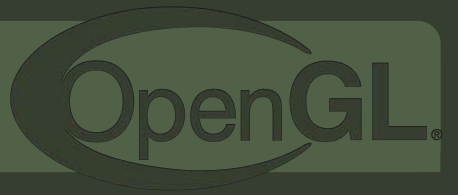


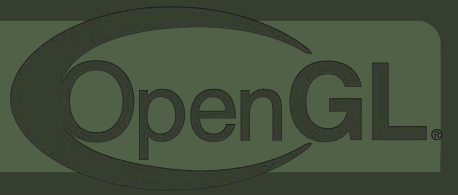
Vincent Forest

forest@irit.fr

http://www.irit.fr/~Vincent.Forest/

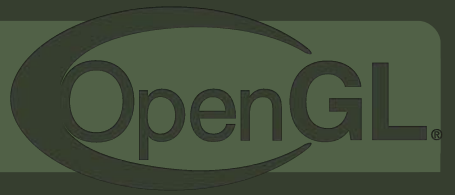
Master Professionnel IIN, 2006 - 2007

Un peu d'histoire



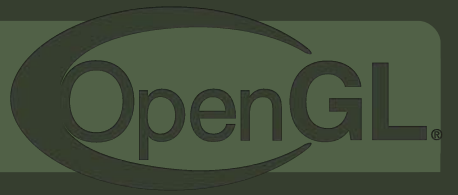
- 1990: PHIGS VS IrisGL
  - PHIGS: API standard et ouverte
  - IrisGL: API propriétaire de SGI
  - IrisGL -> OpenGL (standard et ouvert)
- 1992: OpenGL 1.0
  - Création de l' OpenGL **A**rchitectural **R**eview **B**oard (OpenGL ARB)
- 1995: Direct3D
  - Microsoft, SGI et HP tetent d'unifier D3D et OGL (Fahrenheit)

Un peu d'histoire



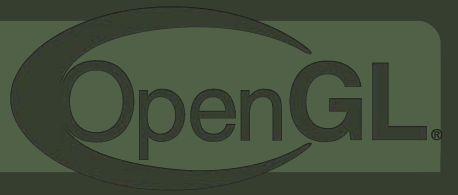
- 1998: OpenGL 1.2 & 1.2.1
  - Introduction des extensions ARB
- 2001/2002/2003: OpenGL 1.3/1.4/1.5
- 2004: OpenGL 2.0
  - Microsoft quitte l'OpenGL ARB
- 2005: Microsoft craque (OGL en surcouche de D3D !)
- 2006: OpenGL passe sous le contrôle du Khronos Group
  - Microsoft s'assagit

OpenGL?



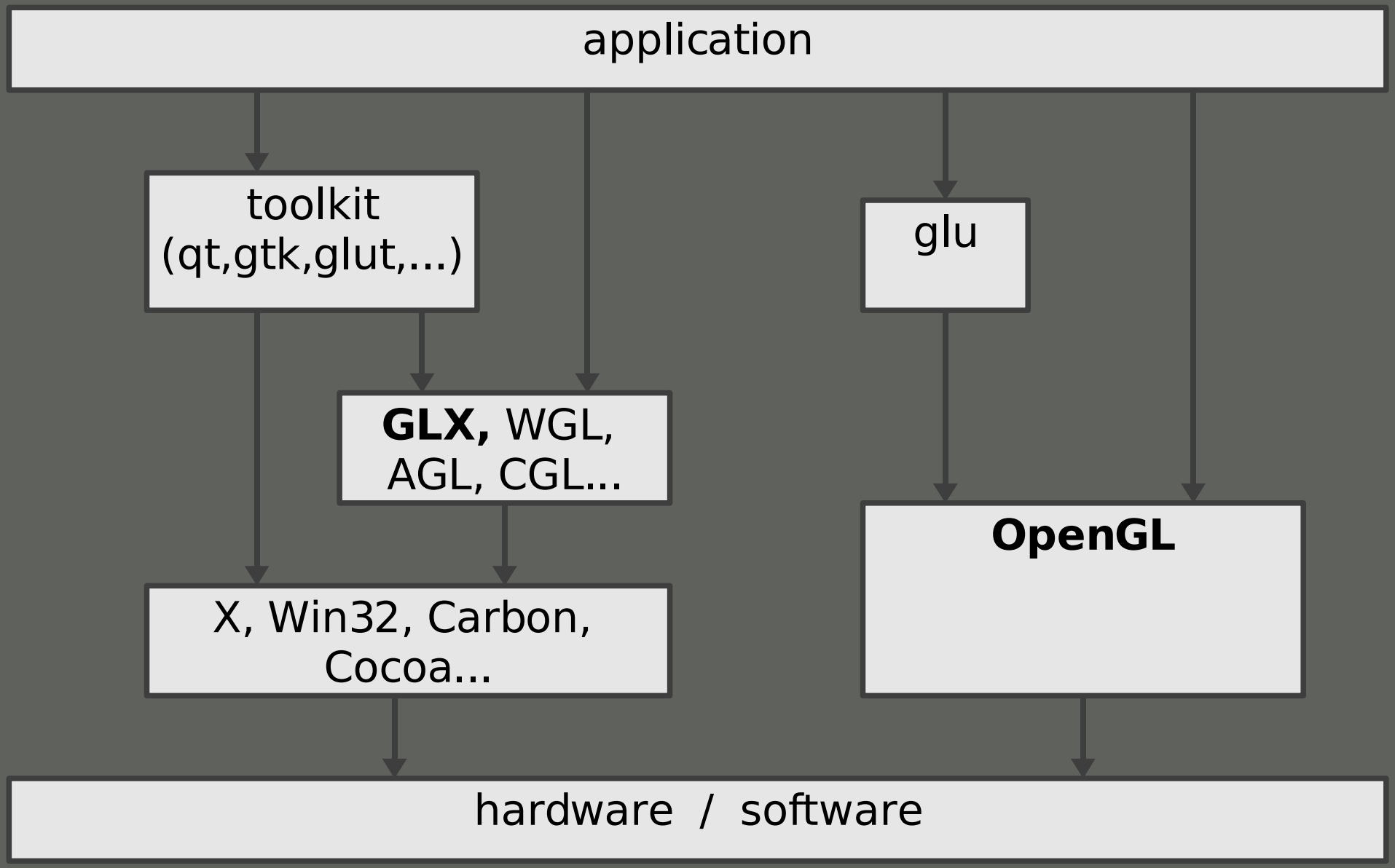
- **Open G**raphics **L**ibrary
  - API standard
  - Multi-plateforme (Unix, Linux, Windows, BSD, MacOS...)
- Une Spécification
  - Décrit un ensemble de fonctions et leur comportement
  - Implentation à la charge des IHV (**I**ndependent **H**ardware **V**endors)
    - Librairie accélérée en hardware
- Masque l'interfaçage avec l'accélération matérielle
- Masque les capacités des plateformes matérielles

OpenGL?



- Machine à états (pipeline graphique)
  - Etat Serveur (GL serveur) / Etat Client (GL client)
  - Controlée par des commandes
  - Etages fixes: OpenGL 1.x
  - Etages programmables: OpenGL 2.x (GLSL)
- API procédurale bas niveau
  - Description de toutes les étapes
  - Requiert une bonne connaissance du pipeline graphique

Architecture logicielle



La Syntaxe des commandes

- Constantes
  - En majuscule
  - Préfixées par **GL**
- Commandes
  - Préfixées par **gl**
  - Suffixées par
    - Le **nombre** d'arguments
    - Le **type** des arguments
  - Scalaires / **V**ectorielles

```
GL_TEXTURE_2D


glBegin(), glEnd()


glVertex2f(float arg1, float arg2)

glVertex2f(float arg1, float arg2)

glVertex3fv(float args[3])
```

```
rtype Nom {Ø1234}{Ø b s i f d ub us ui}{Øv}
        ( [args,] T arg1, ..., T argN [, args] );
```

Type de données

- Correspondance entre les lettres du suffixe d'une commande et le type de ses arguments

Lettre	Type OpenGL correspondant
b	GLbyte
s	GLshort
i	GLint
f	GLfloat
d	GLdouble
ub	GLubyte
us	GLushort
ui	GLuint

- Les Types OpenGL ne sont pas des Types C !

Types OpenGL

Type OpenGL	Nombre de bits Minimum	Description
GLboolean	1	Booléen
GLbyte	8	Nombre entier binaire signé
GLubyte	8	Nombre entier binaire non signé
GLuchar	8	Charactère
GLshort	16	Nombre entier binaire signé
GLushort	16	Nombre entier binaire non signé
GLint	32	Nombre entier binaire signé
GLuint	32	Nombre entier binaire non signé
GLsizei	32	Nombre entier binaire positif de taille
GLenum	32	Valeur entière binaire d'énumération
GLintptr	ptrbits	Nombre entier binaire signé
GLsizeiptr	ptrbits	Nombre entier binaire positif de taille
GLbitfield	32	Champs de bits
GLfloat	32	Valeur réelle
GLclampf	32	Valeur réelle bornée entre [0,1]
GLdouble	64	Valeur réelle
GLclampd	64	Valeur réelle bornée entre [0,1]

Gérer les états

- Une variable d'état peut être
  - Activée/désactivée
    - ```
      void        glEnable     (GLenum pname);
      void        glDisable    (GLenum pname);
      ```
  - Lue/testée
    - ```
      void        glGetTypev   (GLenum pname, Type *param);
      GLboolean   glIsEnable   (GLenum pname);
      ```
  - Enregistrée/restaurée (par groupe d'attributs)
    - ```
      void        glPushAttrib (GLbitfield mask);
      void        glPopAttrib  ();
      ```
      ```

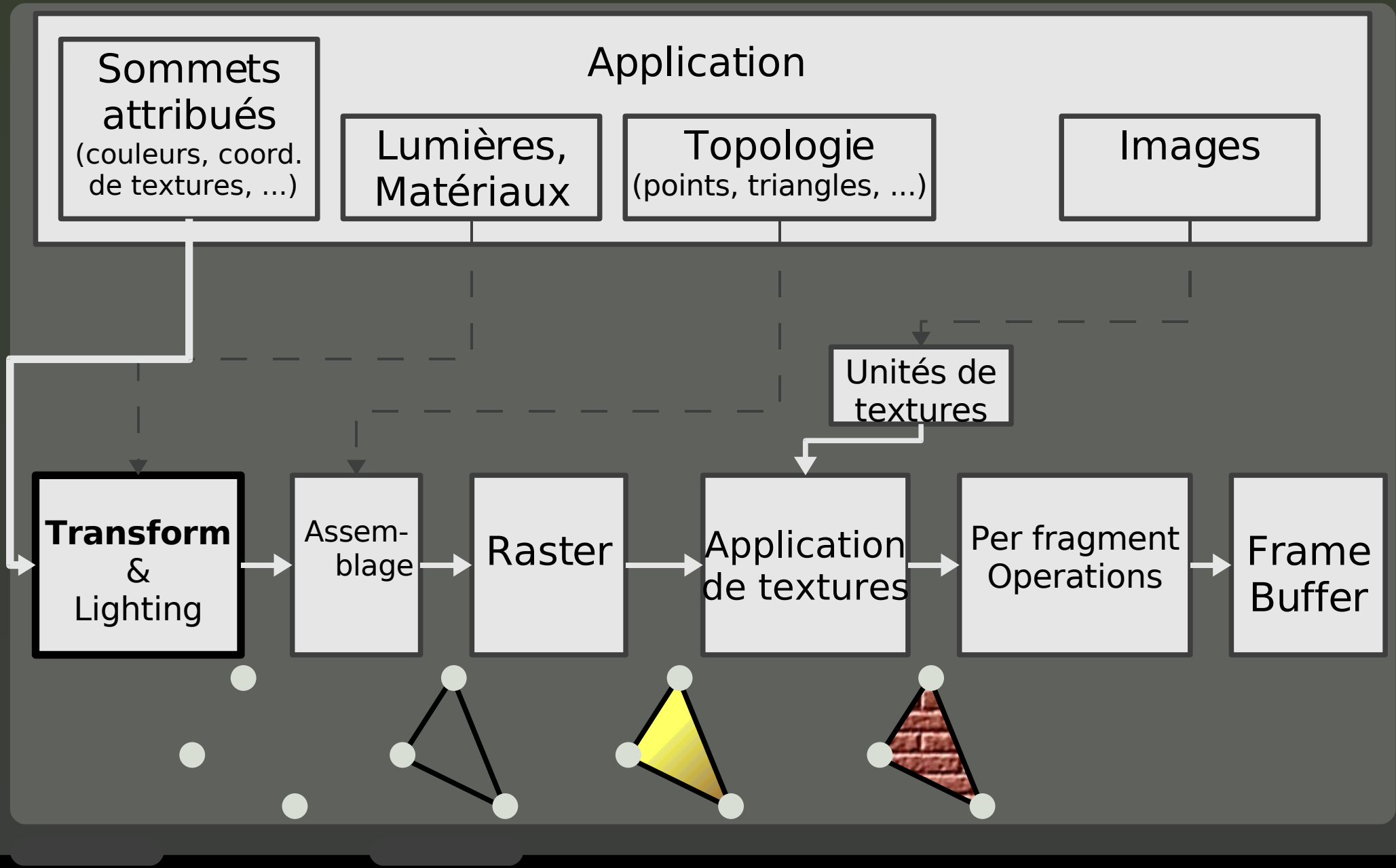
Gestion des erreurs

- Erreurs OpenGL
  - Chaque erreur détectable est associée à un code numérique
  - ```
    GLenum glGetError();
    /*
    GL_NO_ERROR
    GL_OUT_OF_MEMORY
    GL_INVALID_ENUM
    GL_INVALID_VALUE
    GL_INVALID_OPERATION
    GL_STACK_OVERFLOW
    GL_STACK_UNDERFLOW
    GL_TABLE_TOO_LARGE
    ...
    */
    ```
  - ```
    GLubyte*  gluErrorString(GLenum error_code);
    ```

Rendu d'une scène 3D

- En entrée
  - Un point de vue
  - Une description de la scène (ensemble de polygones)
  - Des attributs de matériaux associés à chaque object
  - Un ensemble de lumières
- En sortie
  - Un tableau de couleur RGB
  - Frame buffer: zone en mémoire vidéo qui contient l'image calculée, juste avant son envoi à l'écran.

