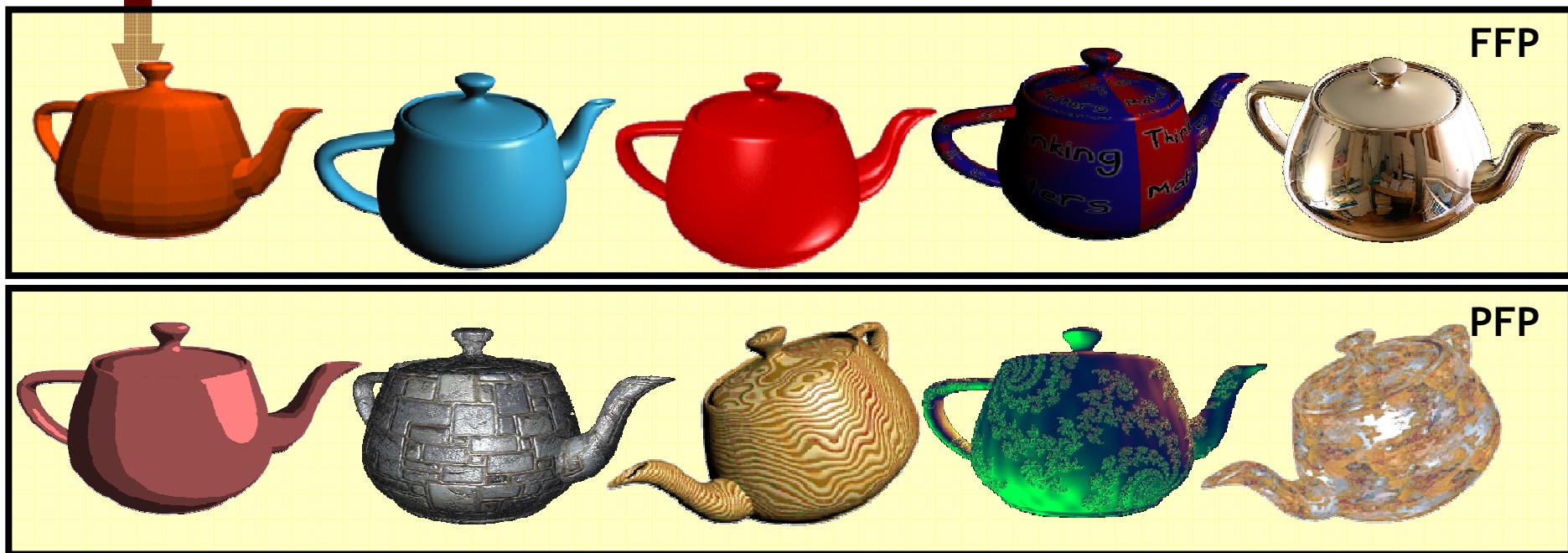
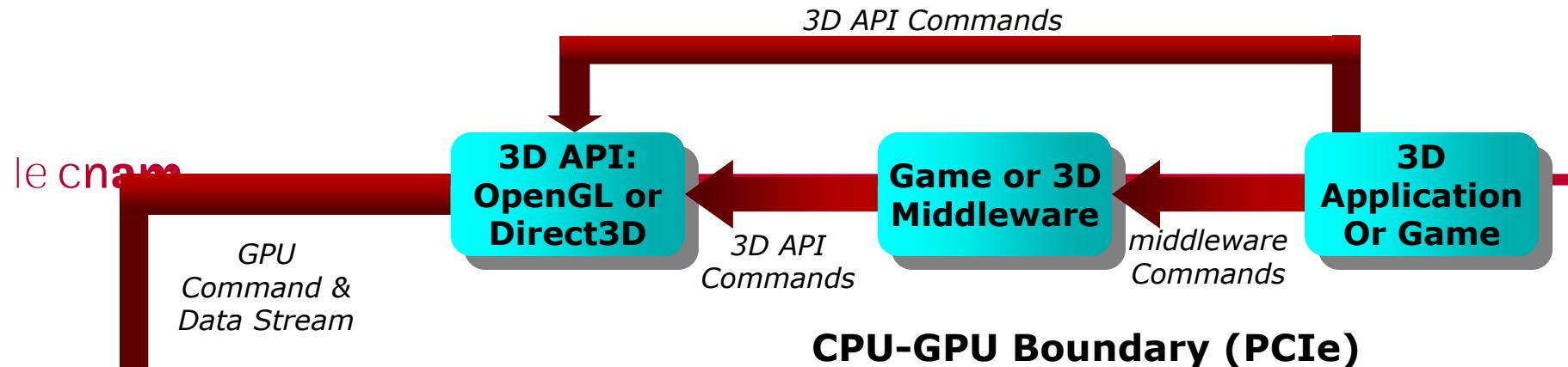


Halo 2



Jet Set Radio Future





Les animations

le cnam

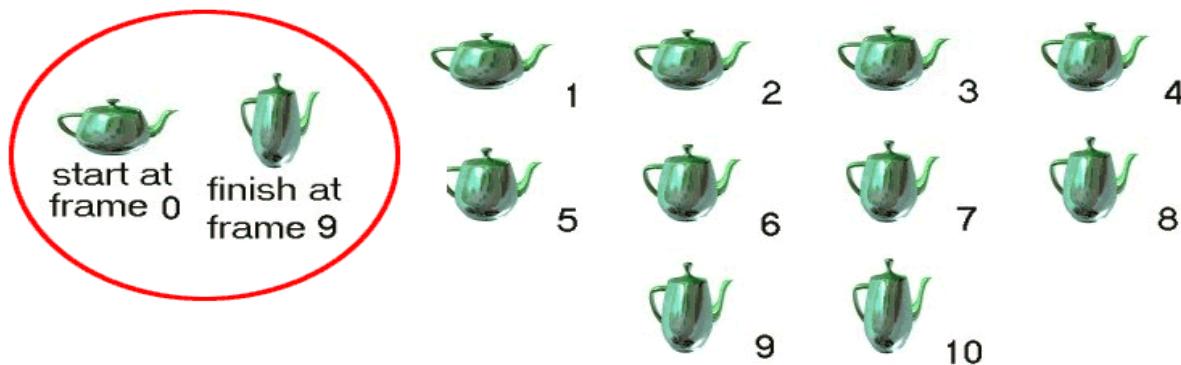
■ Plusieurs techniques :

1. Keyframing > animation par calcul d'image clé
2. Bones and Skin
3. Capture de mouvement (motion capture)
4. Animation procédurale

