OpenGL

Géométrie et Prise de Vue

Didier Verna

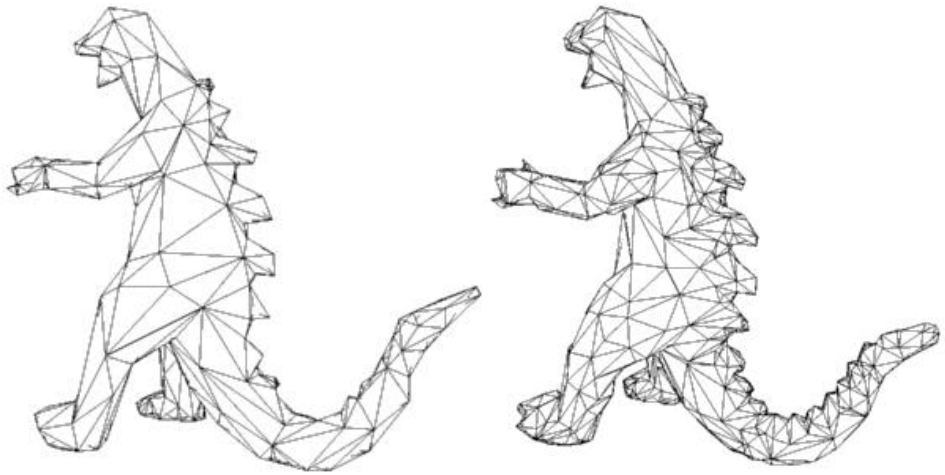
didier@lrde.epita.fr
http://www.lrde.epita.fr/~didier

Version ENSTA – D9-1 du 16 novembre 2009

License d'exploitation

Table des matières

Modélisation géométrique
La « facette » cachée des objets



- **Modélisation :** décomposition en « primitives »
- **Primitives** : point, lignes, polygones (triangles)
- **Spécification :** par sommets (vertex, *plur.* vertices)

Composition d'objets en OpenGL

À coup de primitives

- Primitives OpenGL : points, lignes, polygones
- Ligne OpenGL : segment de droite
- Polygone OpenGL : polygone simple et convexe
- Vertex : $(x, y, z, w) \Longrightarrow (x/w, y/w, z/w)$ dans l'espace euclidien

Pour les gens pressés...

void glRect{sifd} (TYPE x1, TYPE y1, TYPE x2, TYPE y2);

void glRect{sifd}v (TYPE *v1, TYPE *v2);

Tracent un rectangle défini par une diagonale dans le plan (X, Y).

Mécanisme général
Composition de formes par liste de vertices

void glBegin (GLenum mode);

Démarre la composition de primitive(s) géométrique(s).
mode : type de primitive(s)

void glVertex{234}{sifd}[v] (TYPE coords);

Défini un nouveau vertex composant la primitive courante.

void glEnd (**void**);

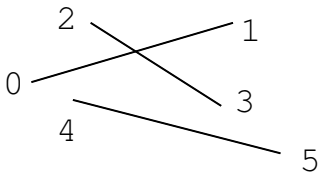
Termine la composition.

```
glBegin (GL_POLYGON);
{
  glVertex3f (5.0, -2.0, 0.0);
  // ...
}
glEnd ();
```