Pipeline de rendu



Algorithme de rendu

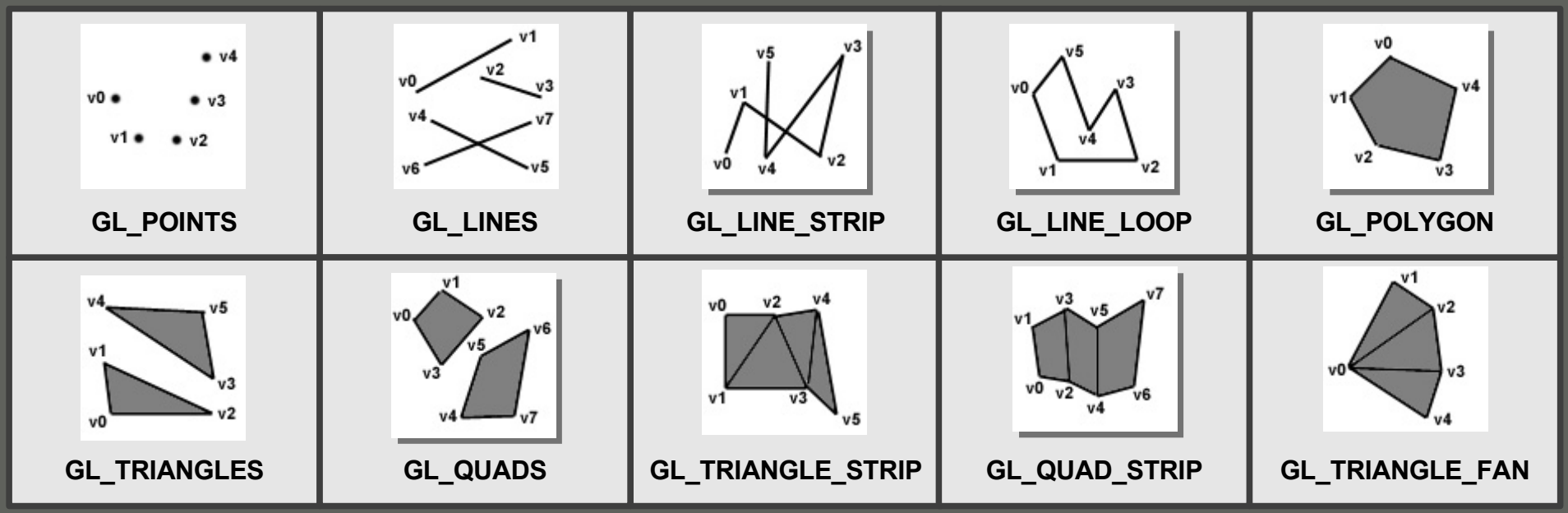
1 - Effacer les tampons de destinations
2 - Positionner la caméra
3 - Activer les sources lumineuses

Pour chaque object
    4 - Sauvegarder les états
    5 - Activer ses propriétés de matériau
    6 - Positionner l'objet (mettre à jour la matrice de modélisation)
    7 - Restaurer les états
FinPour

8 - Afficher l'image calculée
9 - calculer l'image suivante

Primitives Géométriques

- Définies par leurs sommets
  - ```
    void glVertex{234}{Øv}
            ( {Type *coords| Type arg1, ...,argN} );
    ```
- Organisation des sommets


    ```

Exemples

- Un cercle:

```
glBegin(GL_LINE_LOOP);
     for(float a=0 ; a<2*PI ; a+=0.1f)
        glVertex2f( cosf(a), sinf(a) );
glEnd();
```

- Un cube:

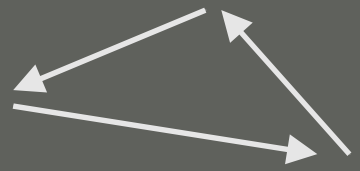
```
int c[8][3] =
  {{0,0,0},{1,0,0},{1,1,0},{0,1,0},{0,0,0},{1,0,0},{1,1,0},{0,1,0}};
glBegin(GL_QUAD_STRIP);
  glVertex3iv(c[0]); glVertex3iv(c[4]); glVertex3iv(c[1]); glVertex3iv(c[5]);
  glVertex3iv(c[2]); glVertex3iv(c[6]); glVertex3iv(c[3]); glVertex3iv(c[7]);
  glVertex3iv(c[0]); glVertex3iv(c[4]);
glEnd();

glBegin(GL_QUADS);
  glVertex3iv(c[0]); glVertex3iv(c[1]); glVertex3iv(c[2]); glVertex3iv(c[3]);
  glVertex3iv(c[7]); glVertex3iv(c[6]); glVertex3iv(c[5]); glVertex3iv(c[4]);
glEnd();
```

Primitives géométriques: les polygones

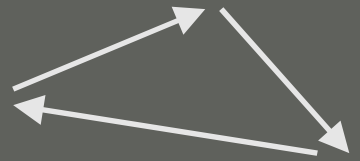
- Définir l'orientation des primitives
  - Sens trigonométrique

```
glFrontFace(GL_CCW);
```



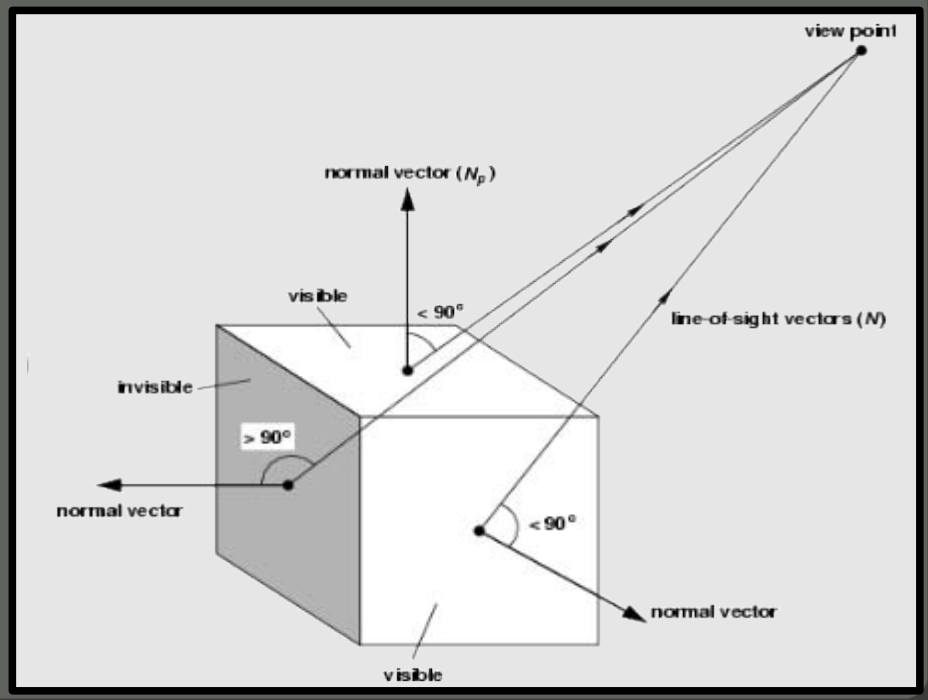
  - Sens des aiguilles d'une montre

```
glFrontFace(GL_CW);
```



- Elimination des faces arrières

```
glEnable/Disable(GL_CULL_FACE);
glCullFace      (GL_BACK);
```



Les attributs des sommets

- Position: `glVertex...`
- Couleur: `glColor...`
  - = un triplet rouge, vert, bleu (RGB)
  - Plus un coefficient de transparence (alpha, A)
  - Chaque composante est un réel entre 0 et 1
- Normale: `glNormal3...`
  - Toujours 3 composantes!
  - Une normale **doit être normalisée**

```
glEnable(GL_NORMALISE)   ;      /* couteux! */
glEnable(GL_RESCALE_NORMAL);/* uniquement avec des normales
  unitaires et mise à l'échelle uniforme */
```

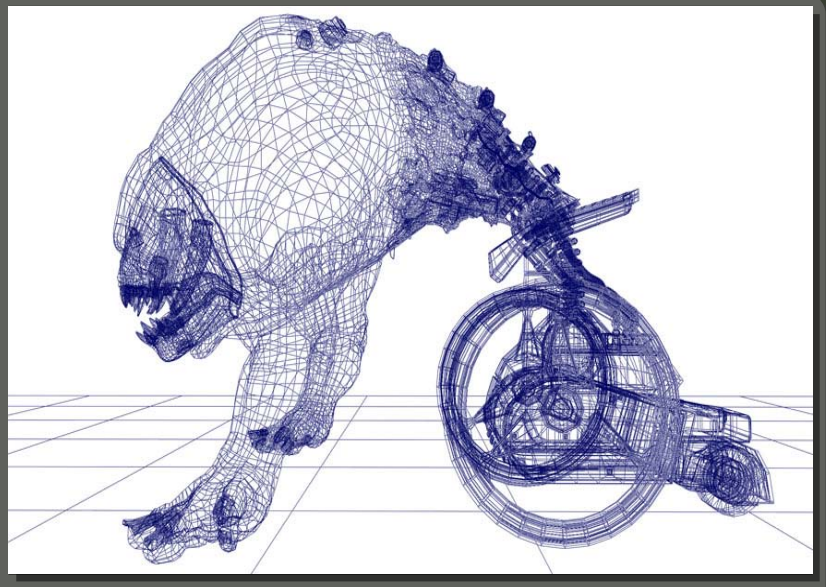
- Coordonée de texture: `glTexCoord...`

Un maillage

- Un maillage c'est:
  - Une liste de sommets attribués

```
typedef struct sommet_s
{
     GLfloat   position  [3];
     GLfloat   normale   [3];
     GLubyte   couleur   [4];
     GLfloat   texcoord  [2];
}sommet_t;

sommet_t sommets[nbrSommets];
```



  - Une liste de faces (triangulaires)
    - un triangle = indices des trois sommets

```
GLuint faces[3][nbrFaces];
```

Tracer d'un maillage

- En mode direct

```
glBegin(GL_TRIANGLES);
for(size_t i=0; i<nbrFaces; ++i)
     for(short j=0; j<3; ++j)
     {
        glNormal3fv  (sommets[faces[j][i]].normale);
        glColor3ubv  (sommets[faces[j][i]].couleur);
        glTexCoord2fv(sommets[faces[j][i]].texcoord);
        glVertex3fv  (sommets[faces[j][i]].position);
     }
glEnd();
```

- En pratique le mode direct n'est jamais utilisé (et va être supprimé)!
  - préférer les **vertex array**

Vertex Array

- Tableaux contenant les données des vertex dans l'espace mémoire **client**

```
gl{Vertex|TexCoord}Pointer(
   GLint size, GLenum type, GLsizei stride, GLvoid *pointer);
gl{Normal|Color}Pointer(
   GLenum type, GLsizei stride, GLvoid *pointer);
```

  - size = nbr de sommets
  - stride = nbr d'octets entre 2 sommets consécutifs
- L'état client d'un tableau est activé/désactivé individuellement

```
void glEnableClientState    (GLenum pname);
void glDisableClientState   (GLenum pname);
```

  - pname = GL_*_ARRAY;
    - * = VERTEX, NORMAL, TEXTURE_COORD, INDEX, ...

Vertex Array

- Utilisation de blocs de données de ces tableaux pour:
  - définir une primitive via son index: rarement utilisé

```
void glArrayElement(GLint index);
```

  - définir plusieurs primitives géométriques en une seule commande
    - Direct: Utilisé lorsque les sommets peuvent être envoyés linéairement (ex.: “triangle strip”)

```
void glDrawArrays (GLenum mode, GLint first, GLsizei count);
```

    - Via une liste d'index: adapté aux maillages

```
void glDrawElements(GLenum mode, GLsizei count, GLenum type,
  GLvoid *indices);
void glDrawRangeElements(GLenum mode, GLuint start, GLuint end,
  GLsizei count, GLenum type, GLvoid *indices);

/*Préférer glDrawRangeElements: le driver optimise si possible
  le format des indices, et gère mieux l'espace mémoire */
```

Vertex Array: exemple

```
glVertexPointer( 3, GL_FLOAT, sizeof(sommet_t),sommets[0].position);
glNormalPointer( GL_FLOAT, sizeof(sommet_t), sommets[0].normale);
glColorPointer( GL_UNSIGNED_BYTE, sizeof(sommet_t),sommets[0].couleur);
glTexCoordPointer( 2, GL_FLOAT, sizeof(sommet_t), sommets[0].texcoord);

glEnableClientState( GL_VERTEX_ARRAY );
glEnableClientState( GL_NORMAL_ARRAY );
glEnableClientState( GL_COLOR_ARRAY );
glEnableClientState( GL_TEXTURE_COORD_ARRAY );

glDrawRangeElements(GL_TRIANGLES, 0, maxIndex, 3*nbrFaces,
  GL_UNSIGNED_INT, faces);

glDisableClientState( GL_VERTEX_ARRAY );
glDisableClientState( GL_NORMAL_ARRAY );
glDisableClientState( GL_COLOR_ARRAY );
glDisableClientState( GL_TEXTURE_COORD_ARRAY );
```

Vertex Buffer Object (VBO)

- Les données des vertex sont contenues dans l'espace mémoire **serveur**
- Deux types de Vertex Buffer Object
  - GL_ARRAY_BUFFER: informations de sommets
  - GL_ELEMENT_ARRAY_BUFFER: indices de sommet
- Gestion d'un Buffer Object

```
void glGenBuffers    (GLsizei n, GLuint *buffers);
void glDeleteBuffers (GLsizei n, GLuint *buffers);
void glBindBuffer    (GLenum target, GLuint buffer);
```

Vertex Buffer Object

- Initialiser un VBO à partir de données contenues dans l'espace mémoire client

```
void glBufferData(GLenum target, GLsizeiptr size, const GLvoid*
  data, GLenum usage);
```

  - usage = **suggestion** de l'usage des données
    - GL_STREAM_{DRAW READ COPY}
    - GL_STATIC_{DRAW READ COPY}
    - GL_DYNAMIC_{DRAW READ COPY}

- Modifier une partie d'un VBO

```
void glBufferSubData(GLenum target, GLintptr offset, GLsizeiptr
  size, const GLvoid* data, GLenum usage);
```

Vertex Buffer Object

- Accéder aux données d'un VBO dans l'espace mémoire client

```
void *glMapBuffer (GLenum target, GLenum access);
void glUnmapBuffer(GLenum target);
```

- Vertex Array dans un VBO

```
GLuint vertexBuffer;

glGenBuffers(1, &vertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glBufferData(GL_ARRAY_BUFFER, nbrSommet*sizeof(sommet_t),
  sommets[0].position, GL_STATIC_DRAW);
/* ... */
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glVertexPointer( 3, GL_FLOAT, sizeof(sommet_t), NULL);
/* ... */
glDeleteBuffers(1, &vertexBuffer);
```

Vertex Buffer Object

- Tableau d'indices dans un VBO