Animation par keyframing

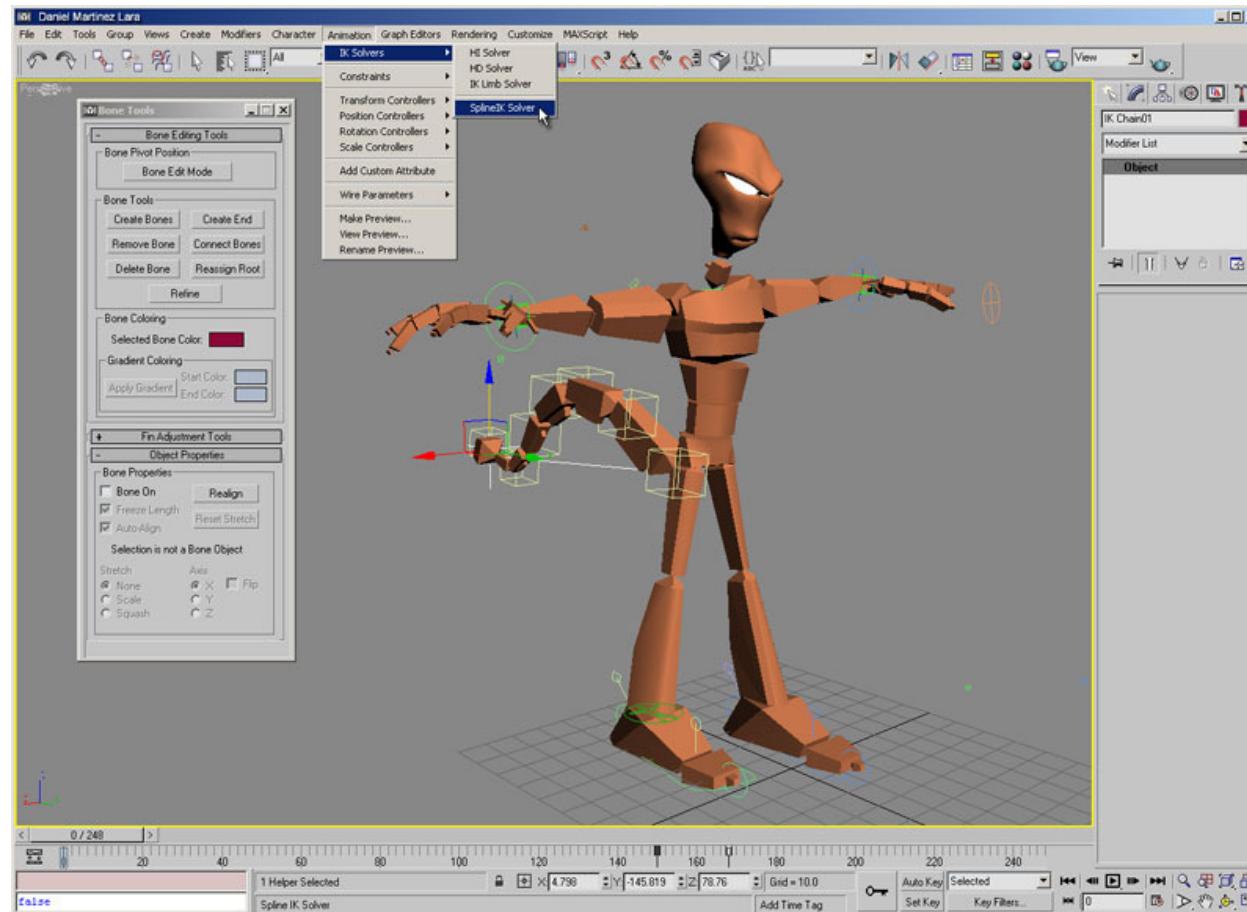
le cnam

- Procédé traditionnel développé par Walt Disney (images clés dessinées par les artistes principaux et les transitions par des « petites mains »).
- Sur ordinateur, le programme se charge de calculer les images intermédiaires par interpolation.
- Ex: le *morphing* est une interpolation de forme



