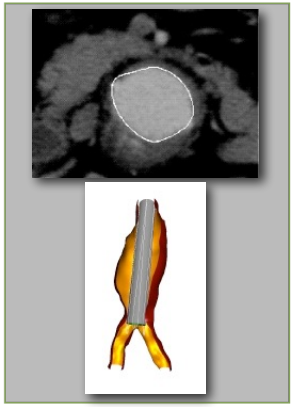
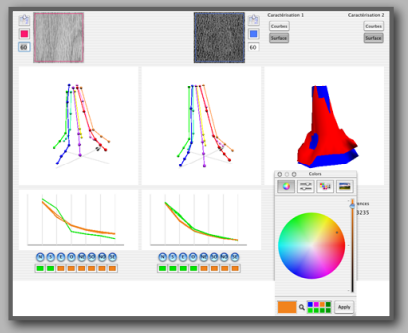
OpenGL

Techniques de bases

Domaines d'application

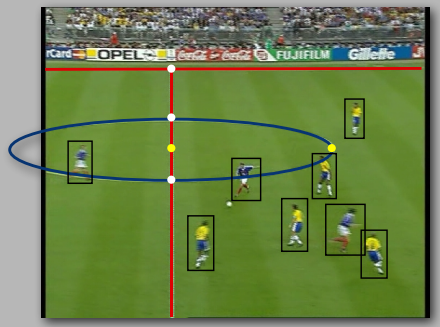
- Synthèse d'image
- Analyse d'image
- Modélisation géométrique
- Visualisation scientifique
- ...



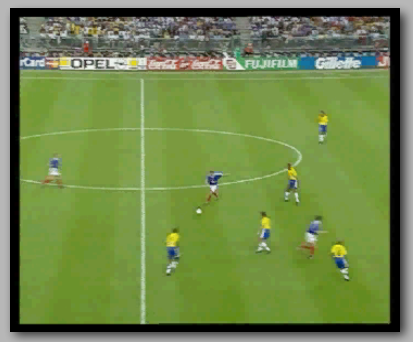


Visualisation Scientifique

Analyse d'Images



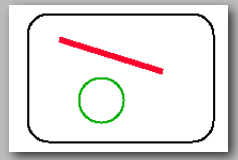
Analyse d'Images

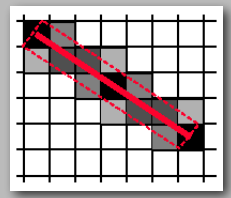


Synthèse d'Images

L'affichage : une brève introduction

- Modèle trait
  - Une image est décomposée en un ensemble de traits d'une certaine épaisseur et couleur
  - La taille du modèle dépend de la complexité de l'image
- Modèle pixel
  - Une image est décomposée en une matrice (finie) de points (pixels)
  - La taille du modèle (la résolution) est indépendante de la complexité de l'image



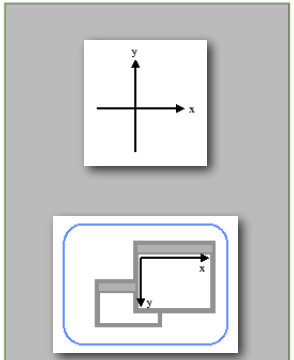


L'affichage : Le "Frame Buffer"

- Propre au modèle pixel
- Espace mémoire dont le contenu est une représentation logique de ce qui apparaît à l'écran
  - Deux dimensions définissent la résolution spatiale de l'image et forment un plan
  - Une troisième dimension appelée "profondeur" spécifie le nombre de bits utilisés pour représenter un pixel
- Intermédiaire entre l'écran et la bibliothèque graphique

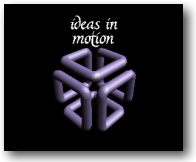
L'affichage : Espace virtuel & Espace d'affichage

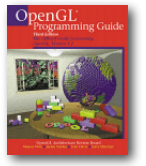
- Espace virtuel
  - Dimension arbitraire
  - Infini
  - Continu
- Espace d'affichage
  - Dimension 2
  - Fini (limité par la résolution)
  - Discret (modèle pixel)



OpenGL : Qu'est-ce que c'est ?

- Bibliothèque de fonctions graphiques
- Un standard
- Portable
- Animation
- Utilisation du hardware disponible
- opengl.org
- mavromatis.org





OpenGL : Quelques fonctionnalités

- Tracé de primitives géométriques (points, ligne, polygone ...)
- Organisation d'objets dans une scène
- Définition un point de vue
- Définition de l'éclairage
- Effets de profondeur (brouillard ...)
- Lissage
- Rendu temps réel

OpenGL & GLUT

- Indépendant du système d'exploitation
  - Ne gère pas le fenêtrage
  - Ne gère pas les interruptions
- open**GL** **U**tility **T**oolkit
- **GLFW**, SFML …

GLUT

Un squelette de programme
La création de fenêtre
Les événements
Les menus
L'animation

GLUT

- Un système de fenêtrage simple et portable
- Basée sur la "programmation événementielle"
  - La structure de l'application est imposée
  - L'application n'est contrôlée que par des interruptions
- Fonctionnalités :
  - Création et gestion de plusieurs fenêtres graphiques
  - Gestion des interruptions (clic souris, touches clavier ...)
  - Création et gestion de menus déroulants
  - Indicateur d'état de certains paramètres systèmes
  - Quelques fonctions supplémentaires : objets 3D

GLUT : Un programme type

- Structure épurée d'une application GLUT

```
#include <GL/glut.h>

// Définition de fonctions ...
// ... de cadrage
// ... d'affichage
// ... de traitement des interruptions

int main(int argc, char **argv) {
// Initialisation de la session graphique
   // Désignation des fonctions ...
   // ... de cadrage, ... d'affichage, ... de traitement des interruptions
   // Création d'une fenêtre (et implicitement d'un CONTEXTE GRAPHIQUE)
// Lancement de la boucle principale
}
```

GLUT : Un programme type

```
int main(int argcp, char **argv)
{
    /* Initialisation de GLUT */
    glutInit(&argcp, argv);

    /* Taille et emplacement de la fenêtre */
    glutInitWindowSize(400, 400);
    glutInitWindowPosition(200, 100);
    /* Choix du type et d'affichage RGBA (mode couleur le plus fréquent), tampon de profondeur
       et d'un double buffer */
    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
    /* Création de la fenêtre */
    glutCreateWindow("GLUTfenetre");

    /* Association des callback pour cette fenêtre */
    glutDisplayFunc(affichage);
    glutReshapeFunc(redim);

    glutMainLoop(); /* On entre dans la boucle d'événements */

    return 0;
}
```

GLUT : Un programme type

```c
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>

/* la fonction "affichage" est appelée a chaque fois qu'un événement de mise
à jour de l'affichage est détecté */
void affichage(void)
{
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      /* Oups ! une instruction OpenGL : on efface le buffer d'écran */
    glutSwapBuffers();              /* Oups ! encore une : on indique ici qu'il faut afficher */
}
/* la fonction "redim" est appelée : une fois a la creation de la fenêtre ;
   ensuite à chaque fois que la fenêtre est redimmensionnée
   width et height representent la taille de la fenêtre */
void redim(int width, int height)
{
    glViewport(0, 0, width, height); /* Oups ! on reviendra sur la notion de viewport plus tard,
    on va simplement retenir ici que cette instruction veut dire :
    on affiche dans toute la fenêtre */
}
```