```
GLuint indexBuffer;

glGenBuffers(1, &indexBuffer);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBuffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 3*nbrFaces*sizeof(GLuint),
  faces, GL_STATIC_DRAW);

/* ... */

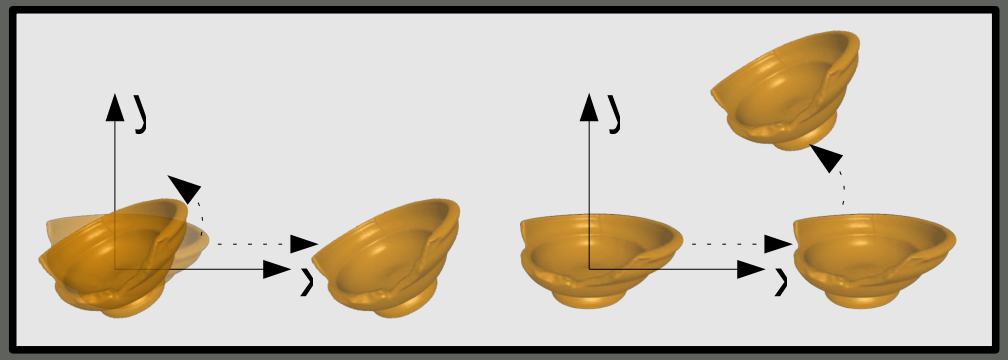
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBuffer);
glDrawRangeElementsEXT(GL_TRIANGLES, 0, 3*nbrFaces, 3*nbrFaces,
  GL_UNSIGNED_INT, NULL);

/* ... */

glDeleteBuffers(1, &indexBuffer);
```

Rappel sur les **Transformations**

- Utilisation des coordonnées homogènes
  - Point euclidien:
    - v = (x,y,z,w) ~ v'=(x',y',z') = (x/w, y/w, z/w) si w ≠ 0
    - w = 0: point à l'infini
  - Transformations affines: produit matriciel
    - v'= M * v v = M^{-1} * v'
  - Enchaînement de transformations = multiplication matricielle
    - v' = T * R * v
  - **ATTENTION:** le produit matriciel **n'est pas commutatif**
    - **T * R** * v ≠ **R * T** * v



Transformations élémentaires

- Translation:

```
void glTranslate{fd}(T dx, T dy, T dz);
```

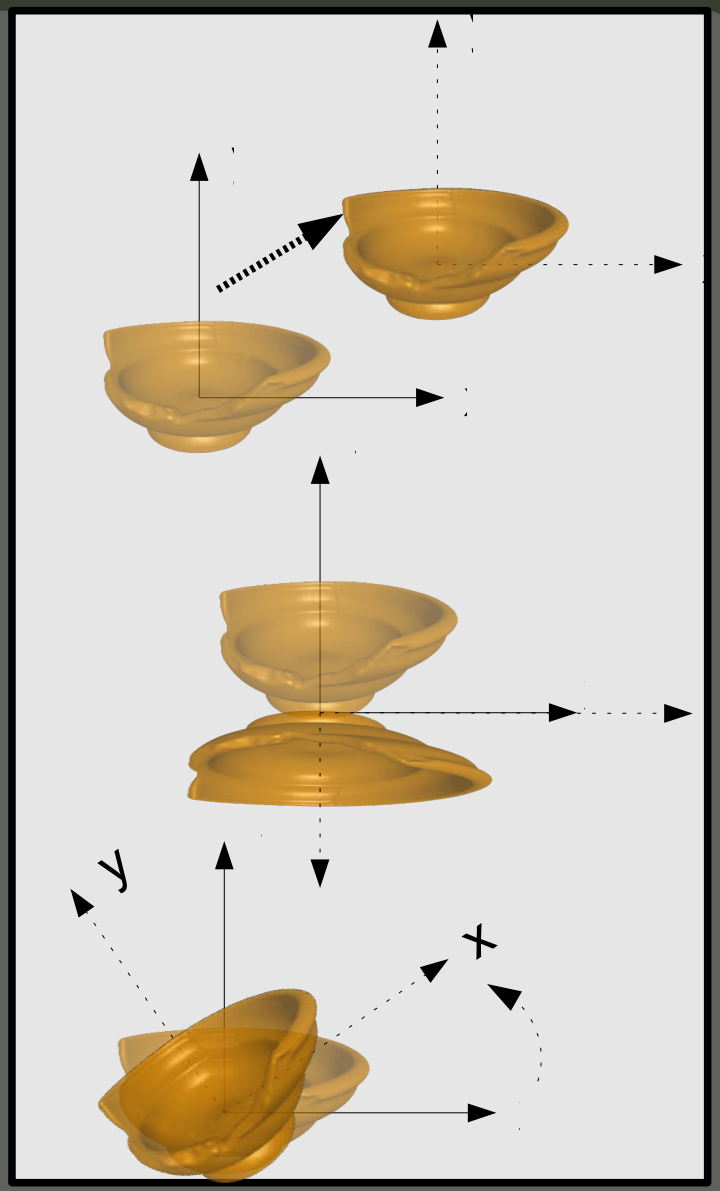
- Mise à l'échelle:

```
void glScale{fd}(T sx, T sy, T sz);
```

- Rotation

```
void glRotate{fd}(T a, T x, T y, T z);
```

  - a = angle en degré
  - [x,y,z] = axe de rotation



Transformations

- Charger la matrice identité:

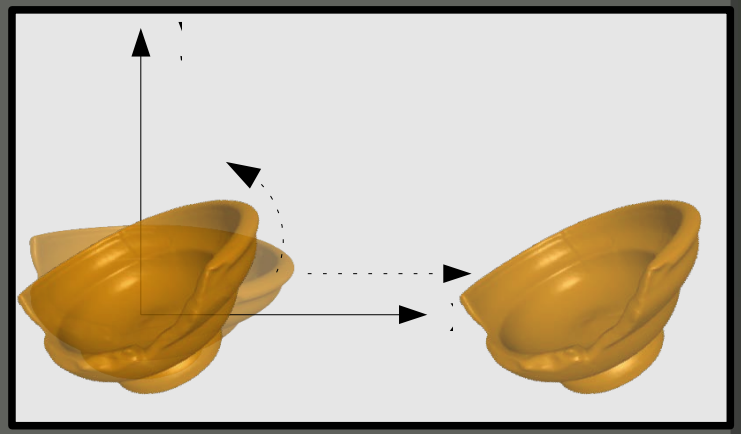
```
void glLoadIdentity();
```

- Transformations personnalisées (**column major!**)

```
void glMultMatrix{fd}(T *matrix);
void glLoadMatrix{fd}(T *matrix);
```

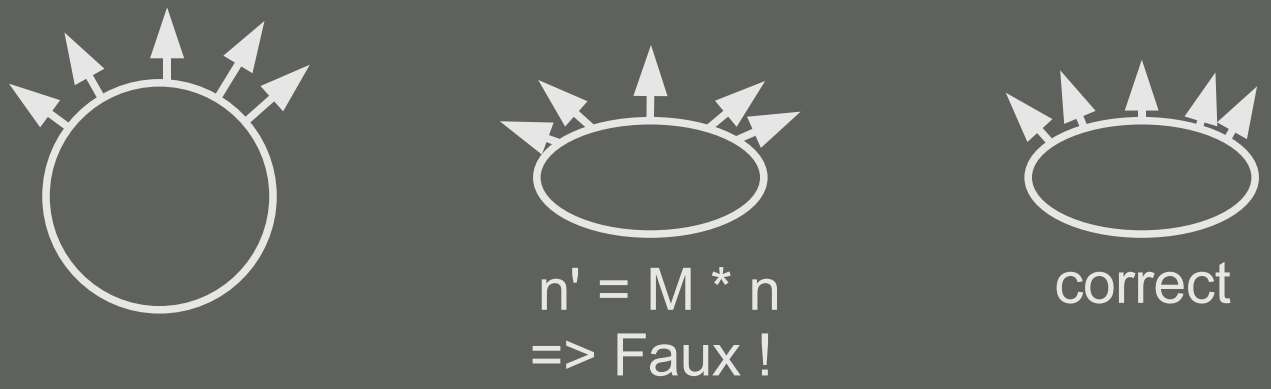
- glTranslate/Scale/Rotate/MultMatrix multiplient la matrice active:

```
glLoadIdentity ();
glMultMatrix   (T);
glMultMatrix   (R);
/* ... */
glVertex3fv    (v); /* v'=I*T*R*v */
```



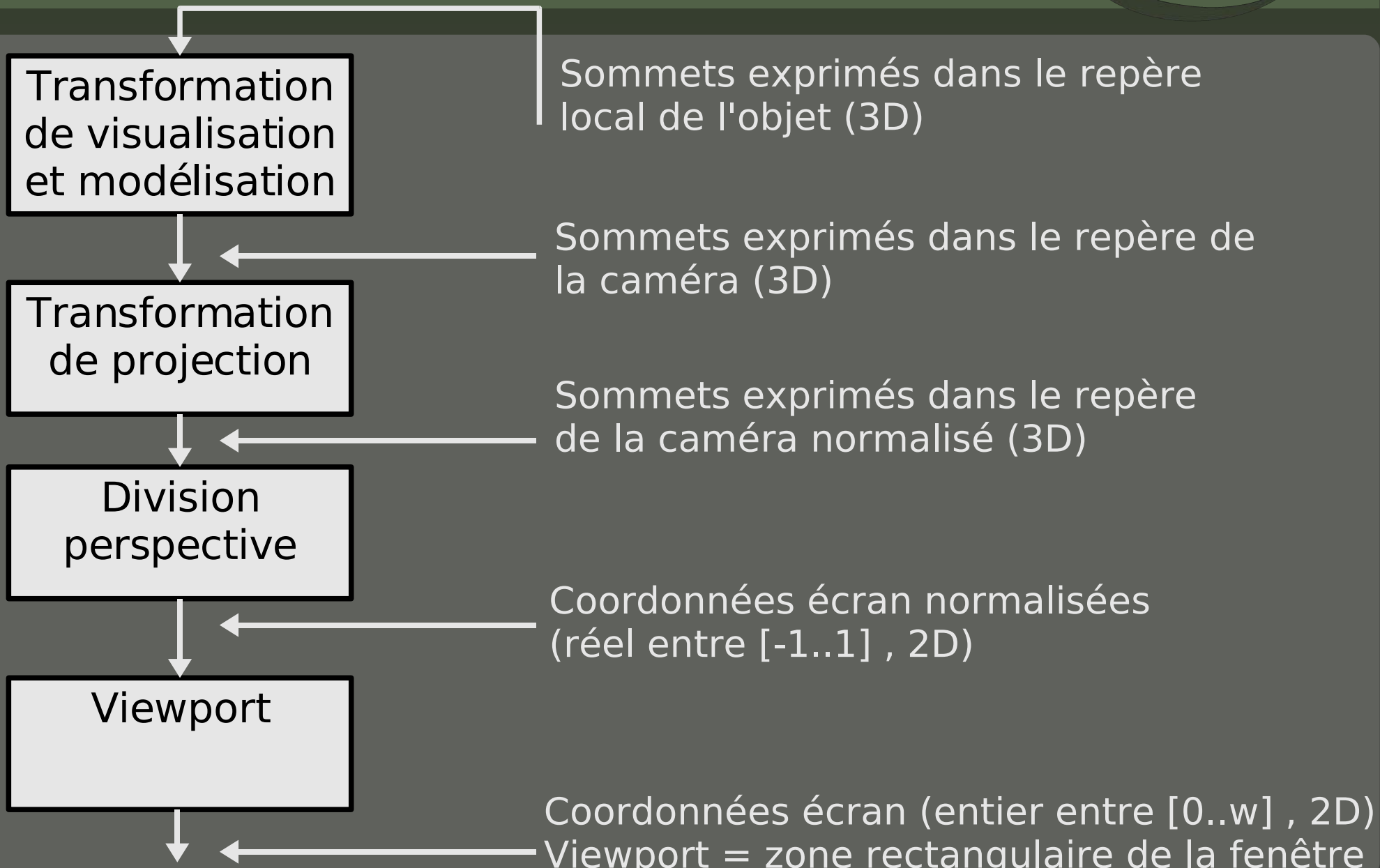
Transformations des normales

- Problème d'une mise à l'échelle non uniforme M:



- Utiliser la **Transposée de l'inverse**:
  - Soit M la matrice 3x3 de transformation courante:
    - $v' = M * v + T$
  - Alors :
    - $n' = (M^{-1})^t * n$

Vertex Pipeline



Gestion des matrices

- Plusieurs type de matrices

```
void glMatrixMode(GLenum mode); /*mode devient la matrice active*/
```

  - Modélisation/vue
  - Projection
  - Texture
  - …

```
/* mode = GL_MODELVIEW   */
/* mode = GL_PROJECTION  */
/* mode = GL_TEXTURE     */
```

- Une pile par type de matrice