Animation par bones

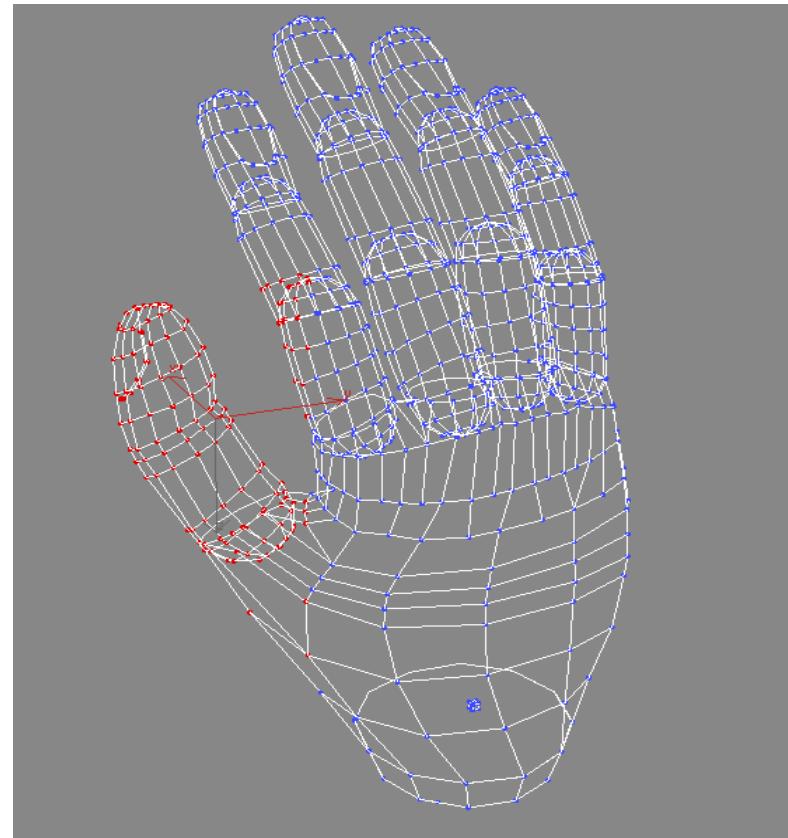
le cnam



Ancienne manière

le cnam

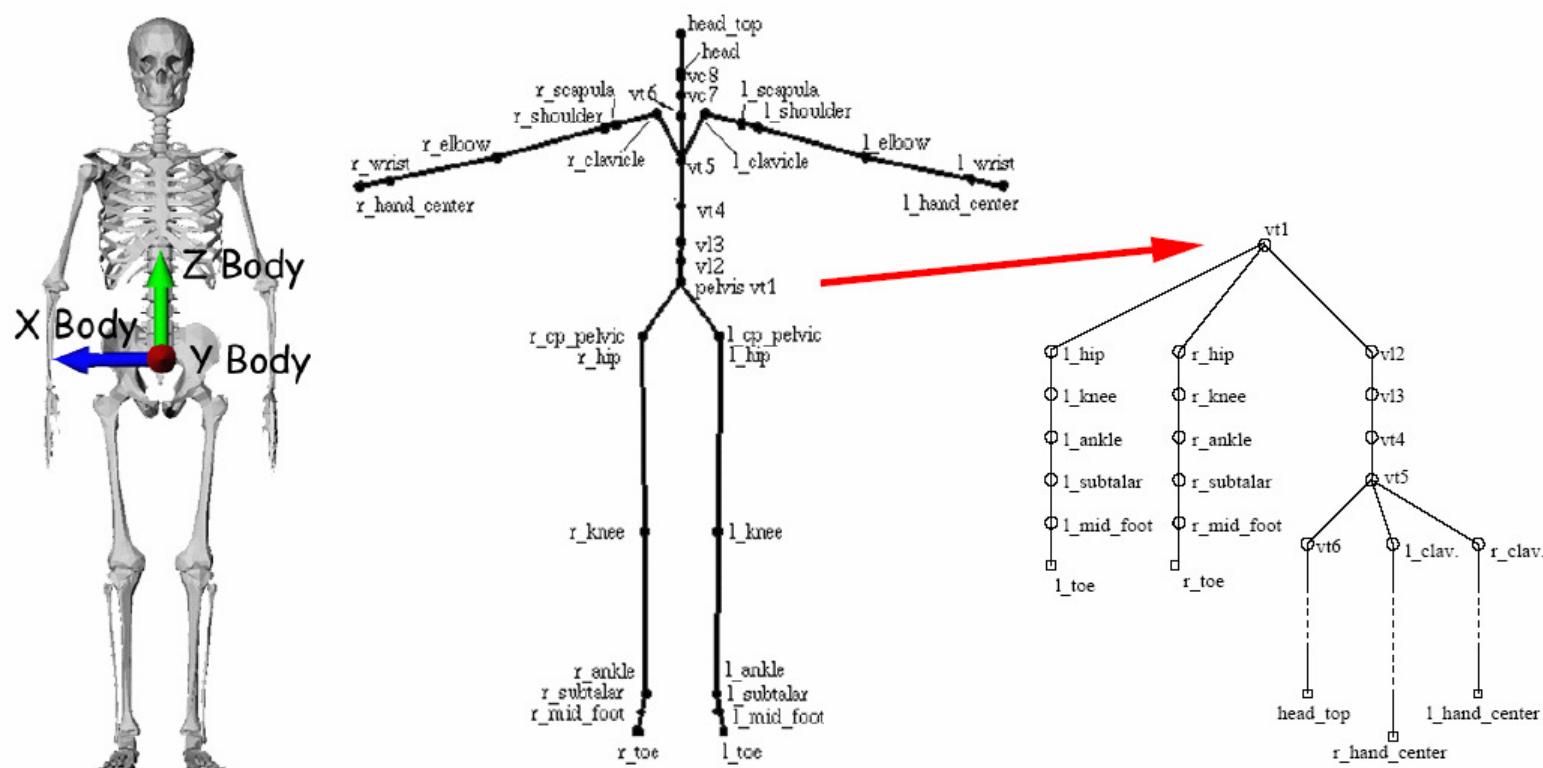
- Animer un maillage complet.
- Problème : animer chaque point ou groupe de points manuellement (...)



Modélisation

le cnam

- Modèle HAnim (EPFL)

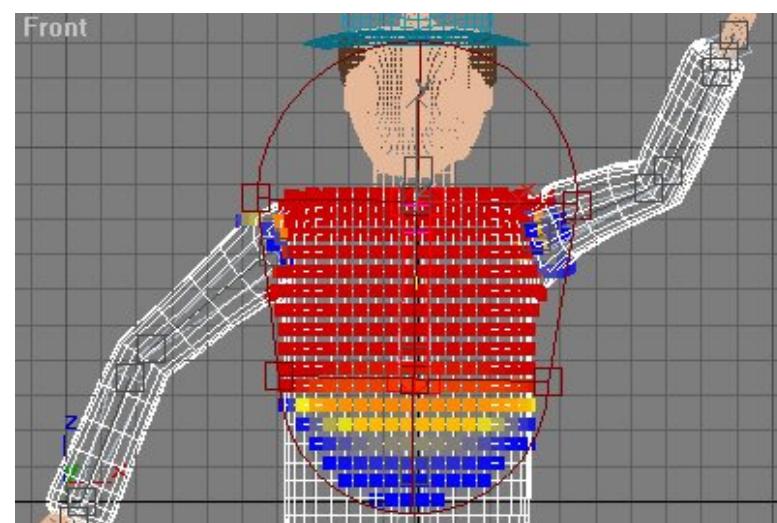
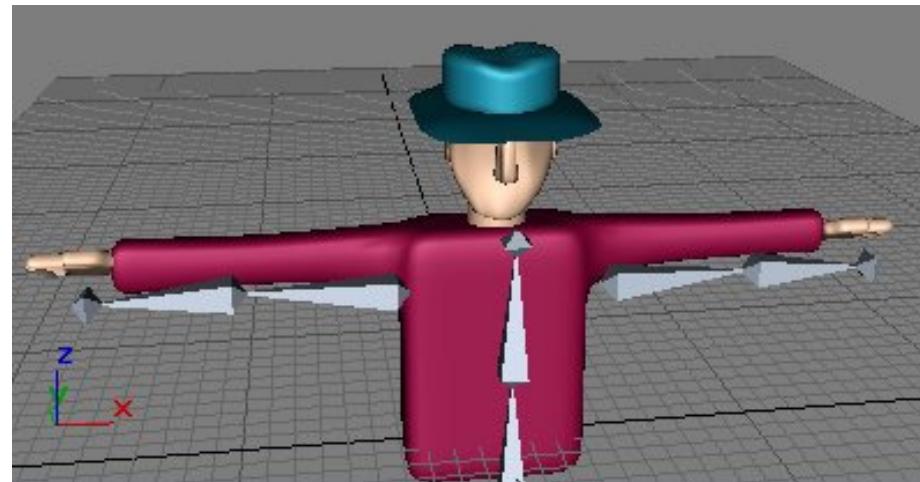
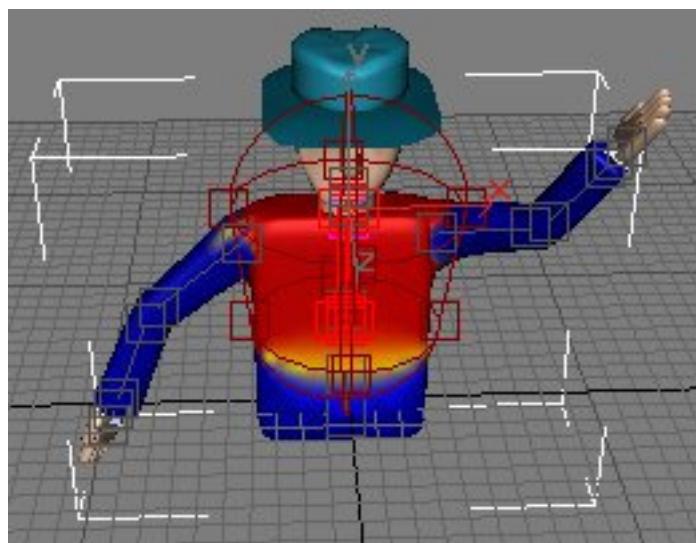


C. Babski, D. Thalmann, *Define Virtual Humans On The Web*, 2000.

Modélisation

le cnam

- Seconde étape :
- A chaque structure est associée un groupe de vertex qui seront affectés par le déplacement de cette structure.