GLUT : La compilation (linux)

- Le fichier “Makefile”

```
XLIBS = -L/usr/X11/lib -L/usr/X11R6/lib -lX11 -lXext -lXi -lXmu
GLLIBS = -L/usr/X11R6/lib/ -lglut -lGL -lGLU

GLUTFenetre :
GLUTFenetre.o
        g++ -o GLUTFenetre GLUTFenetre.o  $(GLLIBS) $(XLIBS) -lm

GLUTFenetre.o : GLUTFenetre.cxx
        g++ -c GLUTFenetre.cxx

clean:
        -rm -f *.o
```

GLUT : Souris, clavier et Cie

```c
int main(int argcp, char **argv) {
    glutInit(&argcp, argv);
    glutInitWindowSize(400, 400);
    glutInitWindowPosition(400, 50);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutCreateWindow("GLUTEvenements");
    glutDisplayFunc(affichage);
    glutReshapeFunc(redim);

    /* la fonction "clavier" est appelee quand un evenement clavier est detecte */
    glutKeyboardFunc(clavier);
    /* la fonction "souris" est appelee quand un evenement souris est detecte */
    glutMouseFunc(souris);

    glutMainLoop();
    return 0;
}
```

GLUT : La souris

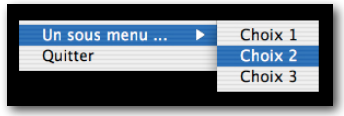
```c
/* la fonction "souris" est appelee quand un evenement souris est detecte

button contient l'identificateur du bouton enfonce :
GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON ou GLUT_RIGHT_BUTTON

state contient l'etat de ce bouton :
GLUT_UP ou GLUT_DOWN
x et y sont les coordonnees de la souris au moment du clic (ou declic) */

void souris(int button, int state, int x, int y)
{
 switch(button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN) printf("Clic gauche en (%d, %d)\n", x, y);
            break;
                  }
 glutPostRedisplay(); //indique que le contenu de la fenetre doit etre affiche
}
```

GLUT : Le clavier

```
/* la fonction "clavier" est appelee quand un evenement clavier est detecte

key est le code ASCII de la touche enfoncé

x et y sont les coordonnees de la souris au moment de l'appui de la touche */

void clavier ( unsigned char key, int x, int y )
{
    switch ( key ) {
        case 27: /* ESC */
        case 'q':
        case 'Q': exit (0);
            break;
}}
```

GLUT : Créer des menus

- Un menu est associé à une fenêtre
  - La fonction de création du menu doit être appelée après celle de création de la fenêtre
- Un menu est attaché à un bouton de la souris



GLUT : Création des menus

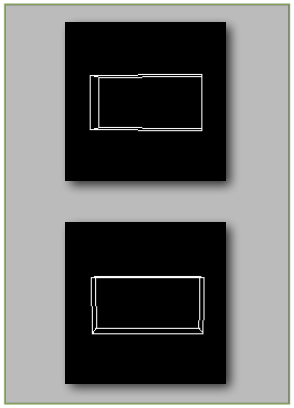
```
void CreerMenu(void) {
    int sousMenu;

    /* creation d'un menu dont la fonction de "gestion" est ChoixSousMenu l'identicateur de ce
    menu est stocké dans l'entier sousMenu afin de pouvoir lier ce menu au menu principal */
    sousMenu = glutCreateMenu(ChoixSousMenu);
    /* ajout d'une entree dans ce menu, les arguments sont : nom apparaissant dans le menu et
       identificateur le representant dans la fonction de gestion */
    glutAddMenuEntry("Choix 1",1);
    glutAddMenuEntry("Choix 2",2);
    glutAddMenuEntry("Choix 3",3);

    /* creation d'un menu dont la fonction de "gestion" est ChoixMenuPrincipal */
    glutCreateMenu(ChoixMenuPrincipal);
    /* ajout d'un sous menu a ce menu, les arguments sont : nom apparaissant dans le menu
       et identificateur de ce sous menu
       ATTENTION : le sous menu doit etre cree avant */
    glutAddSubMenu("Un sous menu ...",sousMenu);
    glutAddMenuEntry("Quitter",10);

    /* on associe l'apparition de ce menu au bouton droit de la souris */
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

GLUT : Gestion des menus

```
/* fonction de gestion du menuPrincipal
value contient l'identificateur associe à l'article du menu selectionne*/
void ChoixMenuPrincipal(int value) {
    switch (value) {
        case 10 : exit(0); /* on quitte */
            break;
} }
/* fonction de gestion du sousMenu */
void ChoixSousMenu(int value) {
    printf("Choix sous menu numero %d\n", value);
    /* l'instruction glClearColor permet de définir la couleur
    d'effacement de la fenetre (du buffer de couleurs) */
    switch (value) {
        case 1 : glClearColor(1., 0., 0., 1.); /* rouge */
            break;
        case 2 : glClearColor(0., 1., 0., 1.); /* vert */
            break;
        case 3 : glClearColor(0., 0., 1., 1.); /* bleu */
            break;
    }
    glutPostRedisplay();
}
```

GLUT : Un peu "d'animation"

- Quand rien ne se passe !
  - glut**Idle**Func(void (*f)(void))
- Les timers
  - glutTimerFunc(unsigned int msecs, void (*f) (int v), v)



OpenGL

Syntaxe
Machine d'états
Les primitives
Les piles de matrices
Le point de vue
La projection

OpenGL : Syntaxe et types

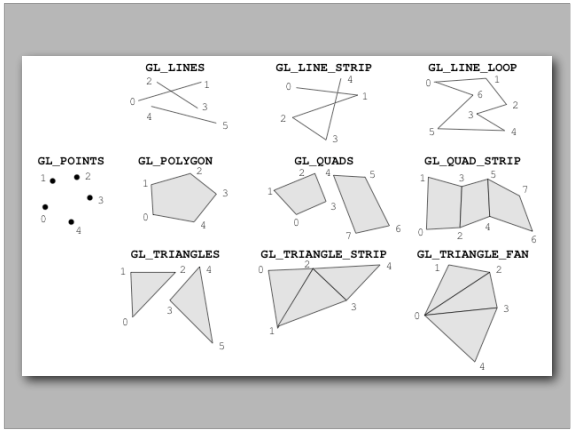
- Préfixe : **gl**, parfois **glu :** **gl**Color3f(1.0, 0.0, 0.0);
- Lettre majuscule à chaque début de mot : gl**C**lear**C**olor(1.0, 0.0, 0.0);
- Suffixe : glColor**3f**(1.0, 0.0, 0.0);
  - précise le nombre et le type des arguments
- Constantes en majuscules : **GL**_COLOR_BUFFER_BIT
- Redéfinition des types de base : **GLint**, **GLfloat**, ...

OpenGL : Une machine d'états

- Nombre de variables d'états immense
- Par exemple : mode de tracé, couleur, type d'éclairage ...
- Certains états booléens :
  - glEnable( ), glDisable( )
- Connaître la "valeur" d'un état courant :
  - glGetBooleanv( ) ... glGetIntegerv( ) ... glGetError( ) ...
- Pile d'états
  - glPushAttrib( ), glPopAttrib( )

OpenGL : Les primitives

- Les sommets sont le fondement des primitives géométriques
  - **glVertex** {2, 3, 4} {s, i, f, d} [v] (...)
- Les sommets se définissent dans un "bloc"
  - **glBegin** (...) : démarre la composition des sommets
  - **glEnd** ( ) : termine la composition des sommets
- Le mode de composition définit l'interprétation des sommets
  - glBegin (**GL_TRIANGLES**);

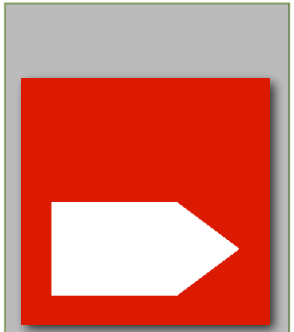


OpenGL : Modes de composition

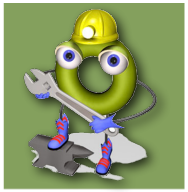
Les primitives : Un exemple

- Dessiner un polygone