```
void glPushMatrix ();
void glPopMatrix  ();
```

- Les opérations s'appliquent sur le sommet de la pile de la matrice active
  - Permet de hiérarchiser les transformations

Transformation de **visualisation**

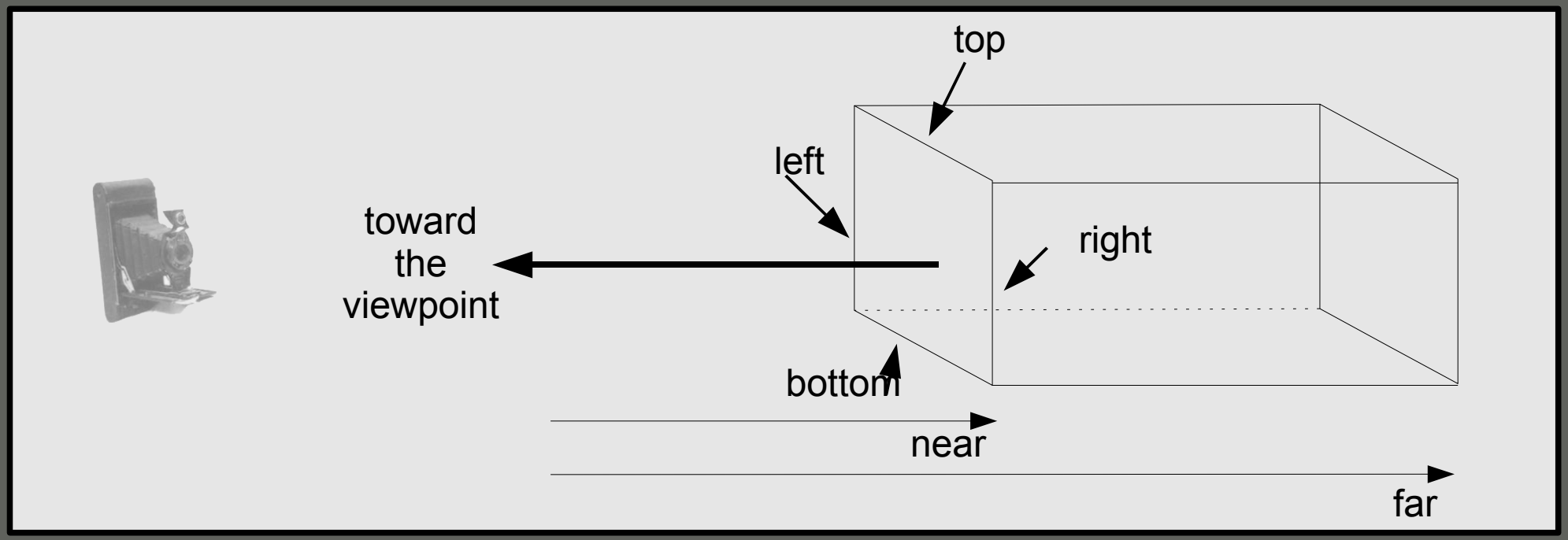
- Positionner et orienter la caméra

```
void gluLookAt(
GLdouble eX, GLdouble eY, GLdouble eZ,/* position */
GLdouble cX, GLdouble cY, GLdouble cZ,/* point de visé */
GLdouble uX, GLdouble uY, GLdouble uZ);  /* vecteur up */
```

  - Multiplier la matrice courante par une matrice de transformation du repère scène vers le repère caméra
    - En principe la matrice active doit être la matrice de modélisation
  - Deux styles de navigation:
    - Pilote
    - Polaire (tourner autour d'un objet)

Transformation de projection

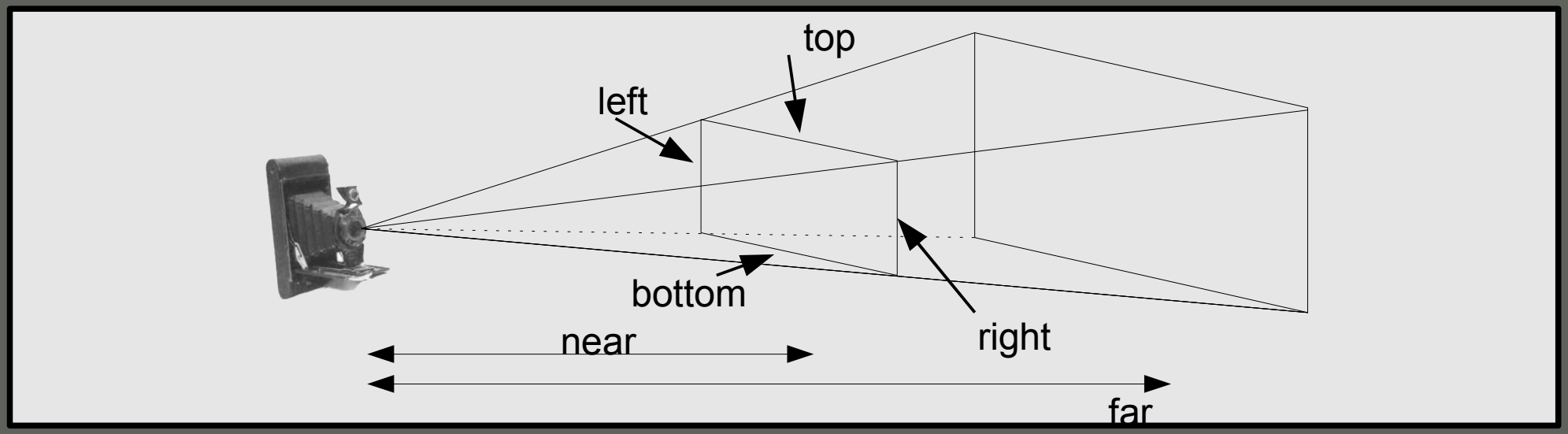
- Projection **orthogonale**




```
void glOrtho(
     GLdouble left,
     GLdouble right,
     GLdouble bottom,
     GLdouble top,
     GLdouble zNear,
     GLdouble zFar);
```

Transformation de projection

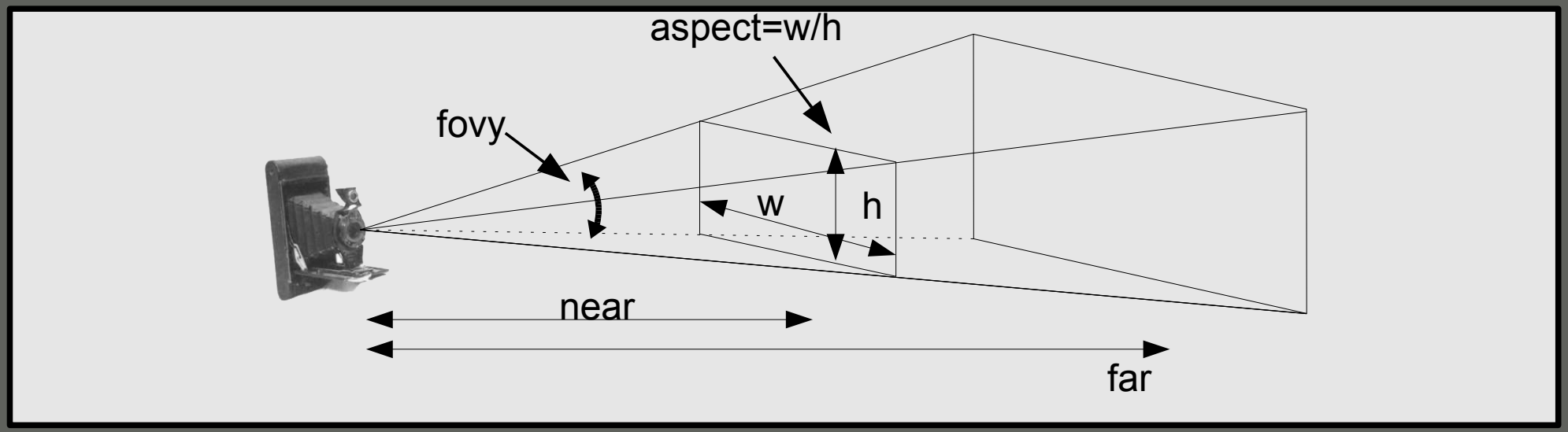
- Projection **perspective**




```
void glFrustum(
     GLdouble left,
     GLdouble right,
     GLdouble bottom,
     GLdouble top,
     GLdouble zNear,
     GLdouble zFar);
```

Transformation de projection

- projection **perspective**




```
void gluPerspective(
     GLdouble fovY,
     GLdouble aspect,
     GLdouble zNear,
     GLdouble zFar);
```

Transformation de Viewport

- Transformation des coordonnées normalisées en coordonnées écran

```
void glViewport(
     GLint     x,
     GLint     y,
     GLsizei   width,
     GLsizei   height);
```

  - xw = (xn + 1) (width / 2) + x
  - yw = (yn + 1) (height / 2) + y
- Peut s'exprimer sous forme matricielle

```
/* /!\ Column major !!!! */
float v[16] = { width*0.5f,   0,                0,     0,
                0,            height*0.5f,      0,     0,
                0,            0,                0.5f,  0,
                x+0.5f*width, y+0.5f*height,    0.5f,  1};
```

Simulation de l'éclairage

- Simule la façon dont l'objet “reflète” la lumière.
  - Dépend des propriétés:
    - de la surface
    - des sources lumineuses
    - de l'environnement

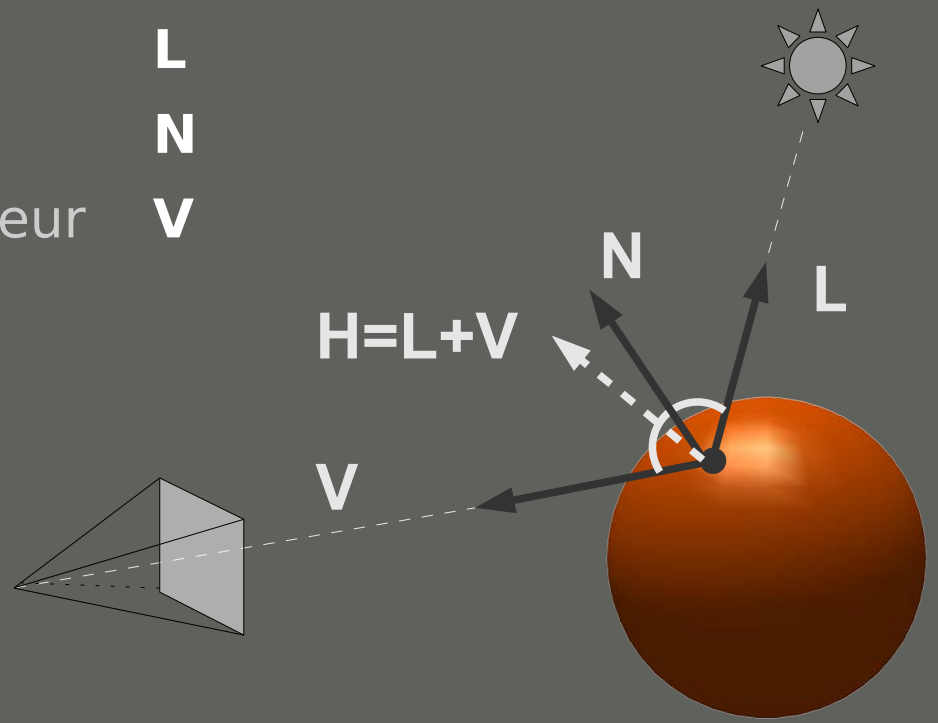
- Problématique
  - Quelle est l'intensité lumineuse reçue par chaque pixel ?
  - i.e. Quelle est l'intensité lumineuse réfléchie par chaque point de la scène en direction de la caméra?

Simulation de l'éclairage

- Problème complexe car
  - un object éclairé devient une source lumineuse... => éclairement en 1 point:
    - éclairage **direct**/local (énergie émise par les sources lumineuses)
    - éclairage **indirect**/global (énergie émise par les “objets” de la scène)
- En OpenGL, simplification de la problématique
  - Eclairage indirect = une constante
  - Eclairage direct = modèle de **Blinn-Phong**
  - Pas de source étendue / d'ombre portée

Modèle de Blinn-Phong

- Fonction de réflectance empirique
- Eclairement calculé aux sommets puis interpolé
  - Dépend de la:
    - direction de la lumière **L**
    - normale à la surface **N**
    - direction de l'observateur **V**



Modèle de Blinn-Phong

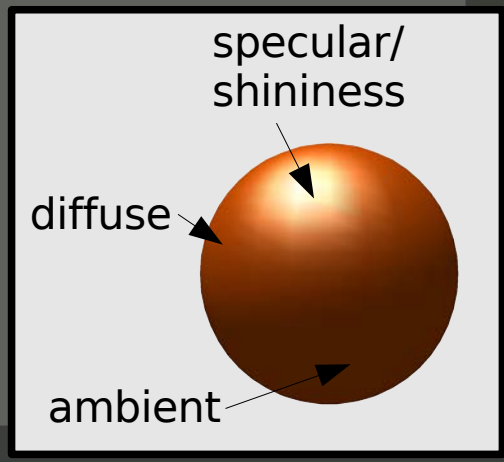
- Sépération diffus + spéculaire
  - Composante diffuse:
    - La lumière est réfléchie uniformément dans toutes les directions (indépendance à la position de l'observateur)
    - Dépend de la d'incidence
    - Intensité reçue/émise modulée par **N.L** (<=> cos de l'angle entre N et L)
  - Composante spéculaire (Blinn)
    - Utilisation du vecteur H = (L+V) / || L + V ||
    - Intensité reçue/emise modulée par $\mathbf{(N.H)^b}$ (<=> $\cos^b$ de l'angle entre N et H)
      b = coefficient de brillance (shininess)

Les matériaux

- Définition des propriétés d'un matériau par face

```
void glMaterial{fd}v(
     GLenum face, GLenum propriete, T* valeurs);
void glMaterial{fd}(
     GLenum face, GLenum propriete, T valeur);
```

  - propriétés
    - GL_AMBIENT: couleur des zones non éclairées directement
    - GL_DIFFUSE: couleur “de base”
    - GL_SPECULAR: couleur des reflets spéculaires
    - GL_EMMISSION: couleur émise par le matériau
    - GL_SHININESS: coefficient de brillance [0, 128]
  - face
    - GL_FRONT, GL_BACK, GL_FRONT_AND_BACK



Les matériaux

- Matériaux et couleurs des sommets
  - Utiliser la couleur des sommets comme une des composantes du matériau

```
/*   Activer/Desactiver l'état GL_COLOR_MATERIAL=
     glEnable/Disable(GL_COLOR_MATERIAL); */

void glColorMaterial  (GLenum face, GLenum mode);
```

    - mode = GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMMISSION, **GL_AMBIENT_AND_DIFFUSE**

  - Avantages
    - La commande glColor est plus rapide que glMaterial
    - Permet de faire varier un des composantes par sommet

Les lumières

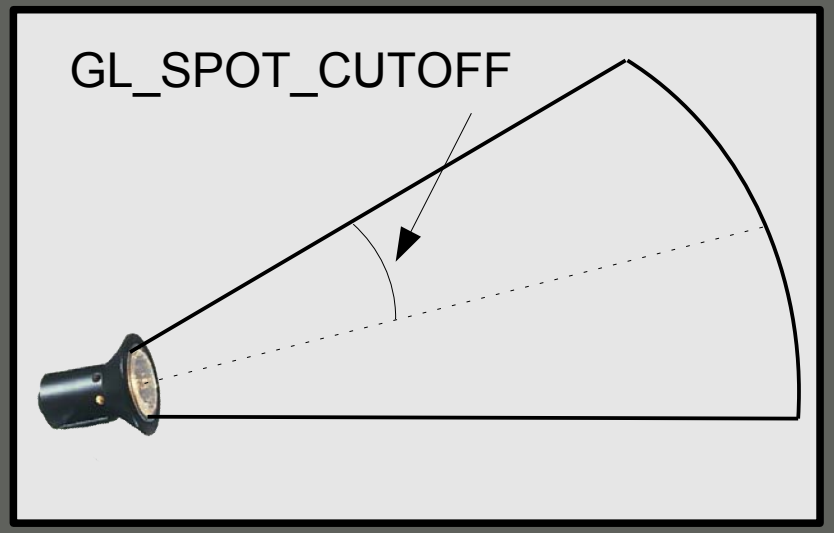
- Définition des propriétés d'une lumière