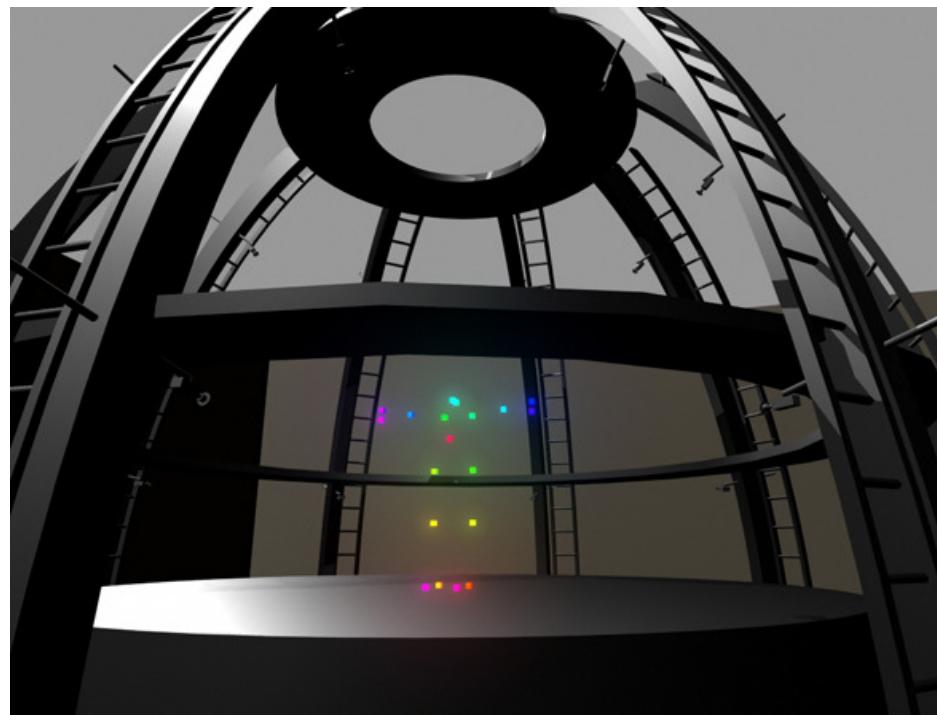


Animation – Mo-cap

le cnam

Soit temps réel

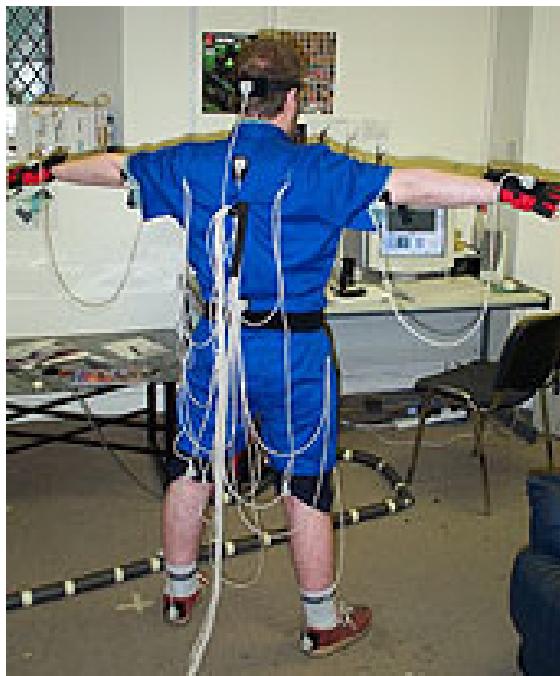
Soit récupérée puis appliquée à une structure



Animation – Mo-cap

lecnam

Motion capture



Animation procédurale – Moteur Physique

le cnam

- Utilisation des lois de la physique pour affiner l'animation ou la gérer complètement (rag dolls)
- Notions de forces motrice, de frottement, d'inertie, etc...
- Moteur physique (PhysX – cf cours PC)

•



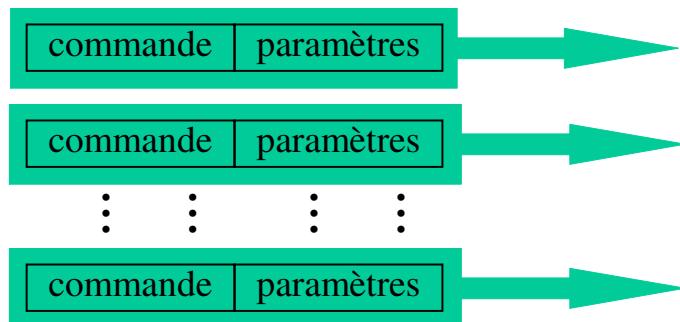
API 3D

le cnam

■ Deux modes de transmission des données à l'API

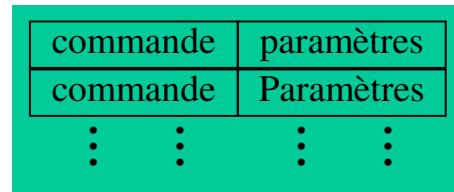
Mode immédiat

- Transmission **immédiate** des ordres à l'API par appels de fonctions

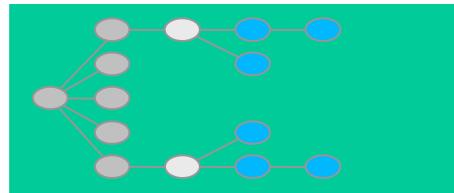


Mode retenu

- Utilisation d'une structure **retenant** les ordres 3D **puis** transmission de cette structure à l'API
- Types de structures :
 - buffer d'exécution, liste d'appels



- graphe de scène



API 3D

OpenGL/Mesa

le cnam

- Librairie graphique 2D/3D multi-plateforme
- Librairies d'outils pour deux aspects :
 - La gestion des événements et des fenêtres :
 - GLX permet de relier une application OpenGL avec Xwindow
 - GLUT gère (par procédures de type callback) les événements et le rendu dans une fenêtre
 - Des primitives graphiques 3D de plus haut niveau :
 - Dans GLUT : sphères, cubes, cylindres, cônes ...
 - Dans GLU : tesselation, splines, ...
 - Dans OpenInventor : structuration sous forme de graphe de scène