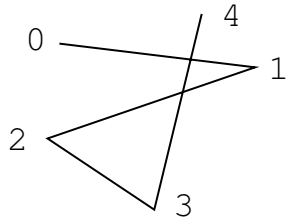
Modes de composition

Ils y sont tous

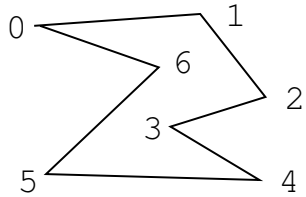
GL_LINES



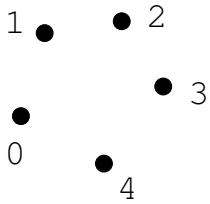
GL_LINE_STRIP



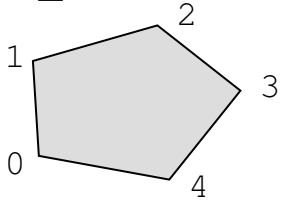
GL_LINE_LOOP



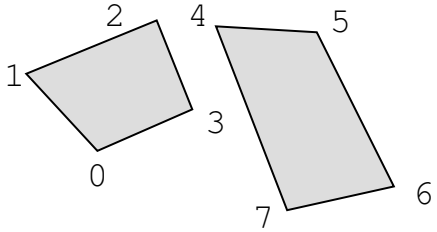
GL_POINTS



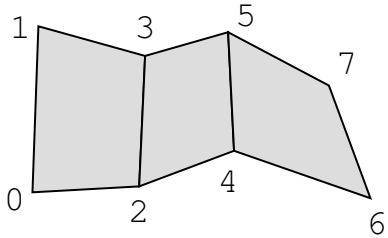
GL_POLYGON



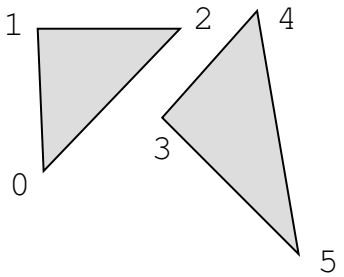
GL_QUADS



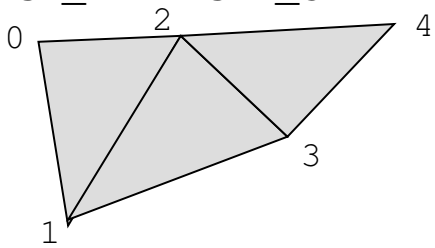
GL_QUAD_STRIP



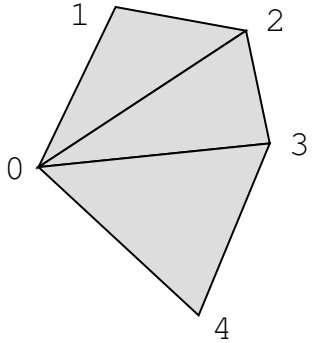
GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



Mode de tracé des polygones

Le coût du rendu n'est pas celui qu'on croit...

void glPolygonMode (GLenum face, GLenum mode);

Défini le mode de tracé pour le(s) face(s) concernée(s).

face : `GL_FRONT_AND_BACK`, `GL_FRONT`, `GL_BACK`

mode : `GL_POINT`, `GL_LINE`, `GL_FILL` (défaut)

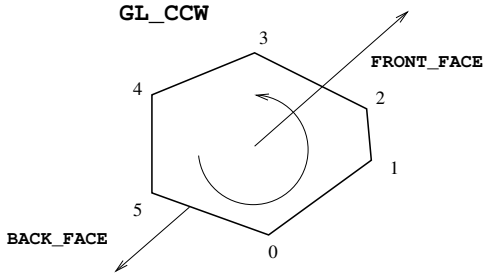
Élimination des surfaces cachées
Vision générale

- **Bas niveau :**
  - **Culling**
    Limitations : objets concaves
  - **Z-Buffer**
    Limitations : précision des calculs (offset), arêtes communes, transparence
- **Algorithmique de tri :**
  - **Listes de priorités**
    Limitations : ambiguïtés, recouvrements cycliques
  - **Partitionnement d'espace** (ex. BSP tree)
    Limitations : géométrie statique

Suppression de (sur)faces par culling

Au niveau des polygones

Un polygone a deux faces appelées `FRONT` et `BACK`



void glFrontFace (GLenum mode);

Défini l'orientation des polygones.

mode : `GL_CCW` (defaut), `GL_CW`

void glCullFace (GLenum mode);

Défini le culling.

mode : `GL_BACK`, `GL_FRONT`, `GL_FRONT_AND_BACK`

glEnable / glDisable (GL_CULL_FACE);

Demande à OpenGL de (dés)activer le culling.