```
void glLight{if}v(
     GLenum light, GLenum propriete, T* valeurs);
void glLight{if}(
     GLenum light, GLenum propriete, T valeur);
```

  - light = GL_LIGHTi avec i < GL_MAX_LIGHTS.
- Composante de la lumière
  - GL_DIFFUSE, GL_AMBIENT, GL_SPECULAR:
    - ex: ambient = (ambient Materiau) * (ambient Lumière)
- Position de la lumière
  - GL_POSITION: contrôle la position et le type de lumière
    - Source directionnelle (w = 0)
    - Source omni-directionnelle (w ≠ 0)

Les lumières

- Définition d'un cône lumineux
  - Direction:
    GL_SPOT_DIRECTION
  - Angle d'ouverture :
    GL_SPOT_CUTOFF
  - Fonction d'atténuation
    GL_SPOT_EXPONENT



- Atténuation de la source lumineuse
  - Variation de l'intensité en fonction de la distance
    - ke GL_CONSTANT_ATTENUATION
    - kl GL_LINEAR_ATTENUATION
    - kq GL_QUADRATIC_ATTENUATION
    - d distance par rapport à la lumière

$$f = \frac{1}{k_e + d\,k_l + d^2 k_q}$$

Gestion de l'éclairage

- Activation de l'éclairage

```
glEnable/Disable(GL_LIGHTING);  /* activation de l'éclairage */
glEnable/Disable(GL_LIGHTi);/* activation de la lumière i
                                avec i< GL_MAX_LIGHTS */
```

- Lumière ambiente globale

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, GLfloat *colors);
```

- Eclairage sur une/deux faces

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GLboolean on_off);
```

- Ajout de la composante spéculaire après l'application des textures

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
  GL_SEPARATE_SPECULAR_COLOR);
```

- Point de vue local/infini

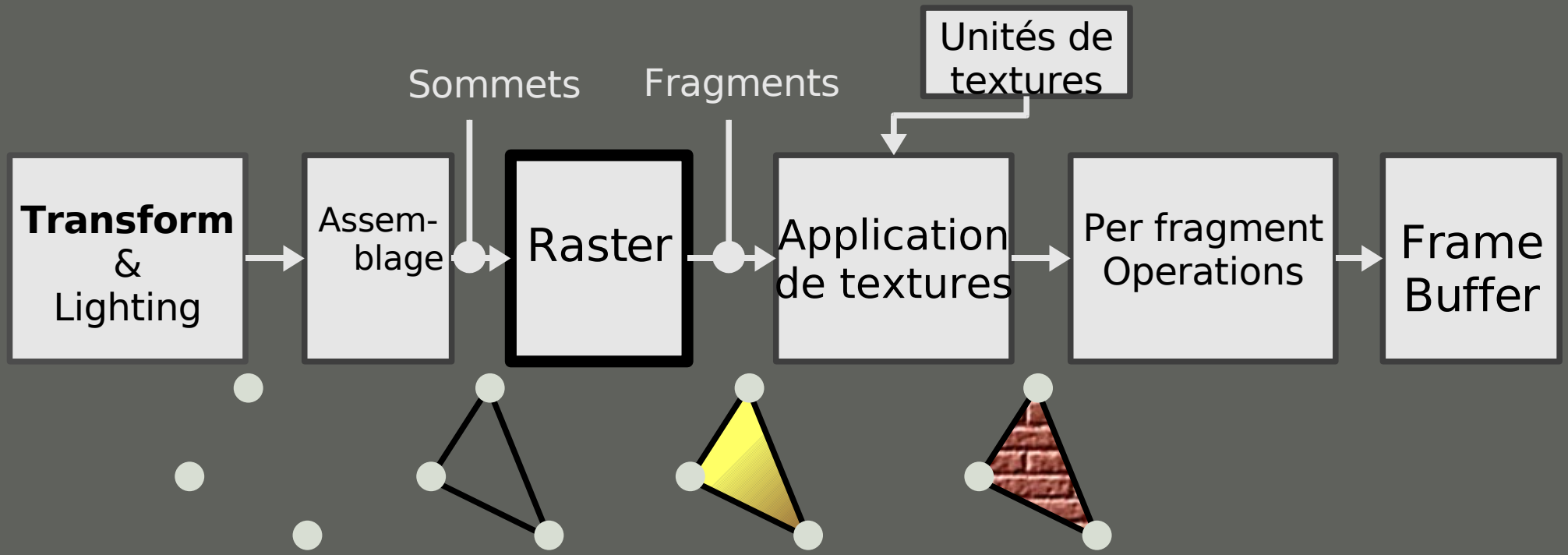
```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GLboolean on_off);
```

Eclairage par sommet

$$\text{couleur du sommet} = \text{emission}_{mat} + \text{ambient}_{lightmodel} * \text{ambient}_{mat} + \sum_i (\text{attenuation}(d))_i * (\text{effet projecteur})_i * [\ \text{ambient}_{light} * \text{ambient}_{mat} + \max\{L.N,0\} * \text{diffuse}_{light} * \text{diffuse}_{mat} + (\max\{H.N,0\})^{shininess} * \text{specular}_{light} * \text{specular}_{mat}\ ]_i$$

- Simplifications
  - $\text{emmission}_{mat} = \text{ambient}_{light} = 0$
  - $\text{specular}_{light} = \text{diffuse}_{light}$
  - Utiliser un coefficient pour l'ambient et le diffuse
    - $\text{ambient}_{mat} = \text{couleur}_{mat} * \text{coeff}_{ambient}$
    - $\text{diffuse}_{mat} = \text{couleur}_{mat} * (1 - \text{coeff}_{ambient})$

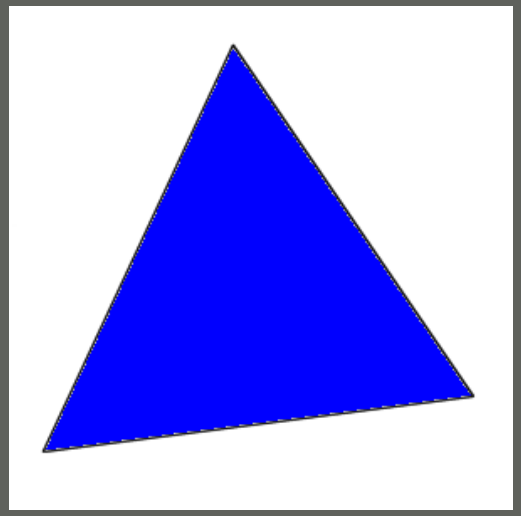
Rasterization

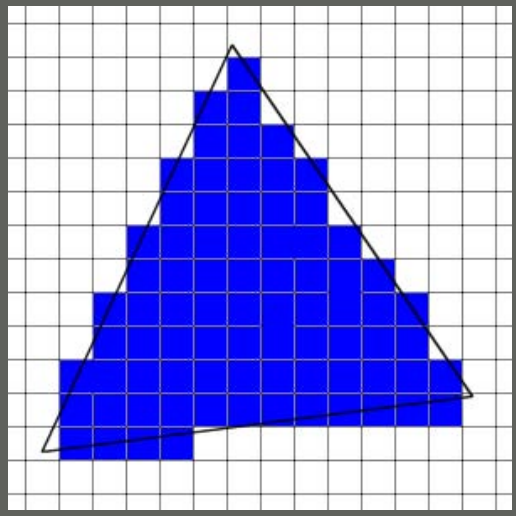


- La rasterization est la conversion en un ensemble de fragments, d'une primitive définie dans l'espace écran

Génération des fragments

- Un fragment coresspond à un pixel dans l'image finale
- Les informations de couleurs, coordonnées de textures, profondeur, attributs...etc sont assignées à chaque fragment
- Ces informations sont interpolées à partir des valeurs aux sommets





Rasterization

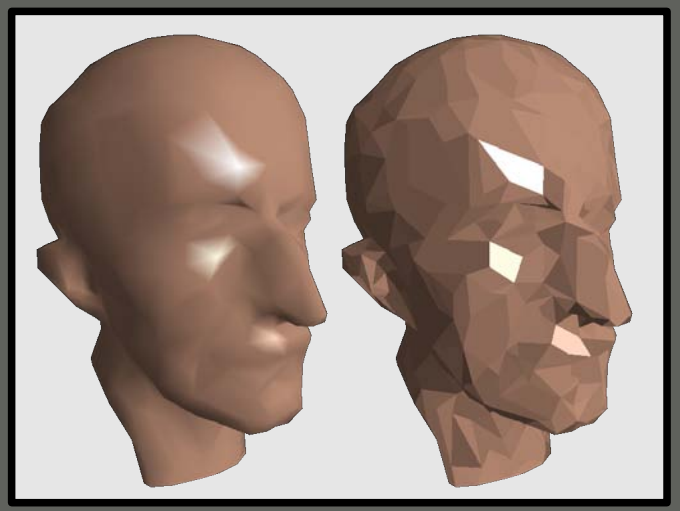
- Exemple

```
glBegin(GL_QUADS);
     glColor3f(1,0,0); glVertex2i(0,0);
     glColor3f(1,0,1); glVertex2i(1,0);
     glColor3f(1,1,1); glVertex2i(1,1);
     glColor3f(1,1,0); glVertex2i(0,1);
glEnd();
```

- Contrôle du modèle de dégradé

```
void glShadeModel(GLenum mode);
```

  - mode =
    - GL_SMOOTH: interpolation
    - GL_FLAT



Contrôle des primitives

- Primitive GL_POINTS
  - Taille des points en pixel:
```
void glPointSize(GLfloat size);
```
  - Point-sprite
```
glEnale/Disable(GL_POINT_SPRITE);
glTexEnvf(GL_POINT_SPRITE, GL_COORD_REPLACE, GL_TRUE);
```
    - Permet d'appliquer une texture 2D sur les points
      ex: système de particules
- Primitive GL_LINE*
  - Epaisseur
```
void glLineWidth(GLfloat width);
```
  - Pointillés
```
glEnable/Disable(GL_LINE_STIPPLE);

/* pattern = liste de 16bits (1bit<-> 1pixel * factor */
void glLineStipple(GLint factor, GLushort pattern);
```

Contrôle des primitives

- Contrôle des primitives polygones

```
/* mode = GL_POINT, GL_LINE, GL_FILL */
void glPolygonMode(GLenum face, GLenum mode);
```

- Lissage des primitives

```
glEnable/Disable(GL_{POINT,LINE,POLYGON}_SMOOTH);

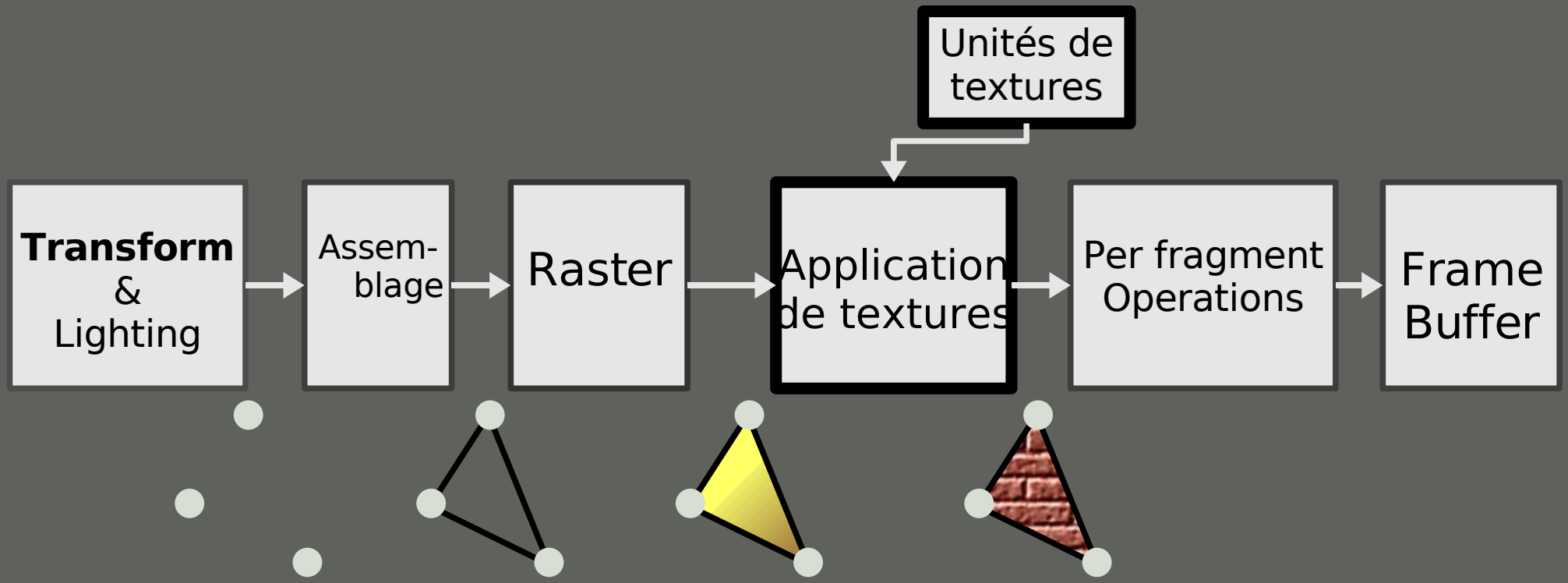
/* mode = GL_FASTEST, GL_NICEST, GL_DONT_CARE */
glHint(GL_{POINT,LINE,POLYGON}_SMOOTH_HINT, mode);
```

  - difficile à utiliser
- Multisampling antialiasing
  - activer le multisampling lors de la création du contexte de rendu puis:

```
glEnable/Disable(GL_MULTISAMPLE);
```

L'application des textures

- Pourquoi?
  - Simuler des matériaux complexes (bois, marbre, ...)
  - Augmente les détails sans augmenter la complexité géométrique



Gestion des textures

- 5 types de textures
  - GL_TEXTURE_1D
  - GL_TEXTURE_2D
  - GL_TEXTURE_RECTANGLE
  - GL_TEXTURE_3D
  - GL_TEXTURE_CUBE
- Activation d'un type de texture

```
glEnable/Disable(GL_TEXTURE_{1D 2D 3D CUBE RECTANGLE);
```