La 3D programmée

OpenGL/Mesa

le cnam

■ Un squelette d'application en C (mode immédiat)

```

#include <GL/gl.h>                                /* Déclaration des fonctions d'OpenGL */
#include <GL/glut.h>                                /* Déclaration des fonctions de GLUT */

void init(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);             /* La couleur d'effacement */
    glShadeModel(GL_FLAT);                         /* Le style de rendu */
}

void display(void) {                                /* Fonction dessinant chaque trame */
    /* effacement du frame-buffer et du Z-buffer */
    glClear(GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER);
    glColor3f(1.0, 1.0, 1.0);                      /* La couleur pour dessiner */
    /* Initialisation de la matrice de transformation */
    glLoadIdentity();
    /* Transformation des objets par rapport au point de vue */
    glTranslatef(0.0, 0.0, 3);
    /* dessin des objets */
    glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, 0.0);
    glVertex3f(0.75, 0.25, 0.0);
    glVertex3f(0.75, 0.75, 0.0);
    glVertex3f(0.25, 0.75, 0.0);
    glEnd();
    glutSolidSphere(1.0, 16, 16);
    glutSwapBuffers();                             /* Affichage de l'image calculée */
}

```

La 3D programmée

OpenGL/Mesa

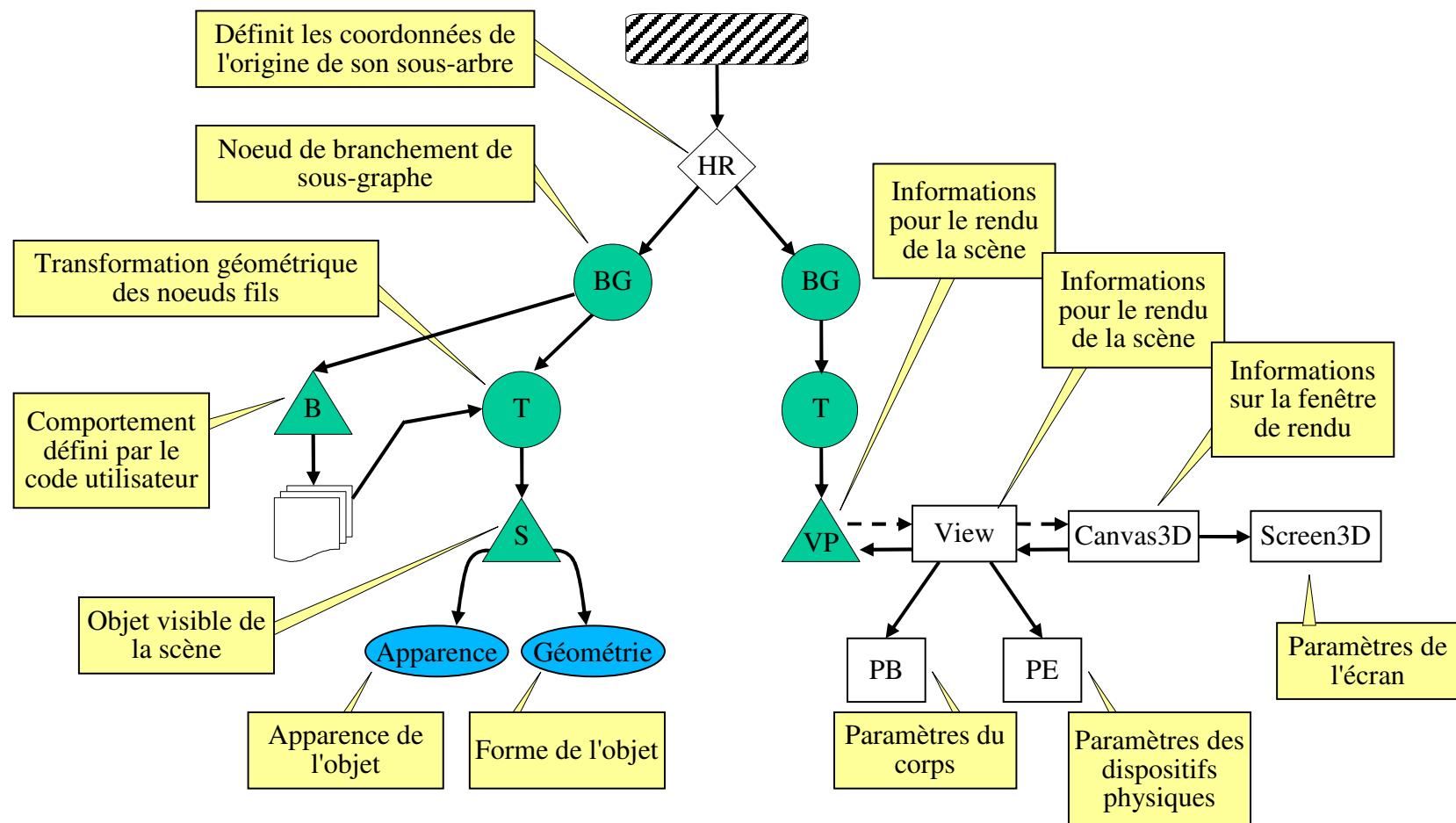
le cnam

- Un squelette d'application en C (mode retenu)
- Utilisation d'une liste de commandes (*display list*)
 - Initialisation
 - Début de remplissage de la liste par *glNewList*
`glNewList (monObjet, GL_COMPILE);`
 - Appel des commandes OpenGL à intégrer dans la liste
 - Fin de remplissage de la liste par appel à *glEndList()*
 - Utilisation lors du rendu (dans *display*)
 - Appel à *glCallList()* pour exécuter les commandes stockées
`glCallList (monObjet);`

La 3D programmée

Java 3D

le cnam



La 3D programmée

Direct 3D

le cnam



- Très proche d'OpenGL dans les principes
 - C et C++
 - Modes retenu et immédiat
 - Accélérations graphiques matérielles
- Direct3D ∈ DirectX
- Très réactive et très utilisée pour les jeux
- Flexible vertex format
- Managed DirectX depuis d'autres langages de programmation

La 3D programmée Performer

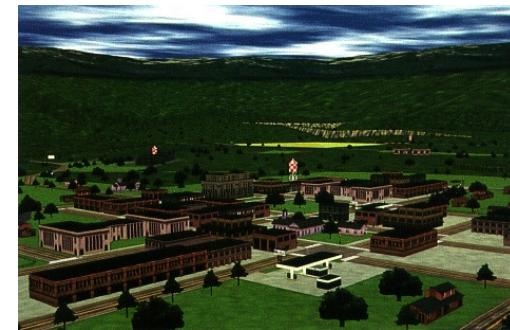
le cnam



IRIS PERFORMER

- Bibliothèque moyen niveau professionnel sur SGI/Linux
- Gère le parallélisme (plusieurs pipelines)
- Gestion de scènes complexes
- Charge de nombreux formats de fichiers 3D
- Graphe de scène
- Crée et gère ses fenêtres

www.sgi.com/software/performer/



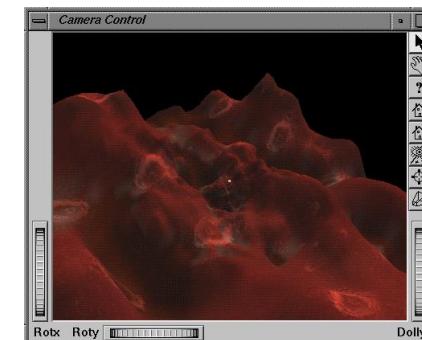
La 3D programmée

Open Inventor

le cnam

- Surcouche moyen niveau de OpenGL
- SGI, portage Windows payant
- Graphe de scène
- Visualiseur avec sélection, extensible
- Lié au format wrl (VRML 1 & 2)
- Pas très efficace

oss.sgi.com/projects/inventor/



Quelle API choisir ?

le cnam

- Visualiser un objet 3D : difficile (cf formats)
 - Maya, 3D StudioMax,
 - perfly (Performer), ivview (Inventor), Java3D
- Petite application graphique non critique
 - OpenGL
 - Java3D (bibliothèque de classes réutilisables)
- Application graphique performante
 - OpenGL, DirectX

Structure d'un programme

le cnam

■ La structure d'un programme OpenGL

■ Initialisation

- Demande de création d'un contexte GL
Double buffer, stéréo, transparence, ...