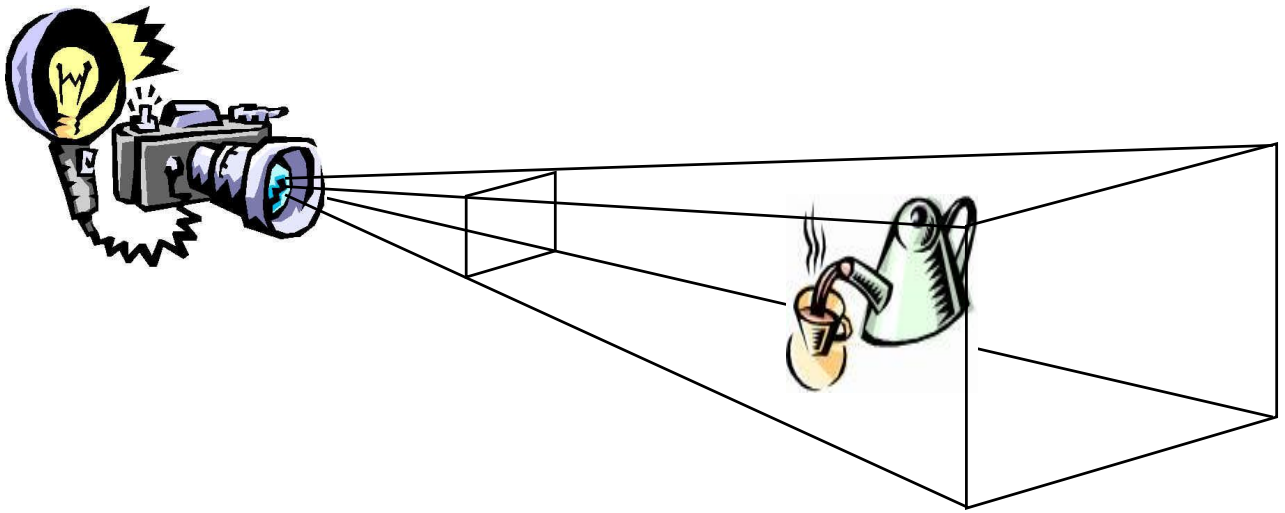
Suppression de zones par Z-buffer

Au niveau de la rasterization

- **Requête :**
  - **GLX :** `glXChooseVisual()` avec `GLX_DEPTH_SIZE`
  - **GLUT :** `glutInitDisplayMode()` avec `GLUT_DEPTH`
- **(Dés) Activation :**

```
glEnable / glDisable (GL_DEPTH_TEST);
```

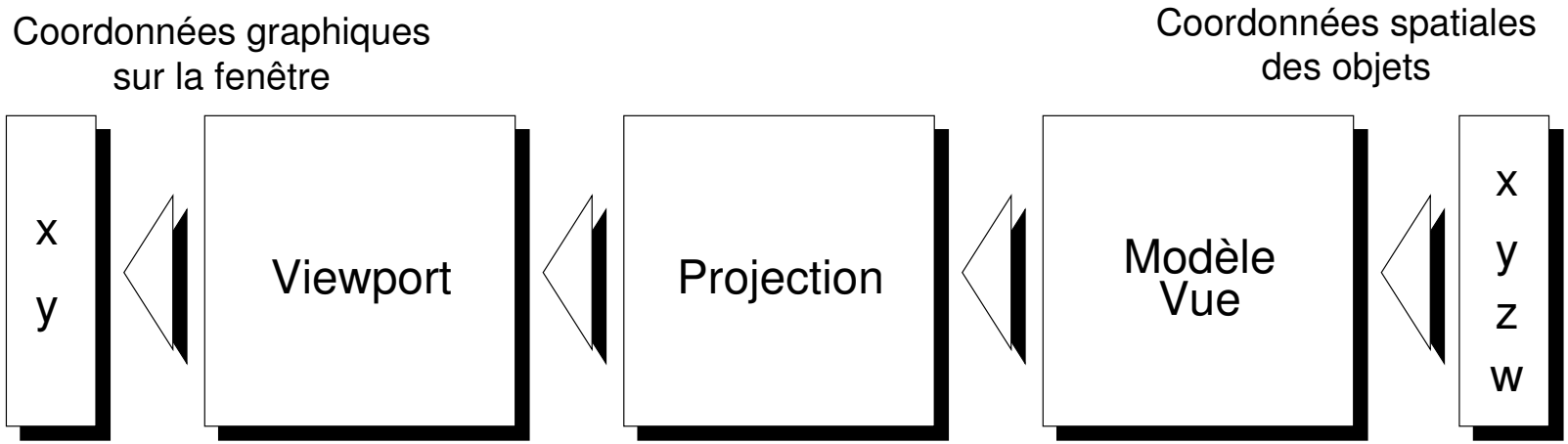
Le pipeline géométrique
Vision générale



1. Arranger la scène — ***modèle***
2. Pointer l'appareil vers la scène — ***vue***
3. Paramétrer l'objectif (grand angle, zoom...) — ***projection***
4. Régler l'agrandissement et tirer la photo — ***viewport***

Équivalent informatique

Bienvenue dans la matrice. . .



- **Remarque :** dualité modèle $\Longleftrightarrow$ vue
- **Implémentation :** transformations matricielles (affines)
- **Dimension 4 :** problème de la translation
- **Coordonnée homogène :** w (point / vecteur)

Mode matriciel d'OpenGL

Sur quelle matrice travaille t'on ?