- Gestion d'un object de texture

```
void        glGenTextures    (GLsizei n, GLuint *names);
void        glDeleteTextures (GLsizei n, GLuint *names);
GLboolean   glIsTexture      (GLuint name);
void        glBindTexture    (GLenum target, GLuint name);
```

Textures & images

- Définir la texture courante depuis un tableau de texels en mémoire client

```
void glTexImage2D(GLenum target, GLint level, GLenum
  internalFormat, GLsizei w, GLsizei h, GLint border, GLenum
  format, GLenum type, const GLvoid *texels);
```

  - Formats : GL_RGB, GL_RGBA, GL_ALPHA...
  - Formats interne: GL_R3_G3_B2, GL_RGB4, GL_ALPHA...
  - Type de données: GL_FLOAT, GL_UNSIGNED_BYTE, GL_INT, GL_HALF_FLOAT_ARB...
  - Target: GL_TEXTURE_2D, GL_TEXTURE_RECT
    - GL_TEXTURE_2D: nécessite une taille puissance de 2

```
void gluScaleImage(GLint format, GLsizei wIn, GLsizei hIn,
  GLenum typeIn, const GLvoid *texelsIn, GLsizei wOut,
  GLsizei hOut, GLenum typeOut, GLvoid *texelsOut);
```

Textures & images

- Mise à jour d'une partie de la texture

```
void glTexSubImage2D(GLenum target, GLint level, GLint
  xoffset, GLint yoffset, GLsizei w, GLsizei h, GLenum
  format, GLenum type, const GLvoid *texels);
```

- Copie du framebuffer dans une texture

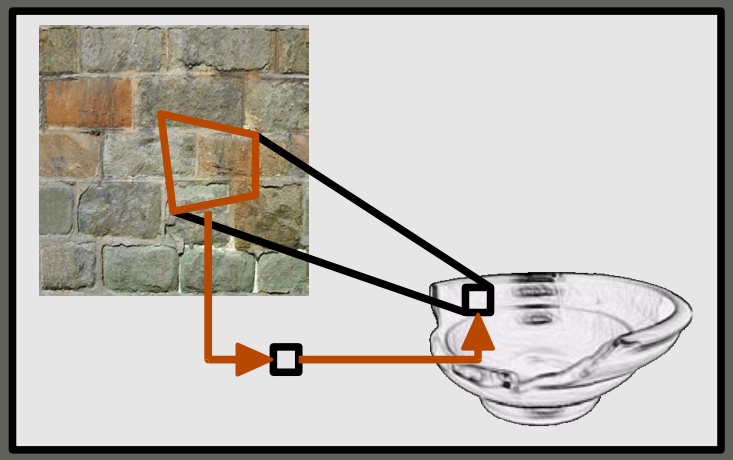
```
void glCopyTexImage2D(GLenum target, GLint level, GLint
  internalFormat, GLint xoffset, GLint yoffset, GLsizei w,
  GLsizei h, GLint border);
```

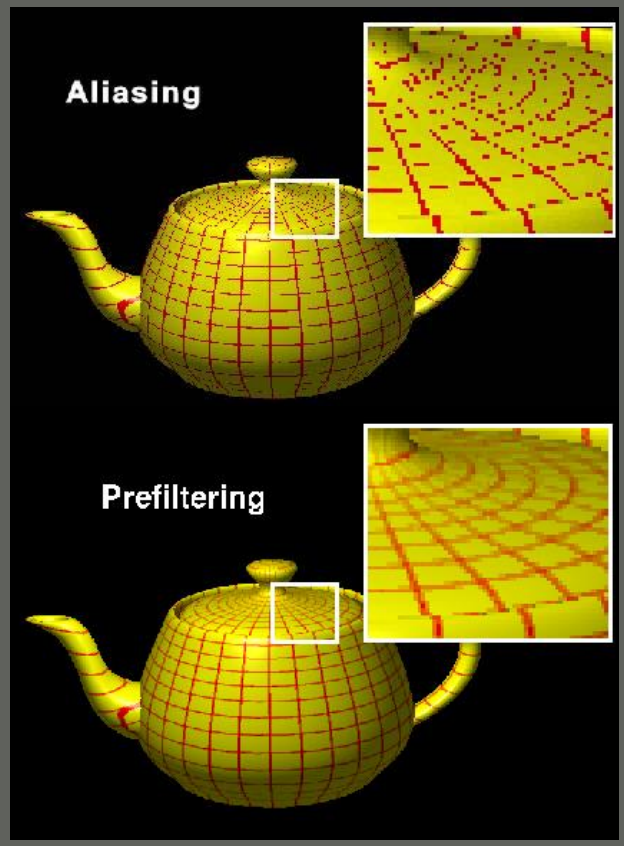
- Mise à jour à partir du framebuffer

```
void glCopyTexSubImage2D(GLenum target, GLint level, GLint
  internalFormat, GLint xoffset, GLint yoffset, GLsizei w,
  GLsizei h);
```

Filtrage

- Utilité?
  - Gérer la différence d'échantillonage texture/écran
- Théorie
  - Projection du pixel sur la texture & integration de l'aire: **Cout élevé**



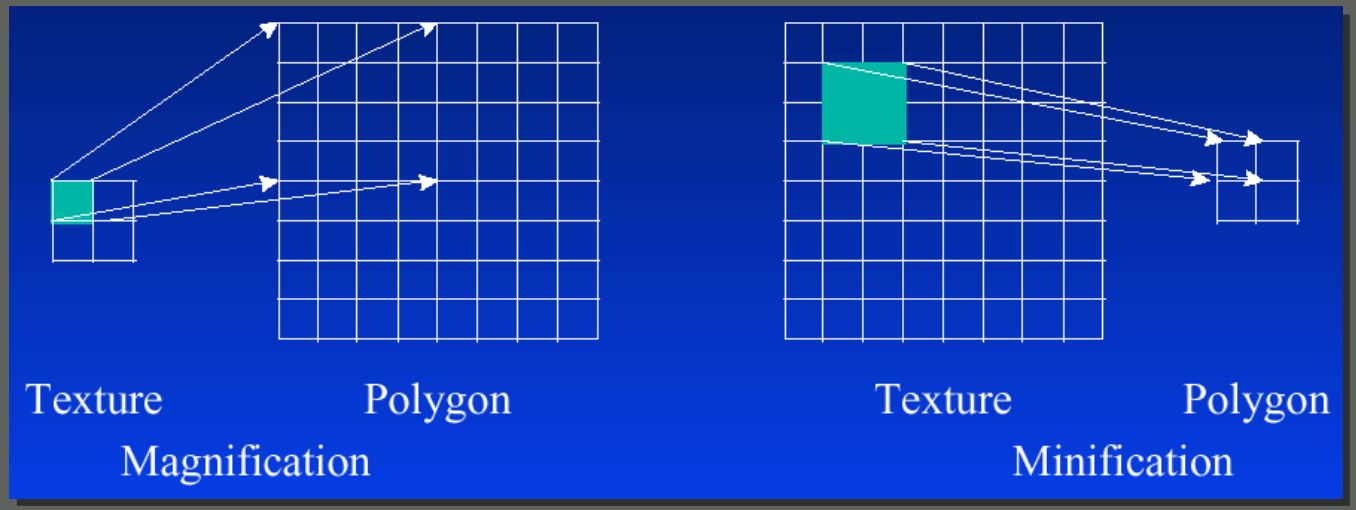


- En pratique: le filtrage OpenGL est plus simple

Filtrage

```
void glTexParameteri(GLenum target, GLenum pname, GLenum param
```

- pname
  - GL_TEXTURE_MAG_FILTER (agrandissement)
  - GL_TEXTURE_MIN_FILTER (réduction)
- param
  - GL_NEAREST
  - GL_LINEAR

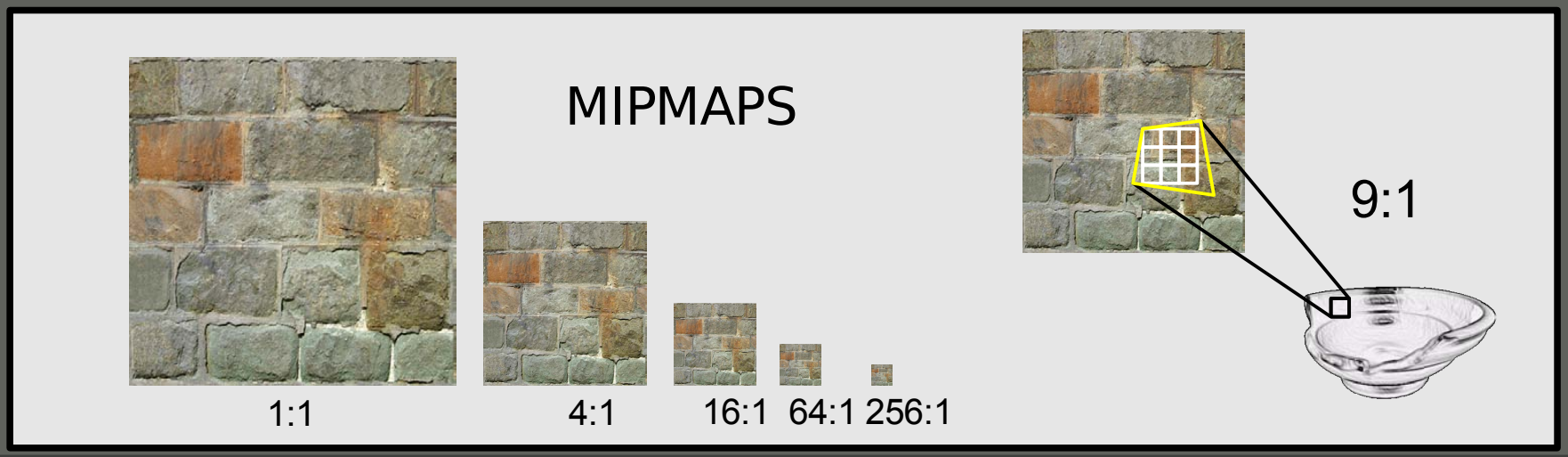


Mipmaps

- Textures à différentes résolutions
- Utilisées pour accélérer et améliorer le filtrage
- Génération automatique

```
void gluBuild2DMipmaps(GLenum target, GLint internalFormat,
  GLsizei w, GLsizei, GLenum format, GLenum type, const
  GLvoid *texels);
```

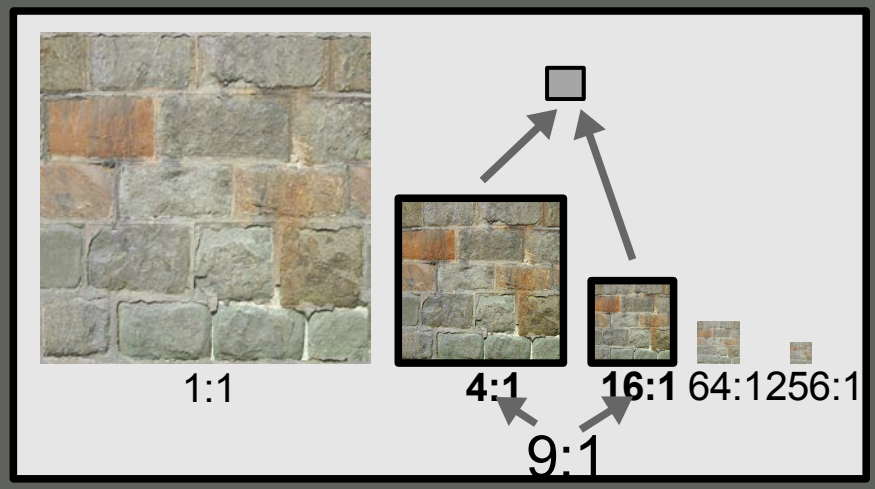
- Gestion automatique
  - niveau choisi par OpenGL en fonction de la taille de l'objet (en pixels)



Filtrage

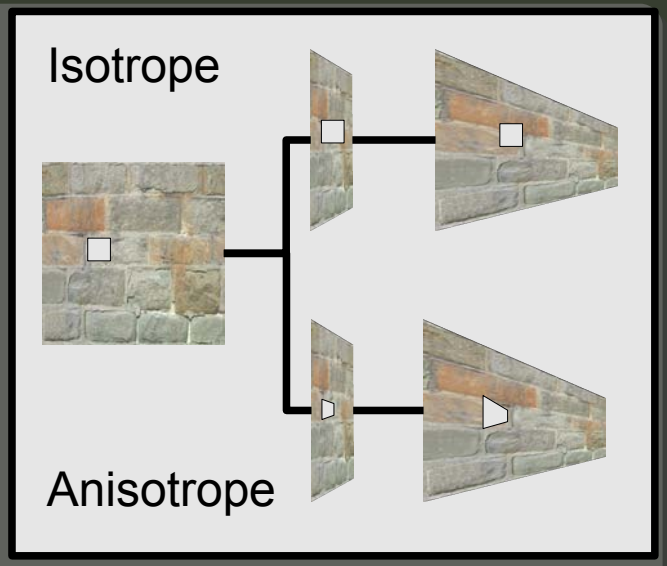
```
void glTexParameteri(GLenum target, GLenum pname, GLenum param
```

- Nouvelle fonction de Minification
  - choix du Mipmap le plus proche GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST
  - Interpolation à partir des mipmaps les plus proches GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR



Filtrage anisotrope

- La projection d'un pixel dans l'espace de la texture:
  - est assimilée à un carré (isotrope)
  - n'est pas un carré (anisotrope)
- ≠ méthodes de filtrage anisotrope




- ```
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT,
    maxAniso); /* maxAniso >= 1.0f */
  ```
  - Mecanisme général qui définit le degré maximum d'anisotropie
  - Degré d'anisotropie définit par objet texture
  - Indépendant des fonctions de minification & magnification
  ```

Filtrage d'une texture 2D: exemple

```
glTexImage2D(
     GL_TEXTURE_2D,       /* target */
     0,                   /* level */
     GL_RGBA,             /* internalFormat */
     256,                 /* imageWidth */
     256,                 /* imageHeight */
     0,                   /* border */
     GL_RGBA,             /* format */
     GL_UNSIGNED_BYTE,    /* type */
     texels);             /* texels */