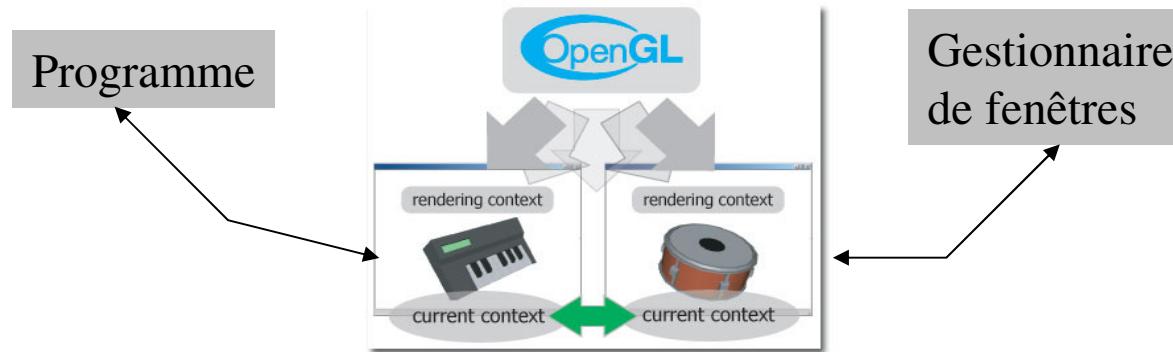
■ Boucle principale

- Gestion des interactions
- Mise à jour des données – animations, interactions
- Calcul de l'image (*render*)
- Affichage de la nouvelle image (*swap*)

■ Le code dépend du système utilisé (X/Windows, Win32, GLUT, VR Juggler)

Le contexte GL

le cnam

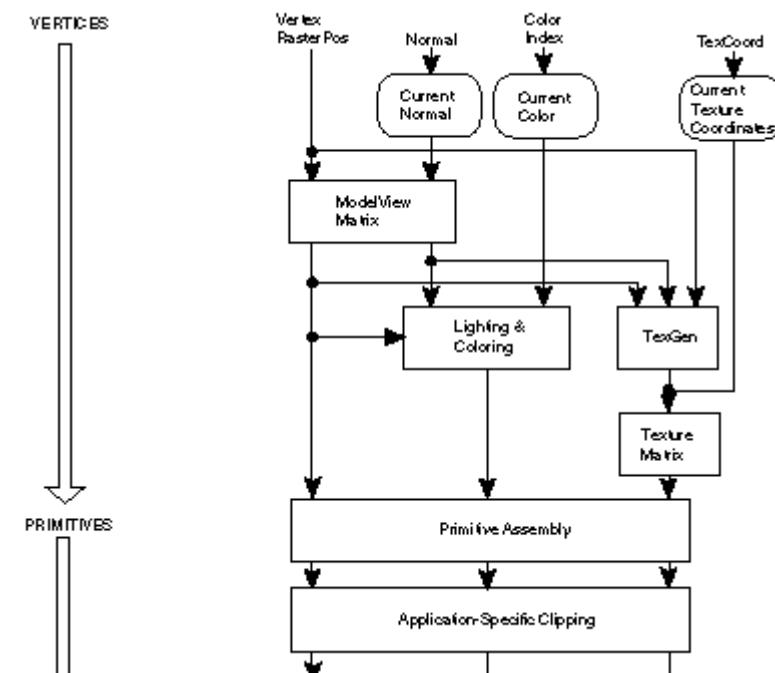


- Machine à états
 - Matrices de transformation, type d'affichage, couleurs, normale ...
- Buffers (color, depth, stencil, ...)
 - Stockant, pour chaque pixel, plus ou moins de bits
- Avec les bibliothèques utilitaires (GLUT, GLX, ...) il est géré de manière transparente et automatique

Procédure d'affichage principale

le cycle d'affichage

- Effacer l'écran
- Description de la caméra
- Pour chaque objet de la scène
 - Placement de l'objet
 - Modification de la machine à état
 - Ordres d'affichage



Effacer l'écran

le **clear** —

■ Couleur de fond – à l'initialisation

- `glClearColor(r, g, b, a);`

■ Effacement – à chaque trame

- `glClear(flags);`
- `flag = GL_COLOR_BUFFER_BIT`
et/ou `GL_DEPTH_BUFFER_BIT`
et/ou `GL_ACCUM_BUFFER_BIT`
et/ou `GL_STENCIL_BUFFER_BIT`
- `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`

Transformations

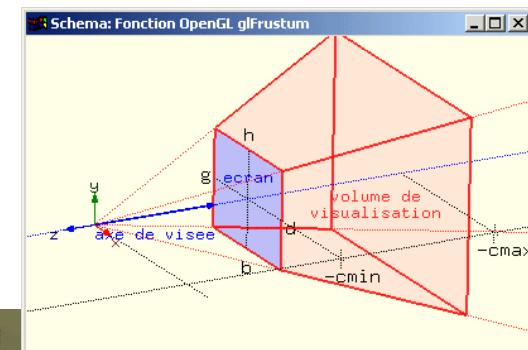
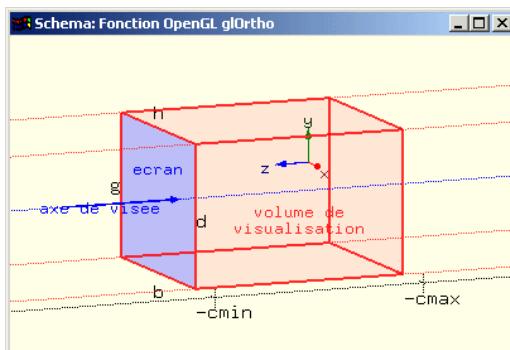
le cnam

- Chaque vertex subit des transformations avant d'être affiché
 - Position et orientation de l'objet dans la scène
 - Inverse de la position et l'orientation de la caméra
 - Projection 3D → 2D
- Les 2 premières transformations sont stockées dans la matrice *MODELVIEW*
- La projection est stockée dans la matrice *PROJECTION*