Deux modes : **Modelview** et **Projection**

void glMatrixMode (GLenum mode);

Défini la matrice courante (affectée par les prochaines transformations).

mode : `GL_MODELVIEW` (défaut), `GL_PROJECTION`

Préliminaires
Des choses qui changent rarement

void glClearColor (GLclampf red, GLclampf green,

GLclampf blue, GLclampf alpha);

Spécifie la couleur de nettoyage (du color buffer).

void glClearDepth (GLclampf depth);

Spécifie la profondeur de nettoyage (du Z-buffer).

Avant / Après :

À ne pas oublier !

- **Avant traçage :**

  void glClear (GLbitfield mask);

  Nettoie les buffers spécifiés dans le masque.

  mask : `GL_COLOR_BUFFER_BIT`, `GL_DEPTH_BUFFER_BIT`

- **Après traçage :**

  void glFlush (**void**);

  Force l'execution de toutes les commandes latentes.

  void glFinish (**void**);

  Idem, et attend la fin de l'exécution.

- **Remarque :** Les commandes qui swappent les buffers en mode double flushent automatiquement.

Commandes spécifiques
Manipulations matricielles indirectes

Les fonctions suivantes définissent des matrices qui multiplient (à droite) la matrice courante :

void glTranslate{fd} (TYPE x, TYPE y, TYPE z);

Applique une translation de x, y, z sur les trois axes.

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

Applique une rotation de a degrés autour de l'axe $\overrightarrow{x, y, z}$

void glScale{fd} (TYPE x, TYPE y, TYPE z);

Applique une homothétie de x, y, z sur les trois axes.
Utiliser `glScale*` avec parcimonie.

Commandes générales
Manipulations matricielles directes

void glLoadIdentity (**void**);

Charge l'identité dans la matrice courante (défaut).

void glLoadMatrix{fd} (**const** TYPE *mat);

Charge la matrice courante avec celle spécifiée.

```
TYPE mat[16];
M ⟵ mat
```

void glMultMatrix{fd} (**const** TYPE *mat);

Multiplie la matrice courante par celle spécifiée.

```
TYPE mat[16];
M ⟵ M.mat
```

Commandes liées aux états

OpenGL dispose d'une mémoire d'états

void glGetFloatv (GLenum name, GLfloat ∗mat);

Récupère la matrice courante spécifiée.

name : `GL_MODELVIEW_MATRIX`, `GL_PROJECTION_MATRIX`

void glPushMatrix (**void**);

Ajoute une matrice sur la pile de matrices courante, initialisée comme la précédente.

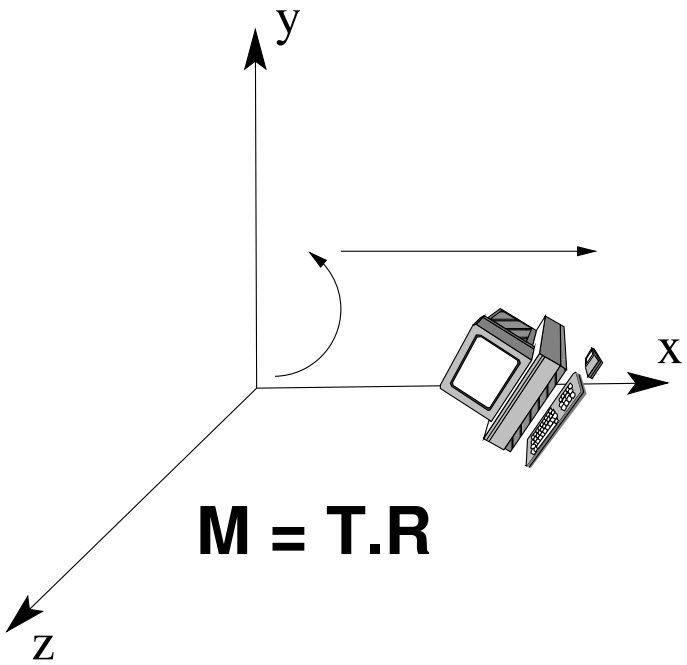
void glPopMatrix (**void**);