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
  GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAX_ANISOTROPY_EXT, 16.f);
```

Coordonnées de texture

- Correspondance entre coordonnées et texels
  - Spécifiées explicitement par sommet

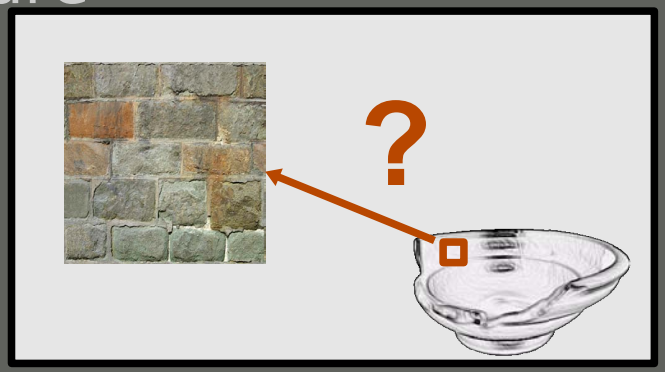
```
void glTexCoord{1234}{sifd}{0v}
     ({T *coords|T coord1, ... T CoordN});
```

  - Générées automatiquement

```
void glTexGen{ifd}(GLenum coord, GLenum pname, T param);
```

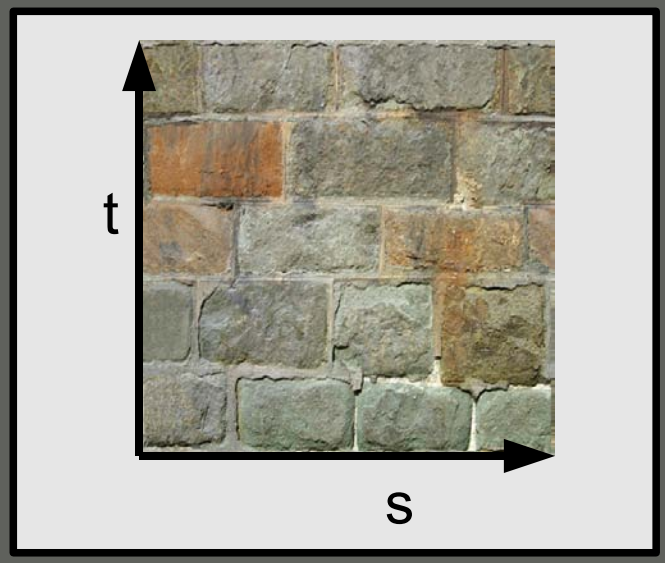
  - Jusqu'a 4 composantes:
    - s, t, r et q (1D, 2D, 3D et 3D homogène)
  - Transformées par la matrice de texture

```
glMatrixMode(GL_TEXTURE);
```



Coordonnées de texture

- Espace de la texture
  - GL_TEXTURE_{123}D
    - compris entre 0 et 1
  - GL_TEXTURE_RECTANGLE
    - s [0, width]; t [0,height]



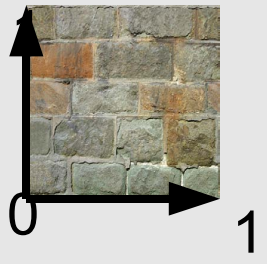
- Plaquer une texture 2D
  - calculer les coordonnées de texture par sommet
    - v (x,y,z) -> (s,t) = paramétrisation de la surface
  - ex: pour une sphère
    - Transformer v en coordonnées sphériques (phi, theta, r)
    - Prendre s=phi et t=theta
- Plaquer une texture 3D: (s, t, r) = (x, y, z)

Contrôle du pavage: Texture2D

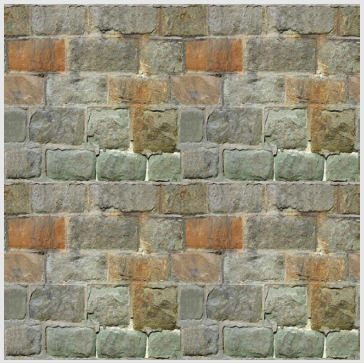
- Gestion des coordonnées de textures hors de [0, 1]

```
void glTexParameteri(GLenum target, GLenum pname, GLenum param);
```

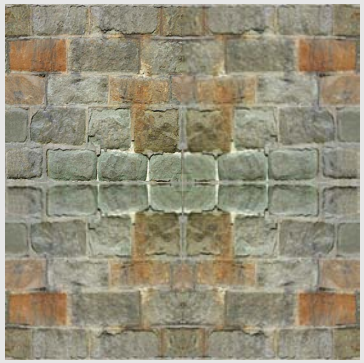
  - pname = GL_TEXTURE_WRAP_{S T}
  - param
    - GL_CLAMP: s = (s<0 ? 0 : (s > 1 ? 1 : s))
    - GL_REPEAT: répétition
    - GL_MIRRORED_REPEAT



GL_CLAMP



GL_REPEAT



GL_MIRRORED_REPEAT

Couleur finale

- La couleur du fragment est mélangée avec la couleur de la texture
- Contrôle du mélange

```
/*
     target doit être GL_TEXTURE_ENV
     pname = GL_TEXTURE_ENV_MODE
*/
void glTexEnv{iv}[v](GLenum target, GLenum pname, T [*]param);
```

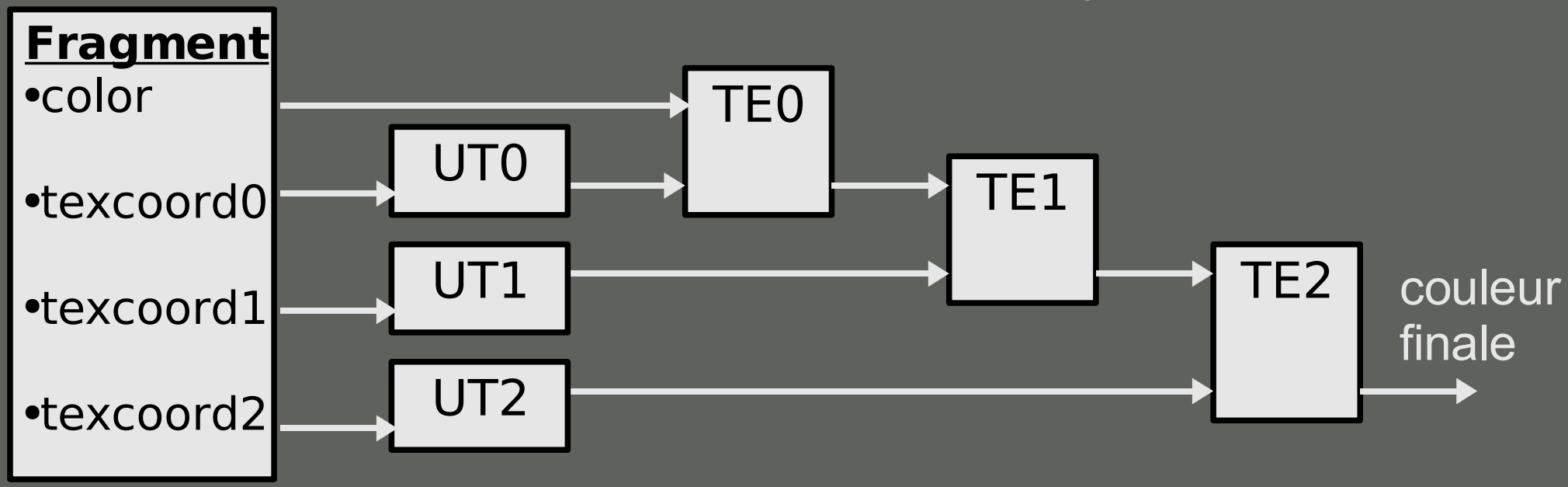
  - param
    - GL_MODULATE: multiplication
    - GL_REPLACE: utilisation seule de la texture
    - GL_DECAL: mélange en fonction de l'alpha de la texture
    - GL_ADD: addition
    - GL_BLEND: interpolation avec GL_TEXTURE_ENV_COLOR

Multi-texturing

- OpenGL supporte plusieurs unités de texture

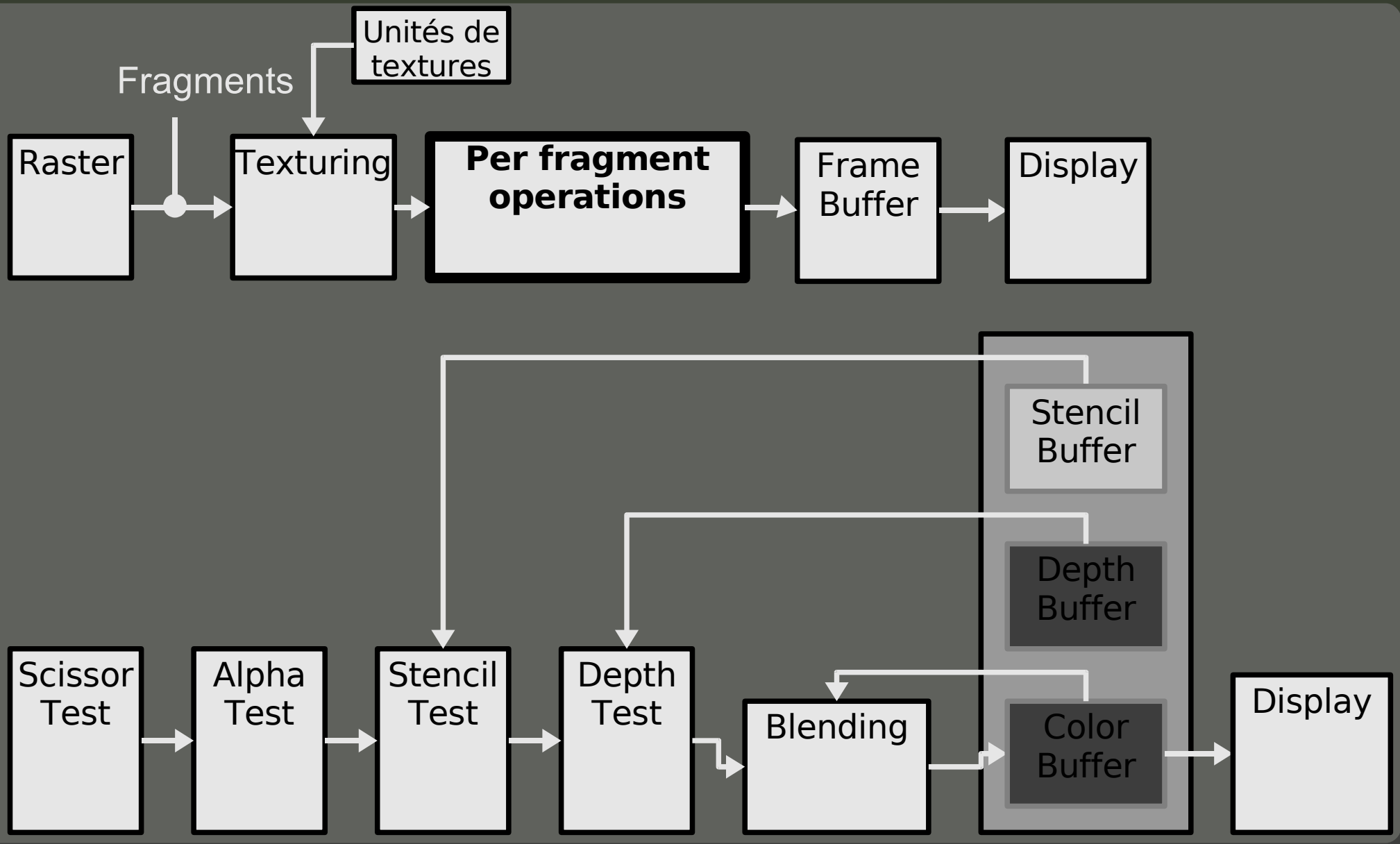
```
void glActiveTexture        (GLenum texUnit);
void glClientActiveTexture  (GLenum texUnit);
```

- Les commandes de gestion de texture et de coordonnées de texture n'affectent que l'unité active



UTi = Unité de Texture i
TEi = Texture Environnement i (GL_REPLACE, GL_MODULATE, ...)

Fragment Pipeline



Test de scission (scissor) et d'alpha

- Test de **scission**

```
glEnable/Disable(GL_SCISSOR_TEST);
```

  - Mise à jour d'une portion du viewport

```
void glScissor(GLint x, GLint y,
  GLsizei width, GLsizei height);
```

- Test d'**alpha**

```
glEnable/Disable(GL_ALPHA_TEST);
```

  - Basé sur la composante de transparence (alpha A)
  - Utilise une fonction de comparaison et une valeur de référence

```
void glAlphaFunc(GLenum func, GLclampf ref);
```

    - func = GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS

Test de pochoir (stencil)

- Test de **stencil** / Stencil buffer (<=> masque)

```
glEnable/Disable(GL_STENCIL_TEST);
```

  - Initialiser/nettoyer le stencil buffer

```
void glClearStencil(GLint s);
glClear(GL_STENCIL_BUFFER_BIT);
```

  - Fonctions/Operations de stencil

```
void glStencilFunc(GLenum func, GLint ref, GLuint mask);
void glStencilOp  (GLenum fail, GLenum zfail, GLenum zpass);
```

    - fail/zfail/zpass = GL_INCR, GL_DECR, GL_INVERT, GL_KEEP, GL_ZERO, GL_REPLACE
      Action a effectuer quand le test de Stencil échoue / le test de profondeur échoue / le test de profondeur est accepté

  - Masquer le stencil buffer

```
void glStencilMask(GLuint mask);
```

Mélange fragment/pixel: **blending**

```
void glEnable/Disable(GL_BLEND);
```

- Mélange des fragments avec les pixels du color buffer
- Le mélange est contrôlé par une fonction de mélange

```
void glBlendFunc(GLenum factor_src, GLenum factor_dest);
```

  - $C_{pixel} = C_{frag} * factor_{src} + C_{pixel} * factor_{dst}$
  - Fonctions de mélange disponibles
    - GL_ZERO, GL_ONE
    - GL_{SRC DST CONSTANT}_{ALPHA COLOR}
    - GL_ONE_MINUS_{SRC DST CONSTANT}_{ALPHA COLOR}
- Définition de la couleur constante de mélange

```
void glBlendColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a);
```

Blending avancé

- Contrôle de l'équation de mélange

```
void glBlendEquation(GLenum equation);
/* equation =
GL_FUNC_ADD:              C_pixel = C_frag * factor_src + C_pixel * factor_dst
GL_FUNC_SUBTRACT:         C_pixel = C_frag * factor_src - C_pixel * factor_dst
GL_FUC_REVERSE_SUBTRACT: C_pixel =-C_frag * factor_src + C_pixel * factor_dst
GL_MIN:                   C_pixel = min(C_frag, C_pixel)
GL_MAX:                   C_pixel = max(C_frag, C_pixel) */
```

- Séparation RGB & Alpha