Description de la caméra

le cnam

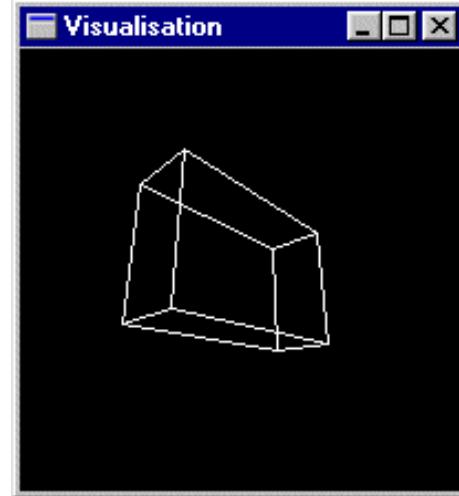
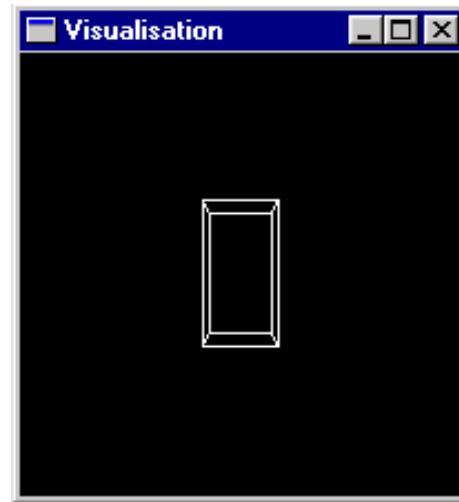
- `glMatrixMode(GL_PROJECTION);`
- `glLoadIdentity();`
- `glOrtho(...); ou glFrustum(...);`
- Ou `gluPerspective(); et gluLookAt(...);`



Positionnement des objets

le cnam

- `glMatrixMode(GL_MODELVIEW) ;`
- `glLoadIdentity() ;`
- `glTranslatef(x, y, z) ;`
- `glRotatef(alpha, x, y, z) ;`
- `glScalef(sx, sy, sz) ;`
- OU `glMultMatrixf(m) ;`
- `glBegin(...);`
- `...`
- `glEnd() ;`



Translation, Rotation, Scaling

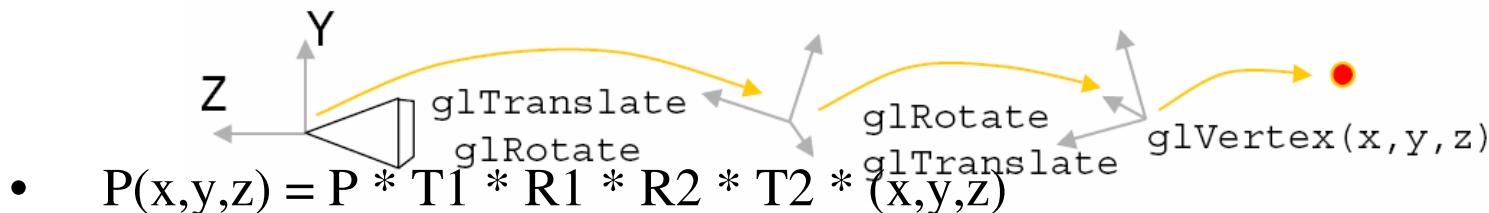
le cnam

- `glTranslate[d/f](x, y, z)` déplace l'objet du vecteur spécifié
 - Ex: `glTranslatef(0.0f, 0.0f, -6.0f);`
- `glRotate[d, f](angle, x, y, z)` tourne l'objet autour de l'axe spécifié (angle en degré)
 - Ex: `glRotatef(90.0f, 0.0f, 1.0f, 0.0f);`
- `glScale[d/f](x, y, z)` étire l'objet selon les facteurs spécifié pour chacun des axes
 - Ex: `glScalef(2.0f, 2.0f, 2.0f);`

Pile de matrices

le cnam

■ Structure de repères hiérarchiques



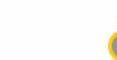
■ Pour chaque `glMatrixMode`, on a une pile

- `glPushMatrix();`
- `// Modification et utilisation de la matrice`
- `glPopMatrix();`

■ Très utile pour le dessin de structures hiérarchiques

Primitives graphiques : glBegin(...)

le cnam



GL_POINTS



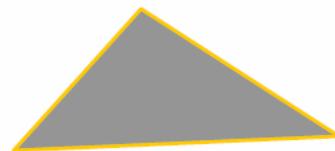
GL_LINES



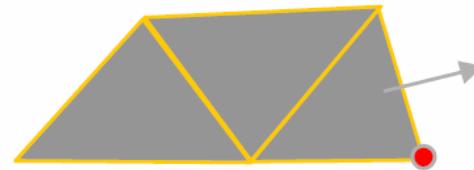
GL_LINE_STRIP



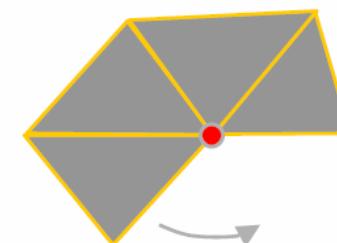
GL_LINE_LOOP



GL_TRIANGLES



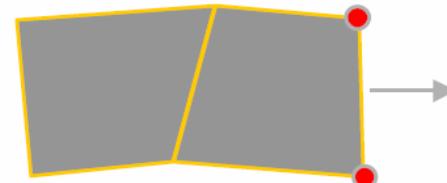
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP

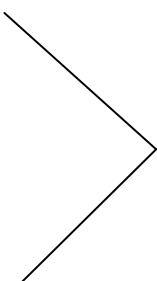


GL_POLYGON

Affichage de primitives

le **cnam**

- **glBegin**(*primitive*) ;
- // *primitive* = GL_POINTS, GL_LINES, GL_TRIANGLES...
- glColor3f(*r*, *g*, *b*) ;
- glNormal3f(*nx*, *ny*, *nz*) ;
- glTexCoord(*u*, *v*) ;
- glVertex3f(*x*, *y*, *z*) ;
- ...
- **glEnd**() ;



Répéter *n* fois
(*n* selon la primitive)

- Seul le **glVertex** est obligatoire,
- sinon la dernière valeur spécifiée est utilisée

Exemple

Afficher un triangle

le cnam

```
#include <GL/gl.h>
void render() {
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f,0.0f,0.0f);          // couleur V1
        glVertex3f( 0.0f, 1.0f,-3.0f); // position V1
        glColor3f(0.0f,1.0f,0.0f);          // couleur V2
        glVertex3f(-1.0f,-1.0f,-3.0f); // position V2
        glColor3f(0.0f,0.0f,1.0f);          // couleur V3
        glVertex3f( 1.0f,-1.0f,-3.0f); // position V3
    glEnd(); // GL_TRIANGLES
}
```

Exemple

Programme principal avec GLUT (1)

```
#include <GL/glut.h>
void init();
void display();
void idle();
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    glutMainLoop();
}
```