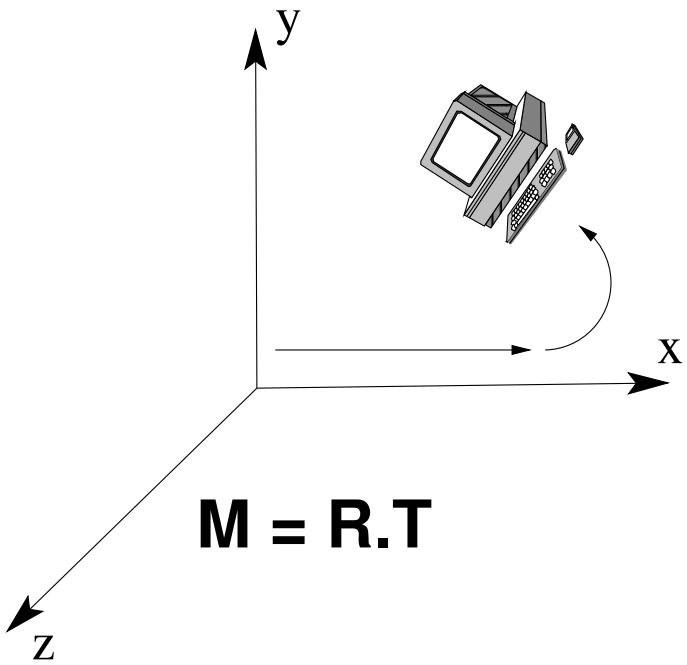
Enlève une matrice de la pile de matrices courante.

- **Modelview** : pile de profondeur 32
- **Projection** : pile de profondeur 2

Réfléchissez à l'ordre des transformations ! !

Si vous ne voulez pas perdre vos cheveux





Attention :