```
void trace_objet(void)
{
   glBegin(GL_POLYGON);

      glVertex2f(0.0, 0.0);
      glVertex2f(0.0, 3.0);
      glVertex2f(4.0, 3.0);
      glVertex2f(6.0, 1.5);
      glVertex2f(4.0, 0.0);

   glEnd();
}
```



D E M

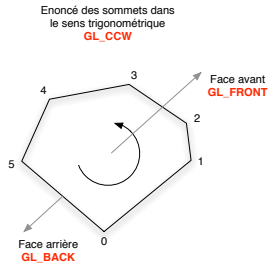


OpenGL : Suppression de surface

- Un polygone a deux faces: GL_FRONT et GL_BACK
- Définir l'orientation des polygones :
  - void glFrontFace(GLenum mode)

    mode : GL_CCW, GL_CW
- Activer la suppression
  - void glEnable(GL_CULL_FACE)
- Définir le mode de suppression
  - void glCullFace(GLenum mode)

    mode : GL_BACK, GL_FRONT, GL_FRONT_AND_BACK

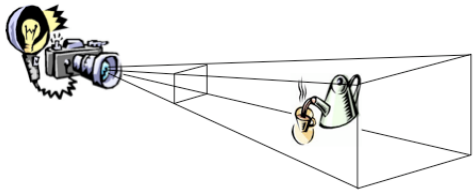


OpenGL : Mode de tracé des polygones

- Le mode de tracé est (ou peut-être) fonction du type de face
- Par défaut, les faces sont tracées en mode opaque (GL_FILL)
- void glPolygonMode(GLenum face, GLenum mode)
  - face : GL_BACK, GL_FRONT, GL_FRONT_AND_BACK
  - mode : GL_FILL, GL_LINE, GL_POINT

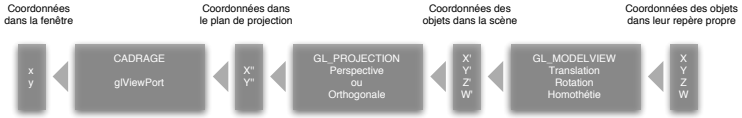
OpenGL : Composition d'image

- Arranger la scène — modèle
- Pointer l'appareil vers la scène — vue
- Régler l'objectif — projection
- Régler l'agrandissement et tirer la photo — viewport



OpenGL : Composition d'image

- Processus de composition



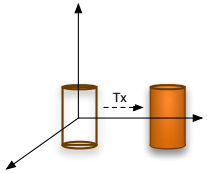
OpenGL : Par exemple

OpenGL : Transformations géométriques

- Translation

$$P\begin{pmatrix} x \\ y \end{pmatrix} \xrightarrow{T} P'\begin{pmatrix} x' \\ y' \end{pmatrix} \; avec\, T\begin{pmatrix} dx \\ dy \end{pmatrix}$$

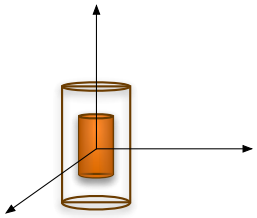
$$P' = P + T \; soit \begin{cases} x' = x + dx \\ y' = y + dy \end{cases}$$



OpenGL : Transformations géométriques

- Homothétie
  - Changement d'échelle par rapport à l'origine
  - Transformation définie par une matrice de la forme :

$$H\begin{pmatrix} h_x & 0 \\ 0 & h_y \end{pmatrix}$$



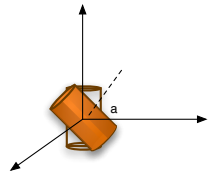
  - si $h_{\{x,y\}} < 1$, on rapproche les points de l'origine et l'objet devient plus petit
  - si $h_{\{x,y\}} > 1$, on éloigne les points de l'origine et l'objet devient plus grand
  - Si $h_x = h_y$: homothétie uniforme sinon homothétie différentielle

OpenGL : Transformations géométriques

- Rotation
  - Rotation d'un angle 'a' par rapport à l'origine

$$P' = RP \; avec\, R\begin{pmatrix} cos(a) & -sin(a) \\ sin(a) & cos(a) \end{pmatrix}$$

$$P'\begin{cases} x' = xcos(a) - ysin(a) \\ y' = xsin(a) + ycos(a) \end{cases}$$



OpenGL : Espace projectif

- Objectif : n'utiliser que des matrices
  - La composition de transformations est représentée par une seule matrice
- En 2D, l'espace projectif est un plan défini par t = 1
  - Si P(x, y, 1) est le point initial alors P'(x', y', A) sera le point final
  - Si par calcul on obtient P'(x', y', t') avec t' différent de 1, on se ramène à P'(x'/t', y'/t', 1)

$$T\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \quad H\begin{pmatrix} h_x & 0 & 0 \\ 0 & h_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad R\begin{pmatrix} cos(a) & -sin(a) & 0 \\ sin(a) & cos(a) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

OpenGL : Transformations géométriques

- En 3D, il suffit d'ajouter la coordonnée Z

$$T\begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad H\begin{pmatrix} h_x & 0 & 0 & 0 \\ 0 & h_y & 0 & 0 \\ 0 & 0 & h_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Soit R la matrice de rotation 2D : $R\begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix}$

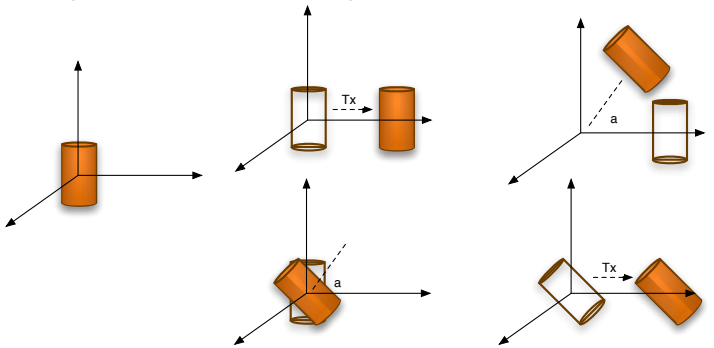
$$R_x\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & R_{11} & R_{12} & 0 \\ 0 & R_{21} & R_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_z\begin{pmatrix} R_{11} & R_{12} & 0 & 0 \\ R_{21} & R_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_z\begin{pmatrix} R_{11} & 0 & R_{12} & 0 \\ 0 & 1 & 0 & 0 \\ R_{21} & 0 & R_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL : Les instructions de transformations

- Les fonctions suivantes définissent des matrices qui multiplient (à droite) une matrice "courante"
  - glTranslate{f, d} (TYPE x, TYPE y, TYPE z)
    applique une translation de *x, y, z* sur les trois axes
  - glRotate{f, d} (TYPE a, TYPE x, TYPE y, TYPE z)
    applique une rotation de *a* degrés autour de l'axe *x, y, z*
  - glScale{f, d} (TYPE x, TYPE y, TYPE z)
    applique une homothétie de *x, y, z* sur les trois axes

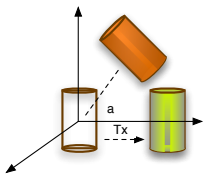
OpenGL : Composition des transformations

- Le produit de matrices n'est pas commutatif !

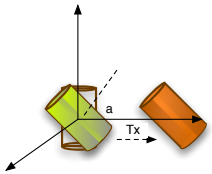


OpenGL :
Les instructions de transformations