Exemple

Programme principal avec GLUT (2)

```
void init() { // initialise la caméra
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1.3333, 1.0, 5000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void display() { // calcule et affiche une image
    render();
    glutSwapBuffers();
}

void idle() { // reaffiche une fois terminé
    glutPostRedisplay();
}
```

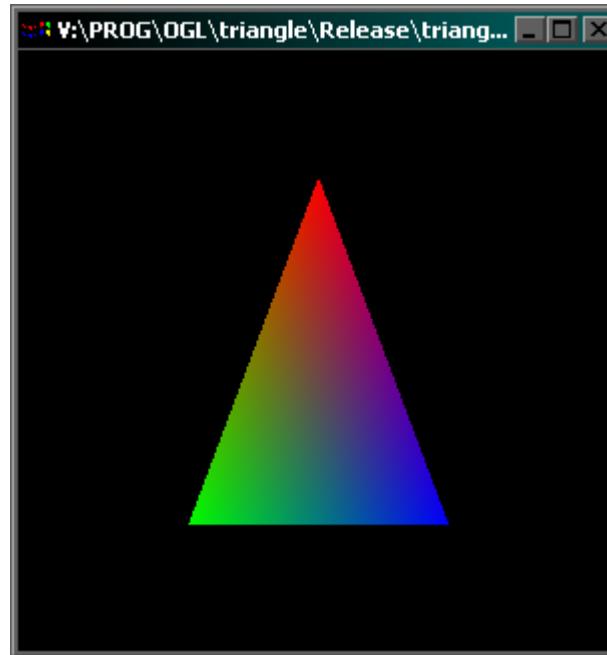
Exemple

Compilation avec GLUT

■ Compilation:

- `gcc -o triangle main_glut.c render_triangle.c -lglut -lGLU -lGL -lXmu -lXext -lX11 -lm`

■ Résultat:

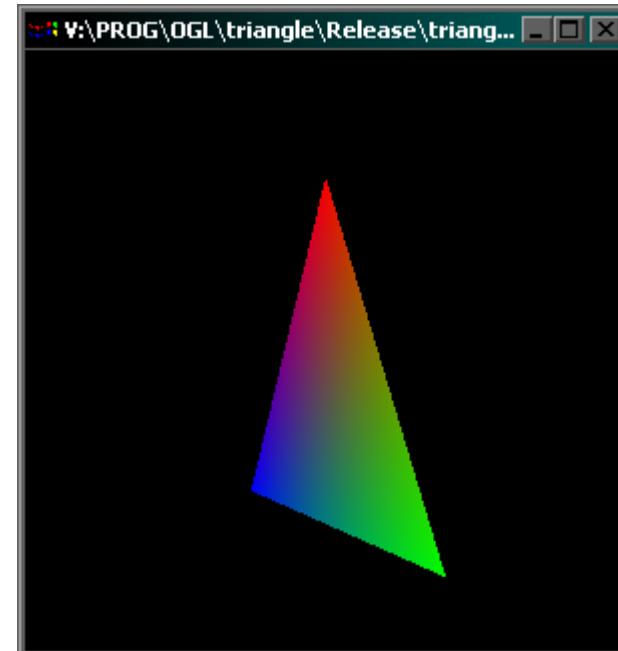


Exemple

Ajout de transformations

le cnam

```
// render_rotate.c
#include <GL/gl.h>
int nbframe=0;
void render() {
    glClear(...); // comme Exemple 1
    glPushMatrix();
    glTranslatef(0, 0, -3);
    glRotatef(nbframe, 0, 1, 0);
    glBegin(GL_TRIANGLES);
        ... // comme Exemple 1
    glEnd(); // GL_TRIANGLES
    glPopMatrix();
    glBegin(...)
    ...
    glEnd();
    ++nbframe;
}
```



Options d'affichage

le cnam

```
glEnable(GL_LIGHTING);
```

Prise en compte de la normale

```
glCullFace(face);
```

```
glEnable(GL_CULL_FACE);
```

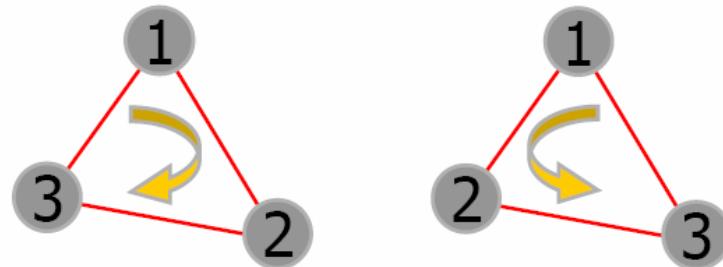
```
face = GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
```

```
glPolygonMode(face, mode);
```

mode = GL_FILL, GL_LINE

```
glLineWidth(3);
```

```
glPointSize(12);
```

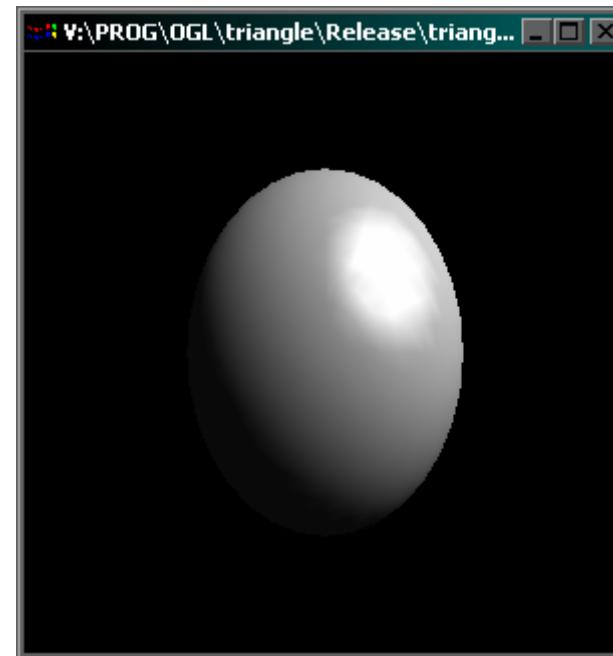


Matériaux et Lumières

Exemple

le cnam

```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat mat_shininess[] = { 50.0 };  
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };  
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);  
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);  
glLightfv(GL_LIGHT0, GL_POSITION, light_position);  
 glEnable(GL_LIGHTING);  
 glEnable(GL_LIGHT0);
```



Et plein d'autres choses dans ...

le cnam

■ OpenGL Programming Guide (The Red Book)

- http://ask.ii.uib.no/ebt-bin/nph-dweb/dynaweb/SGI_Developer/OpenGL_PG/

■ OpenGL Reference Manual (The Blue Book)

- http://ask.ii.uib.no/ebt-bin/nph-dweb/dynaweb/SGI_Developer/OpenGL_RM/

■ <http://www.opengl.org> : site officiel. Forums

■ <http://nehe.gamedev.net> : le site de tutoriels OpenGL

■ Une mine d'infos en français : le site de Nicolas Janey : <http://raphaello.univ-fcomte.fr/Ig/>