- Ordre des transformations inversé dans le code
- Matrices en système américain

Placement du point de vue

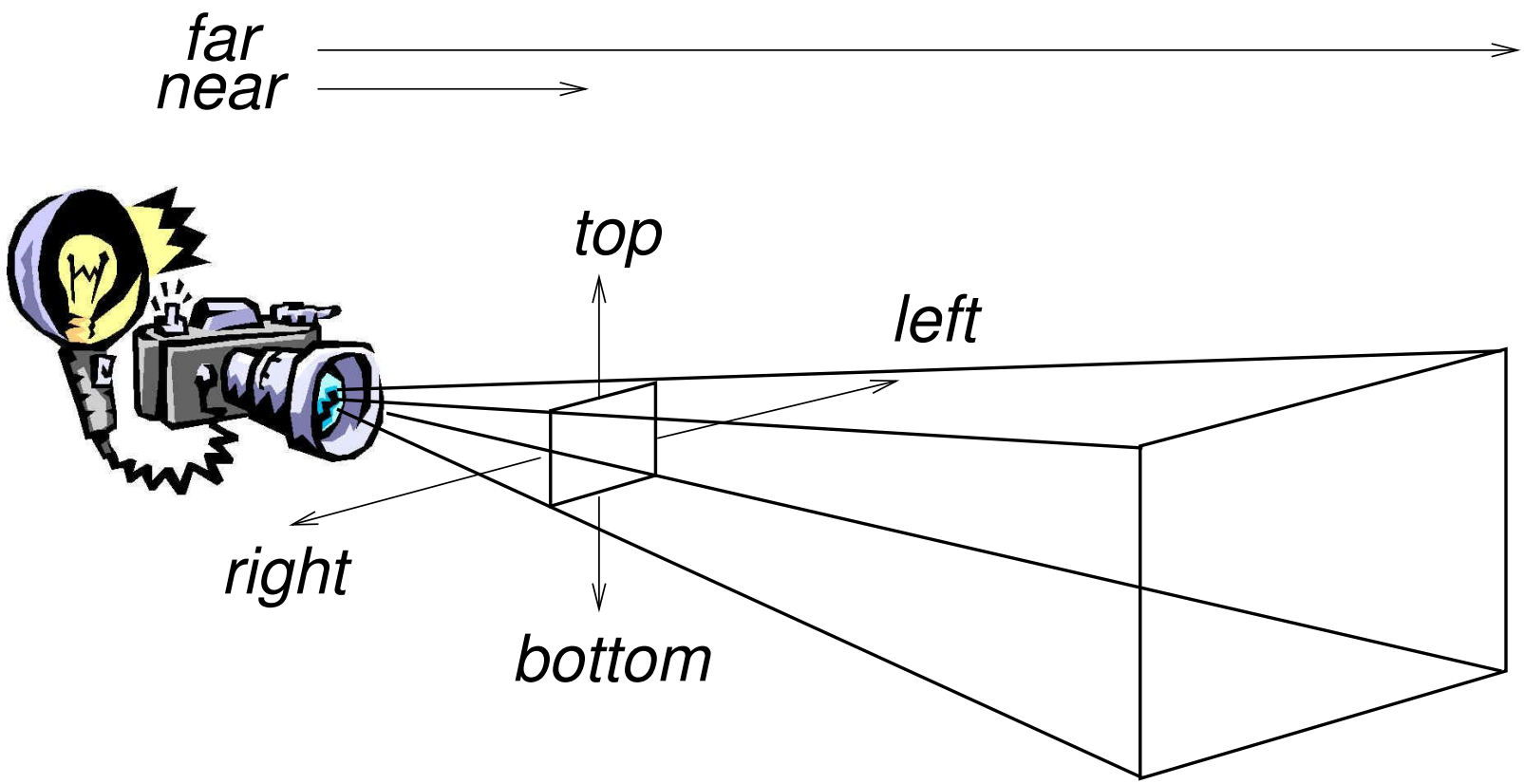
C'est pareil

Les mêmes fonctions que les 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 « haut » par le vecteur $\overrightarrow{U}$.
La matrice obtenue multiplie (à droite) la matrice courante.

void glFrustum (GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);