```
void glBlendFuncSeparate(GLenum rgb_src,   GLenum rgb_dst,
                         GLenum alpha_src, GLenum alpha_dst);

void glBlendEquationSeparate(GLenum rgb_equation,
                             GLenum aplha_equation);
```

Blending: quelques exemples

- Alpha Blending

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
/* C_pixel = C_frag * C_frag.alpha + C_pixel * (1 - C_frag.alpha) */
```

  - Algorithme
    - Tracer les objets opaques
    - Activer le blending
    - Tracer les objets transparents du plus profond au moins profond

- Addition

```
glBlendFunc(GL_ONE, GL_ONE);
/* C_pixel = C_frag * 1 + C_pixel * 1 <=>  C_pixel = C_frag + C_pixel */
```

  - Utilisé pour les phénomènes lumineux (lens flare, lumières…)

Brouillard (fog)

```
glEnable/Disable(GL_FOG);
```

- Définition des propriétés du brouillard

```
void glFog{if}[v](GLenum pname, Type [*]param);
```

  - $C = f * C_{fragment} + (1 - f) * C_{fog}$
  - Intensité de f: pname=GL_FOG_MODE
    - Dépend de la profondeur du fragment.
    - Peut varier linéairement (GL_LINEAR), de façon exponentielle (GL_EXP) ou exponentielle au carré (GL_EXP2)
  - Contrôler la fonction f: pname = GL_FOG_START / GL_FOG_END / GL_FOG_DENSITY
  - Couleur C_{fog}: pname = GL_FOG_COLOR

Test de profondeur

- Test de profondeur / Z-Buffer (<=> depth buffer)

```
glEnable/Disable(GL_DEPTH_TEST);
```

  - Pronfondeur: distance par rapport à l'oeil normalisée entre 0 et 1 (0 = plan Near, 1 = plan Far)
  - Initialiser/nettoyer le Z-Buffer

```
void glClearDepth(GLclampd depth);
glClear(GL_DEPTH_BUFFER_BIT);
```

  - Fonction de test de profondeur

```
void glDepthFunc(GLenum func);
/*if(func(fragment.z, depthBuffer(x,y))){
     depthBuffer(x,y) = fragment.z;
     colorBuffer(x,y) = fragment.color;
}*/
```

  - Masquer le Z-Buffer

```
void glDepthMask  (GLboolean   mask);
```

Color Buffer

- Le color buffer est un sous ensemble du frame buffer
  - Frame buffer =
    - color buffer + depth buffer + stencil buffer + ...
- Précision du Color Buffer OGL limitée à 8 bits
- Initialiser/nettoyer le color buffer

```
void glClearColor(GLclampf  red,
                  GLclampf  green,
                  GLclampf  blue,
                  GLclampf  alpha);
glClear(GL_COLOR_BUFFER_BIT);
```

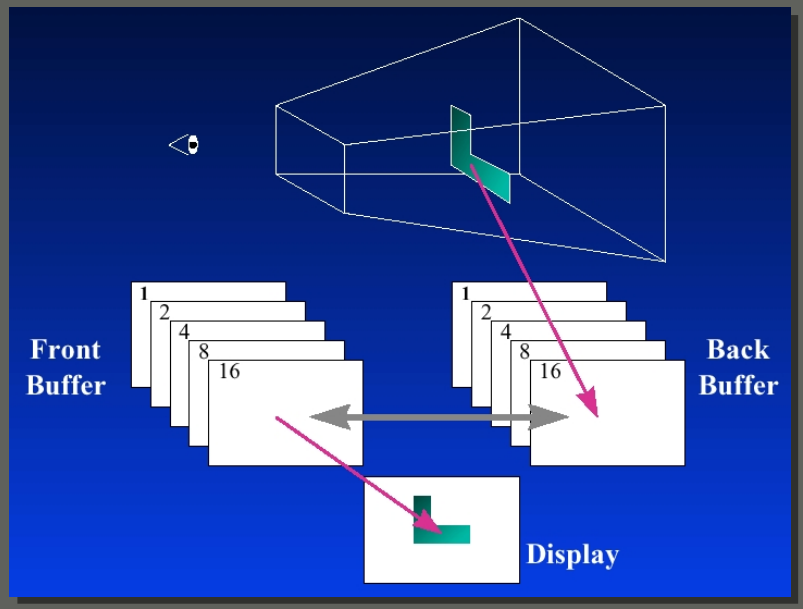
- Masquer le color buffer

```
void glColorMask  (GLboolean   red,
                   GLboolean   green,
                   GLboolean   blue,
                   GLboolean   alpha);
```

Double buffering

- Séparation du color buffer
  - Front buffer: GL_FRONT
  - Back buffer: GL_BACK

```
void glReadBuffer(GLenum mode);
void glDrawBuffer(GLenum mode);
```



- Principe
  - Affichage du front buffer (ex: 100Hz)
  - Tracé dans le back buffer (ex: 30Hz)
  - Echange des deux buffers

Images et Frame buffer

- Positionner la trame active

```
void glRasterPos{234}[v]{sifd}(TYPE [*]coords);
```

  - Transformé par les matrices de modélisation et de projection

- Tracer une image (à la position courante)

```
void glDrawPixels(GLsizei w, GLsizei h, GLenum format, GLenum
  type, const GLvoid *pixels);
```

- Lire dans le frame buffer

```
void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei
  height, GLenum format, GLenum type, GLvoid *pixels);
```

- Copie au sein d'un buffer

```
void glCopyPixels(GLint x, GLint y, GLsizei width,
  GLsizei height, GLenum type); /* type = GL_COLOR, GL_STENCIL,
                                              GL_DEPTH */
```

Pixel Buffer Object

- Extensions de l'interface proposée par les Vertex Buffer Object
  - Les buffers ne contiennent plus des données sur les sommets mais sur les pixels
  - 2 nouveaux types de buffer
    - GL_PIXEL_PACK_BUFFER_ARB: écriture de pixels
    - GL_PIXEL_UNPACK_BUFFER_ARB: lecture de pixels
- Quelques exemple d'utilisation:
  - Mise à jour de texture en streaming
  - Dessiner des pixels en streaming
  - Lecture asynchrone du frame buffer
  - Rendre directement dans un Vertex Array

Pixel Buffer Object: exemple

```
const int   imagewidth       = 640;
const int   imageheight      = 480;
const int   imageSize        = imagewidth*imageheight*4;
GLubyte*    pboMemory        = NULL;
GLuint      imageBuffer;

glGenBuffers(1, &imageBuffer);
glBindBuffer(GL_PIXEL_PACK_BUFFER_ARB, imageBuffer);
glBufferData(GL_PIXEL_PACK_BUFFER_ARB, imageSize, NULL,
  GL_DYNAMIC_READ);
/* dessin de la scene dans le frame buffer */
glDrawBuffer(GL_BACK);
renderScene();
/* lecture des pixels du framebuffer (=ecriture dans le PBO) */
glReadPixels(0, 0, imagewidth, imageheight, GL_BGRA,
                      GL_UNSIGNED_BYTE, NULL);
/* Traitement de l'image */
glBindBuffer(GL_PIXEL_PACK_BUFFER_ARB, imageBuffer);
pboMemory = glMapBuffer(GL_PIXEL_PACK_BUFFER_ARB, GL_READ_ONLY);
processImage(pboMemory);
glUnmapBuffer(GL_PIXEL_PACK_BUFFER_ARB);

glDeleteBuffers(1, &imageBuffer);
```

Frame buffer Object

- Nouvelle interface pour dessiner vers des “destinations de rendu” autre que les tampons OGL
  - remplace les pbuffers
- Rendu effectué dans des images attachées aux differents tampons du frame buffer
  - Z-Buffer, color buffer, stencil buffer...etc
- Les images destinations peuvent être:
  - des images d'objets textures
  - des images d'objets render buffers (=objets qui encapsulent une image 2D)

Frame buffer Object

- Gestion d'un object “frame buffer”

```
void glGenFramebuffersEXT   (GLsizei n, GLuint *framebuffers);
void glDeleteFramebuffersEXT(GLsizei n, const GLuint *framebuffers);
void glBindFramebufferEXT   (GLenum target, GLuint framebuffer);
void glIsFramebufferEXT     (GLuint framebuffer);
```

  - target = GL_FRAMEBUFFER_EXT

- Attacher une image d'objet texture

```
void glFramebufferTexture{1 2}D(GLenum target, GLenum attachment,
  GLenum texTarget, GLuint texture, GLint level);
void glFramebufferTexture3D(GLenum target, GLenum attachment,
  GLenum texTarget, GLuint texture, GLint level, GLint zoffset);
```

  - attachment =
    - GL_COLOR_ATTACHMENTi_EXT (0<i<16)
    - GL_DEPTH_ATTACHMENT_EXT/GL_STENCIL_ATTACHMENT_EXT
  - texTarget = GL_TEXTURE{1D 2D 3D RECTANGULAR...}

Frame buffer Object

- Attacher une image d'objet render buffer

```
void glFramebufferRenderbufferEXT(GLenum target, GLenum
  attachment, GLenum renderbuffertarget, GLuint renderbuffer);
```

  - attachment = identique aux images d'objets textures
  - renderbuffertarget = GL_RENDERBUFFER_EXT
- Verifier le statut de l'objet “frame buffer” actif: non obligatoire mais fortement conseillé

```
GLenum glCheckFramebufferStatusEXT(GLenum target);
```

  - target = GL_FRAMEBUFFER_EXT
  - Valeur retournée: GL_FRAMEBUFFER_COMPLETE_EXT, GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT_EXT, GL_FRAMEBUFFER_UNSUPPORTED_EXT...

Render buffer

- Gestion d'un object render buffer

```
void glGenRenderbuffersEXT   (GLsizei n, GLuint *renderbuffers);
void glDeleteRenderbuffersEXT(GLsizei n, const GLuint *renderbuffers);
void glBindRenderbufferEXT   (GLenum target, GLuint renderbuffer);
void glIsRenderbufferEXT     (GLuint renderbuffer);
```

  - target = GL_RENDERBUFFER_EXT

- Définir le format et la taille de l'image du render buffer

```
void glRenderbufferStorageEXT(
     GLenum target,
     GLenum internalFormat,
     GLsizei width,
     GLsizei height);
```

Frame buffer Object: exemple

```
/* Données: color_tex - TEXTURE_2D objet texture de coleur
            depth_rb  - GL_DEPTH objet render buffer
            fb        - objet frame buffer */
/* Active le rendu vers une texture */
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb);
/* Attribue color_tex et depth_rb à l'objet frame buffer */
glFramebufferTexture2DEXT   (GL_FRAMEBUFFER_EXT,
                             GL_COLOR_ATTACHMENT0_EXT,
                             GL_TEXTURE_2D, color_tex, 0);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT,
                             GL_DEPTH_ATTACHMENT_EXT,
                             GL_RENDERBUFFER_EXT, depth_rb);
/* verifie le statut du frame buffer */
if(glCheckFramebufferStatus(GL_FRAMEBUFFER_EXT)!=
  GL_FRAMEBUFFER_COMPLETE_EXT){
     fprintf(stderr, "frame buffer non complet\n");
     exit(1);
}
renderScene(); /* dessine dans color_tex */
/* reactive le rendu dans le frame buffer OGL*/
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
/* utilise la texture color_tex*/
glBindTexture(GL_TEXTURE_2D, color_tex);
```

Les Extensions

- Fonctions et constantes fournissant des fonctionnalités supplémentaires au core OpenGL
  - Leur support n'est pas obligatoire
- Convention pour le nom d'une extension: api_category_name
  - api: identifie l'API (GL, GLX, WGL, ...)
  - category: identifie la categorie de l'extension
    - Extension spécifique à un constructeur: NV, ATI, SGI...
    - Extension supportée par plusieurs constructeurs: EXT
    - Extension supportée par le Khronos Group: ARB
  - name: label de l'extension
  - exemple: GL_ARB_pixel_buffer_object

Les Extensions

- La convention du nom d'une extension est étendue a l'application
  - ex: glXQueryExtensionsString, wglGetProcAddress, glGenRenderbuffersEXT...
- Récupérer la listes des extensions supportées

```
const GLubyte* extList = glGetString(GL_EXTENSIONS);
```

- Verifier qu'une extension est supportée

```
GLboolean gluCheckExtension(
     const GLubyte *extName,
     const GLubyte *extList);
```

- Récupérer l'adresse d'une fonction proposée par une extension

```
void (*glXGetProcAddress)(const GLubyte *procName);
void (*wglGetProcAddress)(const GLubyte *procName);
```

Les Extensions

- Exemple d'initialisation d'une extension sous Linux

```
PFNGLPRIMITIVERESTARTNVPROC        glPrimitiveRestartNV       = NULL;
PFNGLPRIMITIVERESTARTINDEXNVPROC   glPrimitiveRestartIndexNV  = NULL;

/* ... */
GLboolean init_GL_NV_primitive_restart(const GLubyte *extList)
{
    if(!gluCheckExtension("GL_NV_primitive_restart", extList))
        return GL_FALSE;

    glPrimitiveRestartNV = (PFNGLPRIMITIVERESTARTNVPROC)
        glXGetProcAddress(
            (const GLubyte *)"glPrimitiveRestartNV");
    glPrimitiveRestartIndexNV = (PFNGLPRIMITIVERESTARTINDEXNVPROC)
    glXGetProcAddress(
            (const GLubyte *)"glPrimitiveRestartIndexNV");

    return GL_TRUE;
}
```

Les Extensions

- Pour faire simple: GLEW (Open**GL** **E**xtensions **W**rangler **L**ibrary)
  - Inclure glew.h à la place de gl.h

```
/* #include <GL/gl.h> */
#include <GL/glew.h>
```

Liens & Livres

- http://www.opengl.org/
  - Spécification OpenGL 2.1
  - Liste des extensions
  - Forum
  - ...
- http://www.khronos.org/
- http://glew.sourceforge.net/
- http://freeglut.sourceforge.net/
- http://developer.nvidia.com/object/using_VBOs.html
- http://developer.nvidia.com/object/fast_texture_transfers.html
- http://www.onversity.net/cgi-bin/progarti/art_aff.cgi?Eudo=bgteob&P=a1201

- OpenGL Reference Manual 4th edition
- OpenGL Programming Guide 5th edition