```
glRotatef(45, 1, 0, 0);
glTranslatef(0, 5, 0);
trace_cylindre( );
```




```
glTranslatef(0, 5, 0);
glRotatef(45, 1, 0, 0);
trace_cylindre( );
```



OpenGL :
Gestion des transformations

- Pile de matrices
- Matrice courante = Sommet de pile
- Deux piles : GL_MODELVIEW et GL_PROJECTION
  - Modelview : pile de profondeur 32
  - Projection : pile de profondeur 2
- Définir la matrice courante

  (= affectée par les prochaines transformations)
  - void glMatrixMode (GLenum mode)

    mode = GL_MODELVIEW ou GL_PROJECTION

OpenGL :
Opérations sur les piles

- Charger la matrice identité dans la matrice courante
  - void glLoadIdentity (void)
- Charger la matrice courante avec celle spécifiée
  - void glLoadMatrix{f, d} (const TYPE *mat)
- Multiplier la matrice courante par celle spécifiée
  - void glMultMatrix {f, d} (const TYPE *mat)
- Récupérer la matrice courante
  - void glGetFloatv (GLenum name, GLfloat *mat)

    name : GL_MODELVIEW_MATRIX ou GL_PROJECTION_MATRIX

OpenGL :
Opérations sur les piles

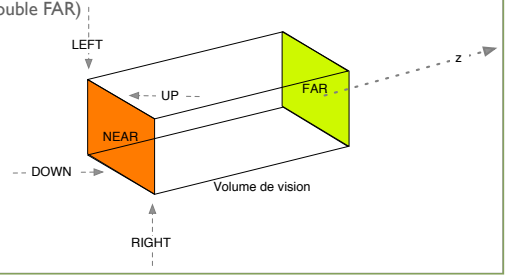
- Ajouter une matrice, sur la pile de courante, initialisée comme la précédente
  - void glPushMatrix (void)
- Enlever une matrice de la pile de matrices courante
  - void glPopMatrix (void)
- Ces deux instructions permettent d'appliquer à la fois des transformations absolues et relatives aux objets composant la scène

OpenGL : Transformation de vue

- Toutes les transformations précédentes
- Plus :
  - void gluLookAt (GLdouble ex, GLdouble ey, GLdouble ez,
    GLdouble cx, GLdouble cy, GLdouble cz,
    GLdouble ux, GLdouble uy, GLdouble uz)
  - positionne "l'oeil " au point E, oriente le point de vue dans la direction $\overrightarrow{EC}$ et défini le vecteur "haut" comme le vecteur $\overrightarrow{U}$
  - la matrice obtenue multiplie (à droite) la matrice courante

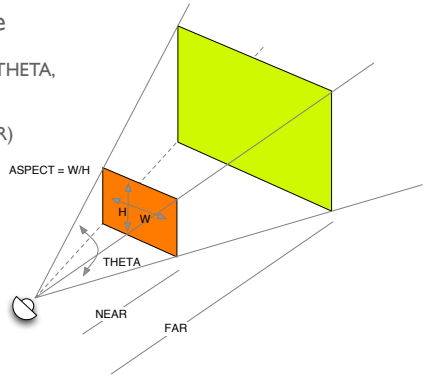
OpenGL : Transformation de projection

- Projection orthogonale
  - Occulter une coordonnée
  - void glOrtho (GLdouble LEFT, GLdouble RIGHT, GLdouble DOWN, GLdouble UP, GLdouble NEAR, GLdouble FAR)



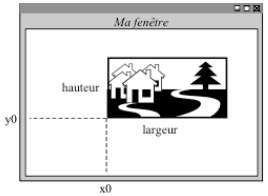
OpenGL : Transformation de projection

- Projection en perspective
  - void gluPerspective (GLdouble THETA,
    GLdouble ASPECT,
    GLdouble NEAR, GLdouble FAR)



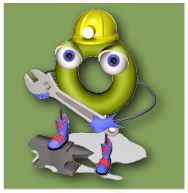
OpenGL : Le cadrage

- C'est maintenant qu'on tire la photo !
- void glViewport (GLint x0, GLint y0,
  GLsizei largeur, GLsizei hauteur)



E

D M



OpenGL : Le ZBuffer

- Un objet peut cacher (en partie ou totalement) un autre objet
  - dépend du point de vue (qui peut varier)
  - à priori indépendant de l'ordre de construction
  - en pratique, le dernier tracé écrase l'existant
- Que faut-il afficher ? Y a t-il un ordre ?
- Le Zbuffer : tableau des distances au point de vue
  - s'affranchir de l'ordre de construction des objets
  - taille de la fenêtre de projection

OpenGL : Le ZBuffer

- "Tracé" d'un objet par le ZBuffer
- Pour tous les pixels de l'objet dans le plan de projection :
  - Evaluer la profondeur (distance au point de vue)
  - Si la profondeur est supérieure à la valeur enregistrée dans le ZBuffer, abandon
  - Sinon le ZBuffer enregistre sa profondeur et le pixel reçoit ses valeurs chromatique

OpenGL : Le ZBuffer

- Activer / Désactiver le ZBuffer
  - glEnable (**GL_DEPTH_TEST**) / glDisable (GL_DEPTH_TEST)
  - peut être utilisé à tous moments (par exemple pour tracer "par dessus" une scène)
- Initialiser le ZBuffer
  - glClear (GL_COLOR_BUFFER_BIT | **GL_DEPTH_BUFFER_BIT**)
- Le contexte graphique doit prévoir l'utilisation d'un ZBuffer
  - Par exemple, avec GLUT,
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | **GLUT_DEPTH**)

OpenGL : Avant et Après avoir tracé

- Avant
  - on efface les buffers
  - void glClear (GLbitfield mask)

    mask : GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT ...
- Après
  - on force l'éxécution de toutes les commandes : void glFlush (void)
  - idem en attendant la fin de l'éxécution : void glFinish (void)
  - Les commandes "systèmes" qui swappent les buffers en mode double flushent automatiquement. Par exemple, **void glutSwapBuffers (void)**

OpenGL

La couleur

L'éclairage

Les textures

La couleur : Sujet vaste et complexe