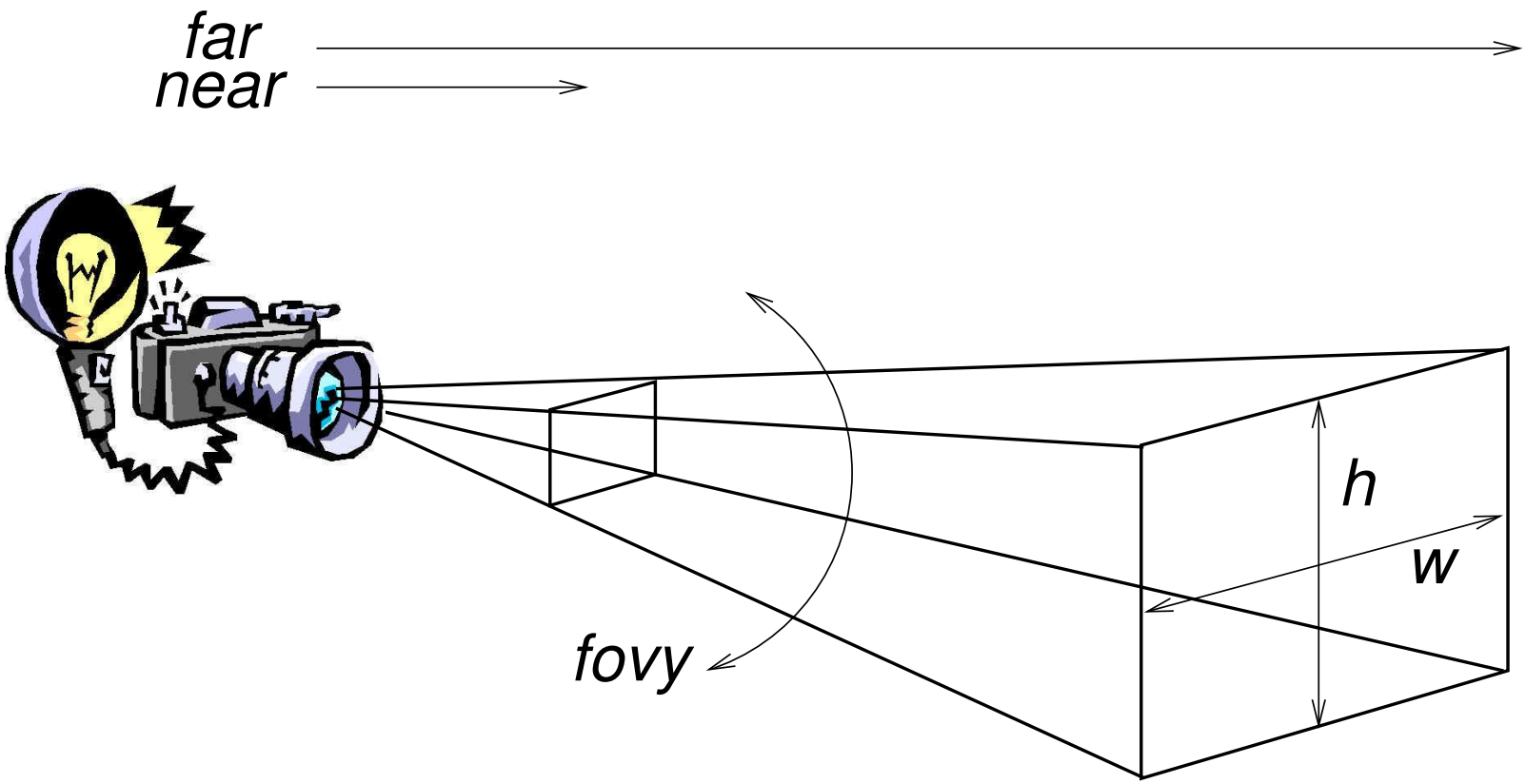


Alternative GLU

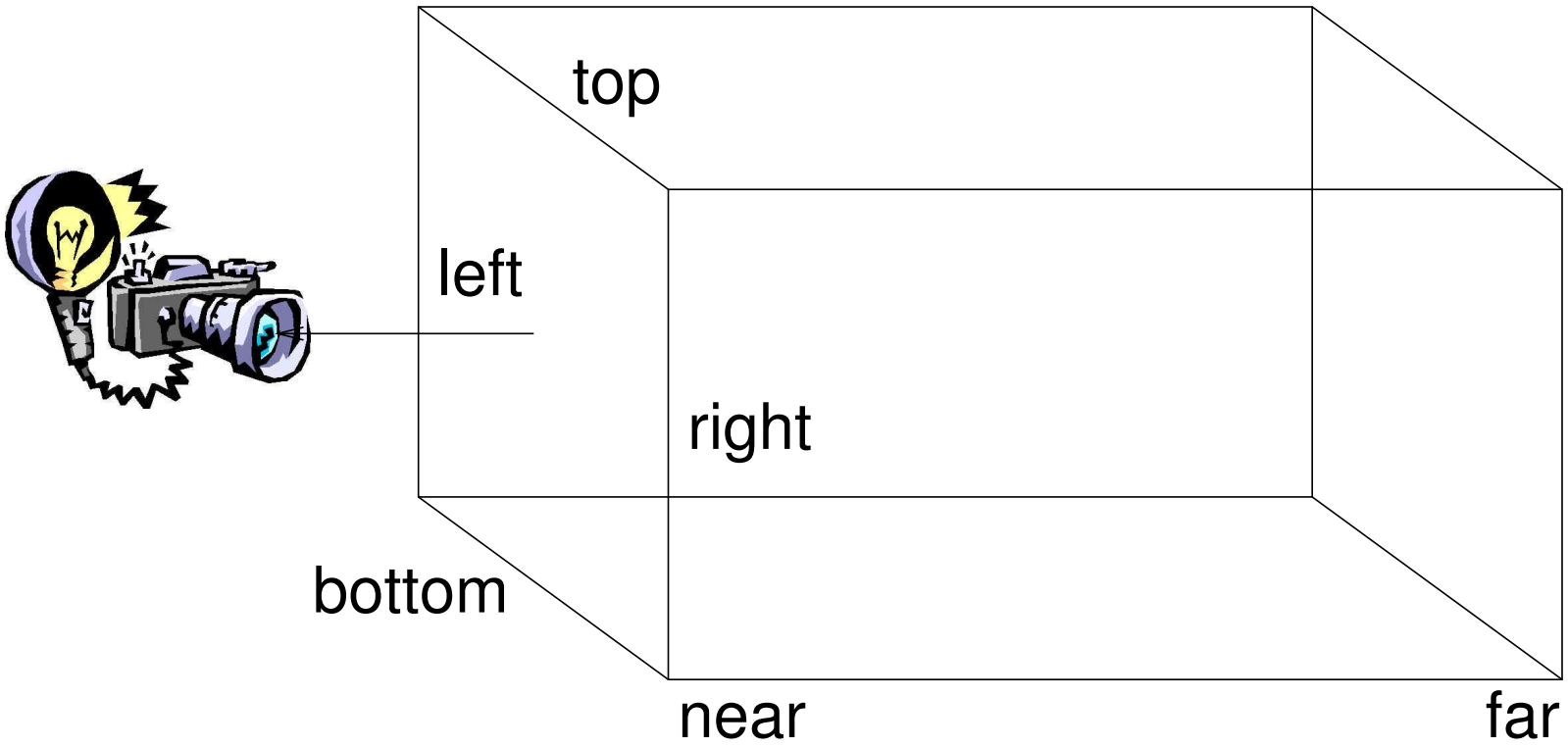
Plus simple, plus limitée

void gluPerspective (GLdouble fovy, GLdouble aspect,

GLdouble near, GLdouble far);



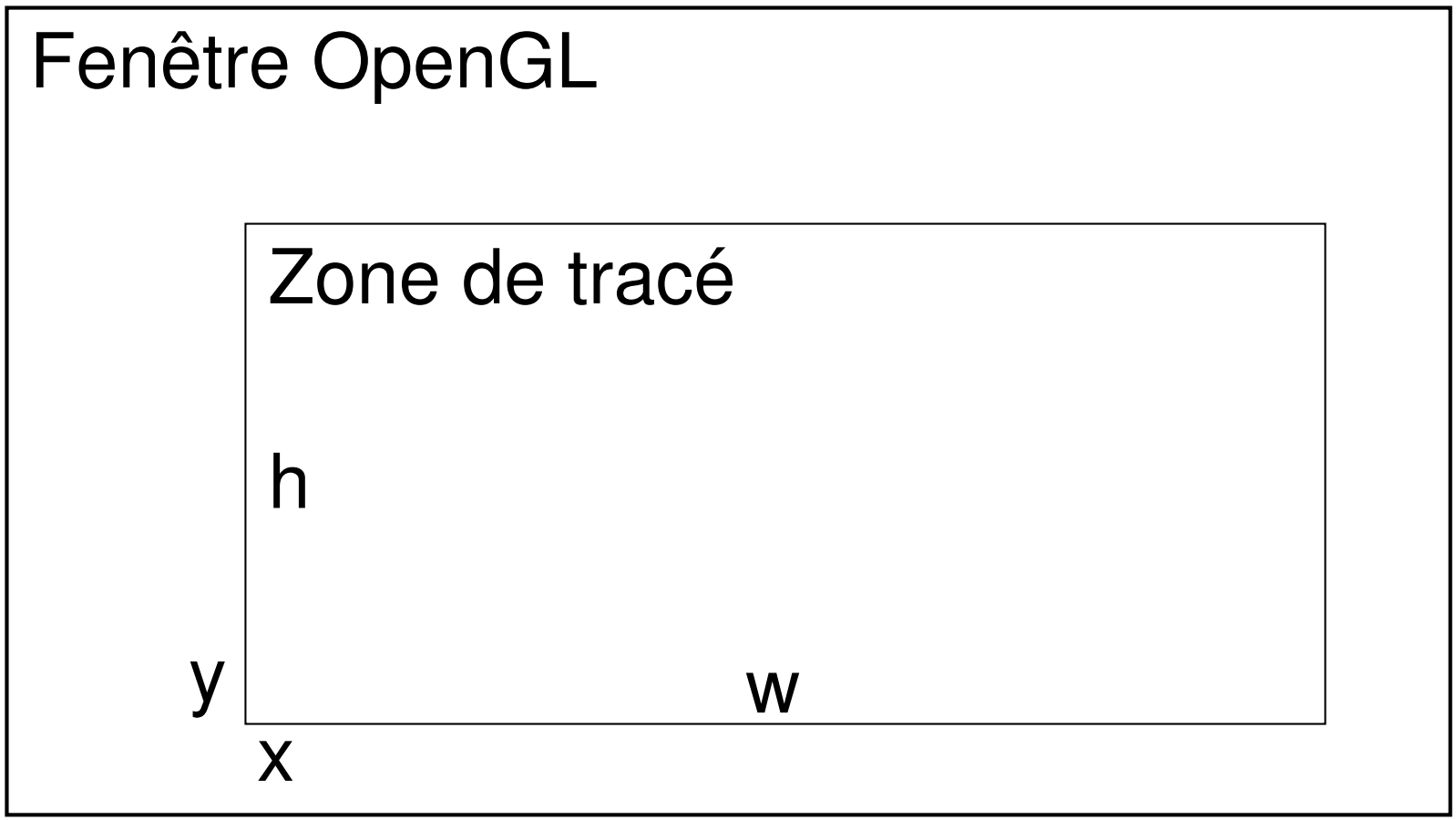
void glOrtho (GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);



Transformation de Viewport

Où tracer, au final ?

void glViewport (GLint x, GLint y, GLsizei w, GLsizei h);



Résultat du TP

Ce que vous devriez obtenir. . .