- Physique
- Physiologie
- Psychologie
- Art
- ...

La couleur : Système achromatique

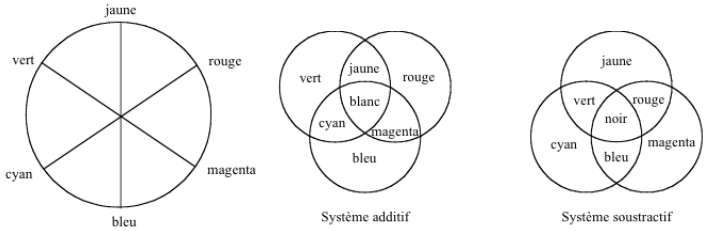
- "Sans couleur"
- Information : quantité de lumière (surface et durée)
  - Intensité ou luminance (cadre physique)
  - Luminosité (perception, cadre psychologique)
- Nuances de gris
  - Niveau d'énergie compris entre le noir (absence de lumière) et le blanc (saturation de lumière)
  - En pratique : 256 niveaux de gris

La couleur : Que l'on voit ...

- Lumière visible entre 400 et 700 nm
- Issue soit :
  - D'une source active (lampe, soleil ...)
  - D'une source passive qui restitue une partie de la lumière reçue
- La perception de la couleur d'un objet dépend :
  - De la distribution des longueurs d'onde de la source lumineuse
  - Des caractéristiques physiques de l'objet (objet rouge, lumière verte => noir)
  - Des positions relatives de la source, l'objet et l'observateur

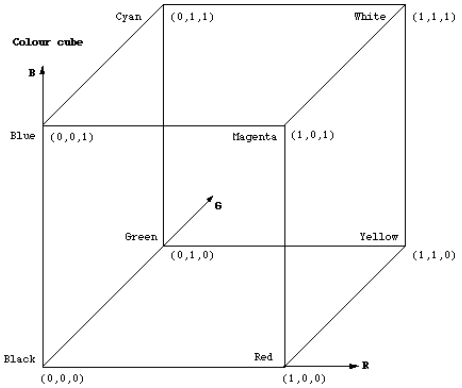
La couleur : Systèmes soustractifs et additifs

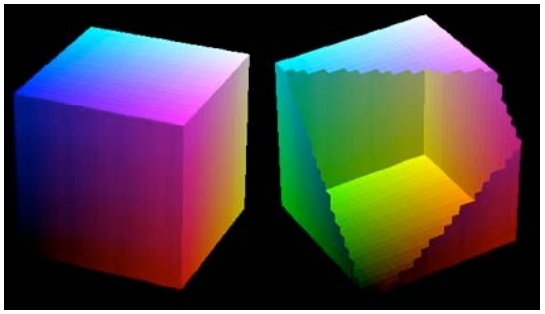
- Construction d'une couleur perçue à partir de trois couleurs primaires
  - Mélange additif : le rouge, le vert et le bleu (écran cathodique ...)
  - Mélange soustractif : le cyan, le magenta et le jaune (imprimante ...)

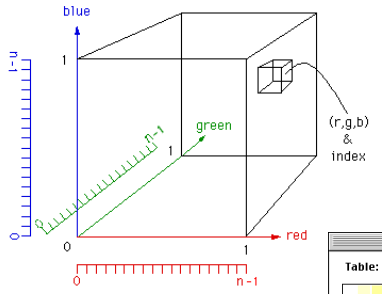


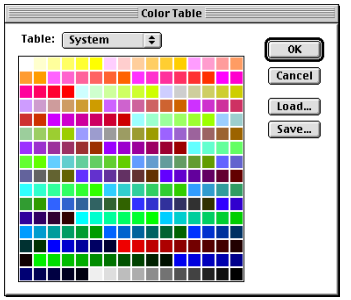
La couleur : Espaces de couleur

- RGB (Red, Green, Blue)
- HLS (Hue, Lightness, Saturation)
- CIE (Brightness, Hue, Colorfulness)
- Passage d'un espace de couleur à un autre par une transformation souvent non linéaire









La couleur : ... en OpenGL

- Mode RGB (le plus fréquemment)
- glColor3**f** (R, G, B)
  - Chaque composante est comprise dans [0.0 ... 1.0]
- glShadeModel (mode)
  avec mode = GL_FLAT ou GL_SMOOTH
- glEnable (GL_COLOR_MATERIAL) pour utiliser à la fois l'éclairage et glColor

L'éclairage : Amélioration du rendu

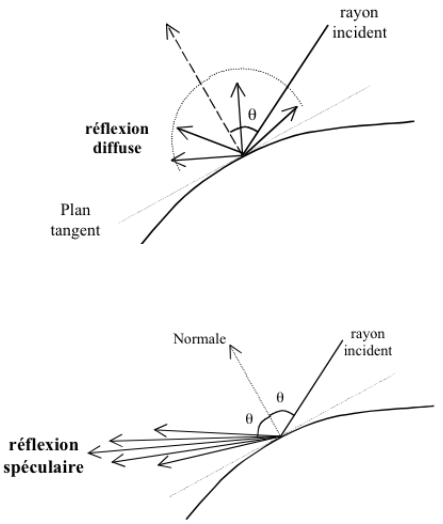
- Rendu réaliste : interaction de la lumière / objets
- Nombreux travaux de recherche - Solutions proposées très coûteuses suivant les effets recherchés
- Communication Homme / Machine
  - Une image simplifiée peut-être plus "réussie" qu'une "photographie"
  - La réalité peut-être intentionnellement altérée, voir même faussée
- 3D -> 2D : perte d'information
- Utiliser l'inconscient pour restituer cette information (perpective, brouillard, éclairage ...)

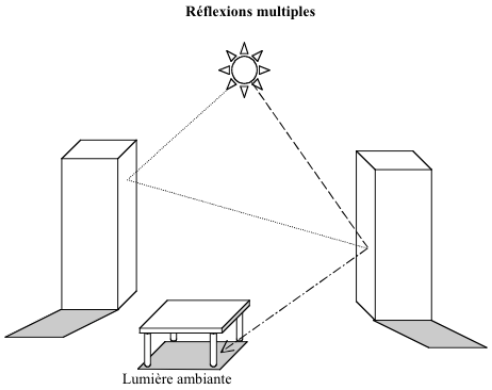
L'éclairage : Définir un éclairage

- Position et propriétés des "lumières"
- Propriétés des matériaux optiques des objets
- Orientation des surfaces

L'éclairage : Plusieurs types

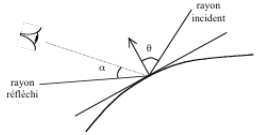
- Un rayon lumineux sur une surface
  - Une partie est absorbée (filtrage)
  - Une autre est réfléchie (lois de la normale) : réflexion spéculaire
  - Le reste est diffusé dans toutes les directions : réflexion diffuse
- Les objets peuvent avoir une lumière émissive
  - Ajoute de l'intensité à l'objet
  - N'ajoute pas d'éclairage supplémentaire à la scène
- Les réflexions multiples sur les objets génèrent la lumière ambiante qui n'a pas d'orientation privilégiée et éclaire les zones d'ombre.





L'éclairage : Le modèle Lambertien

- Intensité ambiante
  - Provient de l'environnement
- Réflexion diffuse
  - Provient d'une direction
- Réflexion spéculaire
  - Provient d'une direction et est réfléchie dans une direction privilégiée
- Lumière emissive



$$L = L_a + k_d cos\theta + k_s cos^n \alpha$$

L'éclairage : En OpenGL

- Activer/Désactiver le mode éclairage
- Définir les propriétés des lumières
- Positionner les lumières
- Allumer/Eteindre les lumières
- Choix du type de rendu (lisse/facettes)
- Définir les faces visibles
- Définir des matériaux (propriétés physiques)
- Définir une normale
- =>Tracer une primitive

L'éclairage : En OpenGL

- glEnable / Disable (GL_LIGHTING)
- 8 lumières (GL_LIGHT0 ... GL_LIGHT7)
- glEnable / Disable (GL_LIGHTi)
- Assimilé à un objet de la scène
  - Subit les transformations géométriques (GL_MODELVIEW)
  - Fixe, liée à un objet, liée à une scène, liée au point de vue ...
- glLightfv (GL_LIGHTi, Attribut, Vecteur de float)
- Seule "GL_LIGHT0" est "visible" par défaut

L'éclairage : Les attributs OpenGL

- La position : GL_POSITION

```
GLfloat Lposition1 [4] = {-5.0, 0.0, 3.0, 0.0}; /* lumière a l'infini */
GLfloat Lposition2 [4] = {-5.0, 0.0, 3.0, 1.0}; /* position "reelle" */
glLightfv (GL_LIGHT0, GL_POSITION, Lposition1);
```

- L'intensité et la couleur : GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR

```
GLfloat Lambiant [4] = {0.4, 0.4, 0.4, 1.0};
GLfloat Lblanche [4] = {1.0, 1.0, 1.0, 1.0};

glLightfv (GL_LIGHT0, GL_AMBIENT, Lambiant);
glLightfv (GL_LIGHT0, GL_DIFFUSE, Lblanche);
glLightfv (GL_LIGHT0, GL_ SPECULAR, Lblanche);
```

L'éclairage : Les matériaux

- Un objet restitue une partie de la lumière perçue
  - Matériau (cuivre, argent ...)
  - Etat de surface (brillant, mat ...)

```
GLfloat Lnoire [4] = {0.0, 0.0, 0.0, 1.0};
GLfloat mat_diffuse [4] = {0.057, 0.441, 0.361, 1.0};
GLfloat mat_specular [4] = {0.1, 0.1, 0.5, 1.0};
GLfloat mat_shininess [1] = {50.0};

glMaterialfv (GL_FRONT_AND_BACK, GL_EMISSION, Lnoire);
glMaterialfv (GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
glMaterialfv (GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
glMaterialfv (GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
```

L'éclairage : Vecteur normal

- Enoncé des sommets dans le sens trigonométrique
- Vecteur normal unitaire sinon : glEnable(GL_NORMALIZE) // COUTEUX
- Le plus souvent il faut le calculer : produit vectoriel
- Spécifier les normales aux sommets

```
glBegin(GL_TRIANGLES);
glNormal3f(0., 0., 1.) ;
glVertex3f(0., 0., 0.);
glVertex3f(5., 0., 0.);
glVertex3f(2.5, 5., 0.);
glEnd();
```

```
glBegin(GL_TRIANGLES);
glNormal3f(0., 0., 1.) ;
glVertex3f(0., 0., 0.);
glNormal3f(1., 0., 0.) ;
glVertex3f(5., 0., 0.);
glNormal3f(0., 1., 0.) ;
glVertex3f(2.5, 5., 0.);
glEnd();
```

L'éclairage : Extrapolation

- Extrapolation de la lumière
  - glShadeModel(GL_FLAT / GL_SMOOTH)
  - Modèle de Gouraud

- Face avant / arrière
  - glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE / **GL_FALSE**)

Par exemple

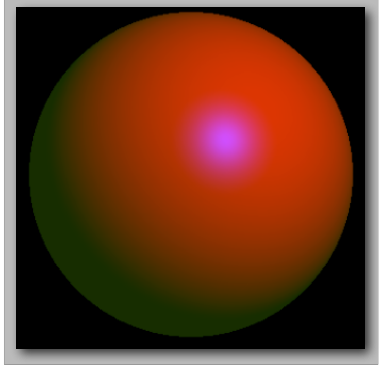
3 composantes :

- Ambiante : rouge
- Diffuse : jaune
- Spéculaire : blanche

Par exemple

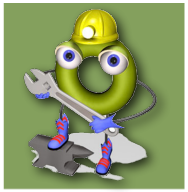
Une lumière blanche sur une sphère définie avec les matériaux suivants :

- Ambiant : vert
- Diffus : rouge
- Spéculaire : bleu



E

D M



OpenGL

Les images
Les textures
+ + +

Qu'est ce qu'une image ?

- Prend la forme d'un tableau rectangulaire
- Stockage dans un fichier
- Représentation interne au logiciel
- Représentation pour l'affichage
  - Dynamique, zoom ...
- TIFF, GIF, JPEG, PNG, ...
  - Bibliothèques spécialisées
  - Image Magick (www.imagemagick.org), DevIL (www.imagelib.org), ... etc ...

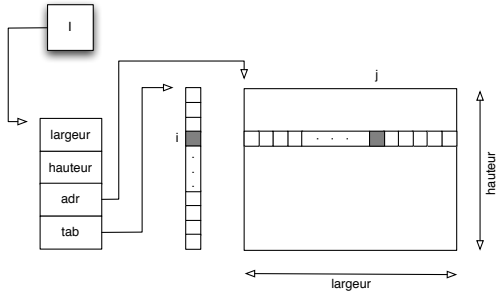
Le format P[B, G, P]M

- Le format PGM
  - Une en-tête (P2 / P4, ...)
  - Les données (ASCII / Binaire)

```
P2
# une image pgm ascii
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 3 0 0 7 7 7 7 0 0 0 11 11 11 11 0 0 0 15 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Représentation interne

- I->tab[i][j] = intensité du pixel (i, j)



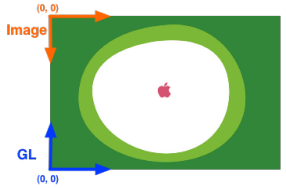
Par exemple en C

```c
/* Definition d'un type image 2D */
typedef unsigned char PIXEL;
typedef PIXEL *LIGNE;
typedef struct
{ int larg, haut;
  PIXEL *adr;
  LIGNE *tab;
} *IMAGE, structImage;
```

```c
IMAGE creerImage(int l, int h)
{ IMAGE I;
  PIXEL *p;
  int i;
  I = (IMAGE) malloc(sizeof(structImage));
  I->larg = l;
  I->haut = h;
  I->adr = (PIXEL*) malloc(sizeof(PIXEL)*h*l);
  I->tab = (LIGNE*) malloc(sizeof(LIGNE)*h);
  p = I->adr ;
  for (i=0; i<h ; i++, p+=l)
    I->tab[i]= p ;
  return I;
}
```

OpenGL et les images

- Représentation pour l'affichage : GLubyte * I_Aff
- Matrice de projection
  - gluOrtho2D(xmin, xmax, ymin, ymax)
  - Concordance pixel image / coordonnée fenêtre




```c
void redim(int large, int haut)
{
  glViewport(0, 0, large, haut) ;
  glMatrixMode(GL_PROJECTION) ;
  glLoadIdentity() ;
  gluOrtho2D(0, large, 0, haut) ;
  glMatrixMode(GL_MODELVIEW) ;
}
```

OpenGL et les images

- Affichage
  - Tracé : glDrawPixels(largeur, hauteur, format, type, tableau)
  - Position : glRasterPos2i(x, y)
  - Zoom : glPixelZoom(Xzoom, Yzoom)
- Une options importante
  - Alignement des octets :

    glPixelStorei(GL_UNPACK_ALIGNMENT, taille)

Un peu de code ...

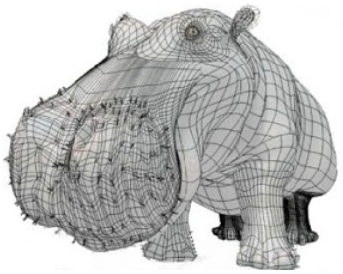
```
IMAGE I;
...
void affichage( )
{
  glClear(GL_COLOR_BUFFER_BIT) ;
  /* positionnement de l'origine pour glDrawPixels */
  glRasterPos2i(0,0);
  /* envoi des données */
  glDrawPixels(I->larg,I->haut,GL_LUMINANCE,GL_UNSIGNED_BYTE,I->adr);
  glutSwapBuffers();
}
```

Un peu de code ...

```
int main(int argc, char** argv)
{
  I = LireImagePgm("cratere.pgm");

  /* initialisation de la fenetre */
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (I->larg, I->haut);
  glutCreateWindow ("Image2D");

  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

  glutDisplayFunc(affichage);
  glutReshapeFunc(redim);

  glutMainLoop();

  return 0;
}
```

La texture

- Pour simuler la complexité des détails sur une surface, on peut augmenter le nombre de polygones définissant la surface
- Cependant, il en résulte une augmentation de la complexité de la modélisation (taille de l'objet, coût de génération, etc.) et du rendu (visibilité, illumination, ...)
- Il est possible de simuler l'effet visuel des détails en apposant une texture sur une surface tel du papier peint
- La "complexité" de la texture n'affecte pas la géométrie de l'image

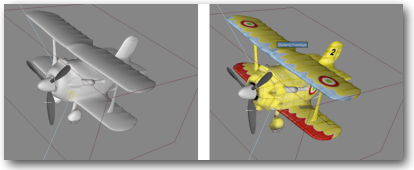


Quelques définitions

- **Texture mapping :**
  - technique consistant à appliquer une texture sur une surface
- **Texture map :**
  - texture apposée sur une surface
- **Texel :**
  - élément de base d'une texture, analogue au pixel d'une image

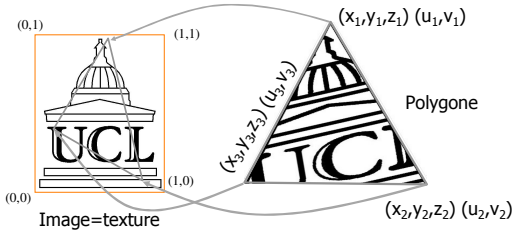
La texture : Image / Surface

- On "colle" une image sur un objet
- Quelle partie de l'image doit être collée sur quelle partie de l'objet ?
- Comment faut-il combiner l'image avec les attributs de lumière et de matériaux de l'objet ?



Le placage de texture

- A chaque sommet
  - Coordonnées textures (u,v)
  - (u,v) correspond à une position dans l'image (texture)



La texture : ... en OpenGL

- Une texture est un tableau de données
  - Les éléments du tableaux sont appelés "texels"
  - Couleur, luminance, alpha ...
  - 1D, **2D**, 3D
- Définir une texture
  - A partir d'une image : niveaux de gris, rgb, rgba ? entiers, réels ? ...
- Activer le placage de texture
- Définir comment la texture est appliquée aux objets
- Afficher la scène

En OpenGL

- Initialisation de la texture
  - glEnable(GL_TEXTURE_2D);
  - glGenTextures(1,&texID);
  - glTexImage2D(GL_TEXTURE_2D, 0,GL_RGBA,..., buffer);
- Choix de la texture courante
  - glBindTexture(GL_TEXTURE_2D, texID);
- Paramètres (répétition, interpolation...)
  - glTexParameteri(GL_TEXTURE_2D, ... , ...)

En OpenGL

- Mélange avec les couleurs des faces
  - glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE/BLEND/REPLACE/DECAL);
- Coordonnées de textures à spécifier pour chaque sommet
  - glTexCoord2f(u, v);
  - glVertexf(x, y, z);
- Attention les dimensions de la texture sont obligatoirement des puissances de 2
- En général au plus 1024 x 1024 ...

La texture : ... en OpenGL

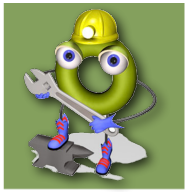
```
void GL_init(void)
{
I = LireImagePNG();

// défini une texture à partir d'une image (dernier arg : pointeur)
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGBA, GL_UNSIGNED_BYTE, I);

// comment la texture est combinée avec les pixels
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

// filtres
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// activation
glEnable(GL_TEXTURE_2D);
}
```

La texture : ... en OpenGL

- Utilisation de la texture : Affectation des coordonnées de textures
  - Manuellement
  - Généré par OpenGL (SphereMap, EnvironmentMap ...)

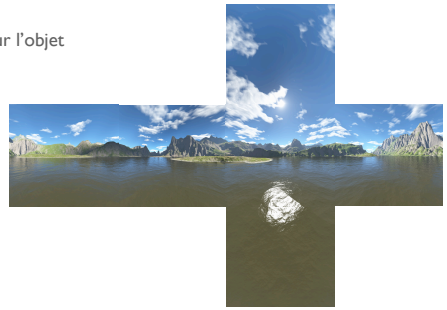
```
// mise en correspondance manuelle des coordonées de texture
glBegin(GL_TRIANGLES);
                glTexCoord2f(0.0, 0.0);
                glVertex2f(-1.0, -1.0);
                glTexCoord2f(1.0, 0.0);
                glVertex2f(1.0, -1.0);
                glTexCoord2f(0.0, 1.0);
                glVertex2f(0.0, 1.0);
glEnd();
```

E D M



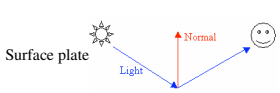
Texture d'environnement : cubemap

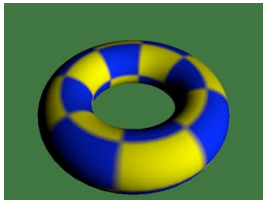
- Un cube texturé par 6 images enveloppe la scène
  - Impression d'espace
  - Reflet de l'environnement sur l'objet

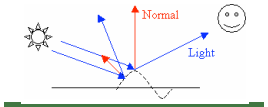


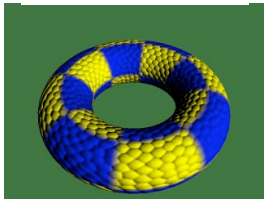
Bump Maps

- Utilisation d'une carte de normales pour donner une impression de relief

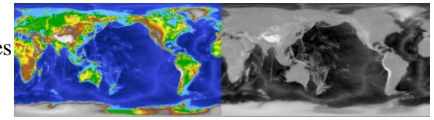




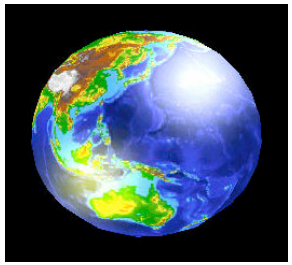




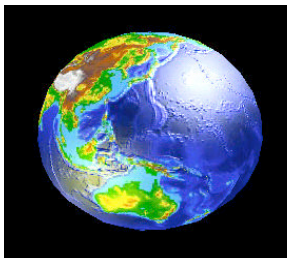
Texture des normales



Texture des couleurs



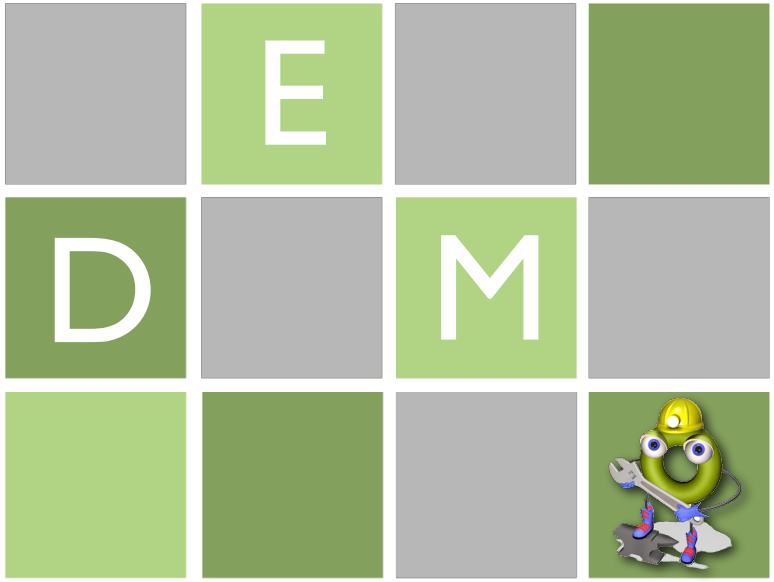
Rendu "classique"



Rendu avec bump-mapping

En conclusion

- Texture
  - Outils très puissant
  - Simple
  - Réaliste
- Egalement de nombreux problèmes
  - Paramétrisation
  - Génération
  - De près ne remplace pas la géométrie



Les listes d'affichage

- Stocker une liste de d'instructions OpenGL
  - Pour accélérer le rendu : il ne reste que des primitives OpenGL
- Instructions
  - Création / Suppression : glGenList(int) / glDeleteList(int, int)
  - Déclaration : glNewList(int, int) / glEndList ( )
  - Utilisation : glCallList (int)
- Les changements d'états OpenGL persistent après l'appel d'un liste
- Une liste peut en appeler une autre

Créer une liste d'affichage

- Création

```
GLuint id;
void init( void ){
   id = glGenLists( 1 );
   glNewList( id, GL_COMPILE );
   /* instructions du rendu */
   glEndList();
}
```

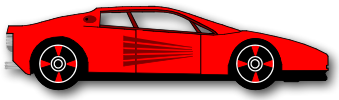
- Utilisation

```
void display( void ) {
   ...
   glCallList( id );
   ...
}
```

Liste d'affichage : Hiérarchie d'une scène

- Une voiture
  - Une liste pour le châssis
  - Une liste pour les roues

```
glNewList( VOITURE, GL_COMPILE );
   glCallList( CHASSIS );
   glTranslatef( … );
   glCallList( ROUE );
   glTranslatef( … );
   glCallList( ROUE );
        …
glEndList();
```

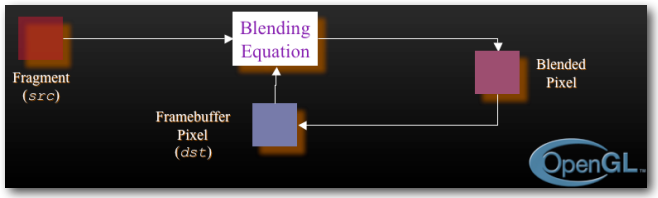


La transparence

- Alpha : la 4° composante de la couleur !
- Contrôle de l'opacité
  - simuler des objets transparents
  - le verre, l'eau ... etc
  - "mixer" des images
  - antialiasing
  - valeur ignorée si le "blending" n'est pas activé
- glEnable(GL_BLEND)

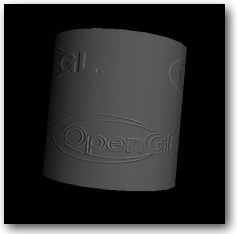
La transparence

- Combinaison des pixels arrivant et du contenu de la mémoire image
- Définition du type de combinaison
  - glBlendFunc (src, dst);



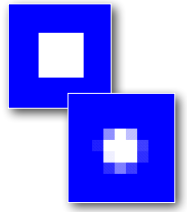
Rendu en plusieurs passes

- En combinant plusieurs passes, il est possible de mettre en oeuvre des algorithmes plus complexes
  - par exemple ici un bump mapping



Antialiassing

- La transparence doit être activée
  - la valeur des pixels est calculée par "recouvrement"
- Trois types : glEnable (...)
  - GL_POINT_SMOOTH
  - GL_LINE_SMOOTH
  - GL_POLYGON_SMOOTH

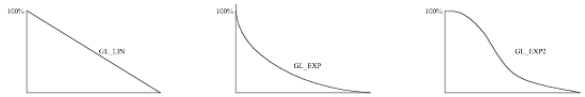


Le brouillard

- Passage 2D / 3D : perte d'information
- L'observateur doit pouvoir la reconstruire au mieux
- Perspective : sensation de profondeur
- Brouillard : renforcement de la sensation de profondeur

Le brouillard

- Estompe les objets en fonction de leur distance au point de vue
  - Activation : glEnable (GL_FOG)
  - Association d'une couleur : GLfloat fogColor[4] = { .5, .5, .3, 1.}
  - En général même couleur pour le fond (glClearColor (...))
  - 3 fonctions de mélange : GL_LINEAR, GL_EXP, GL_EXP2



Pourcentage de la couleur d'origine d'après la distance au point de vue