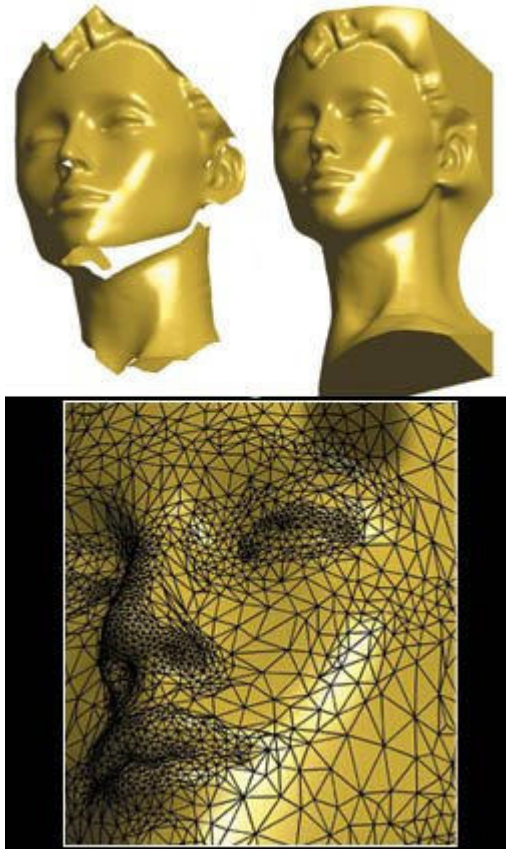


# Plans, polygones, et maillages

Alexandre Meyer

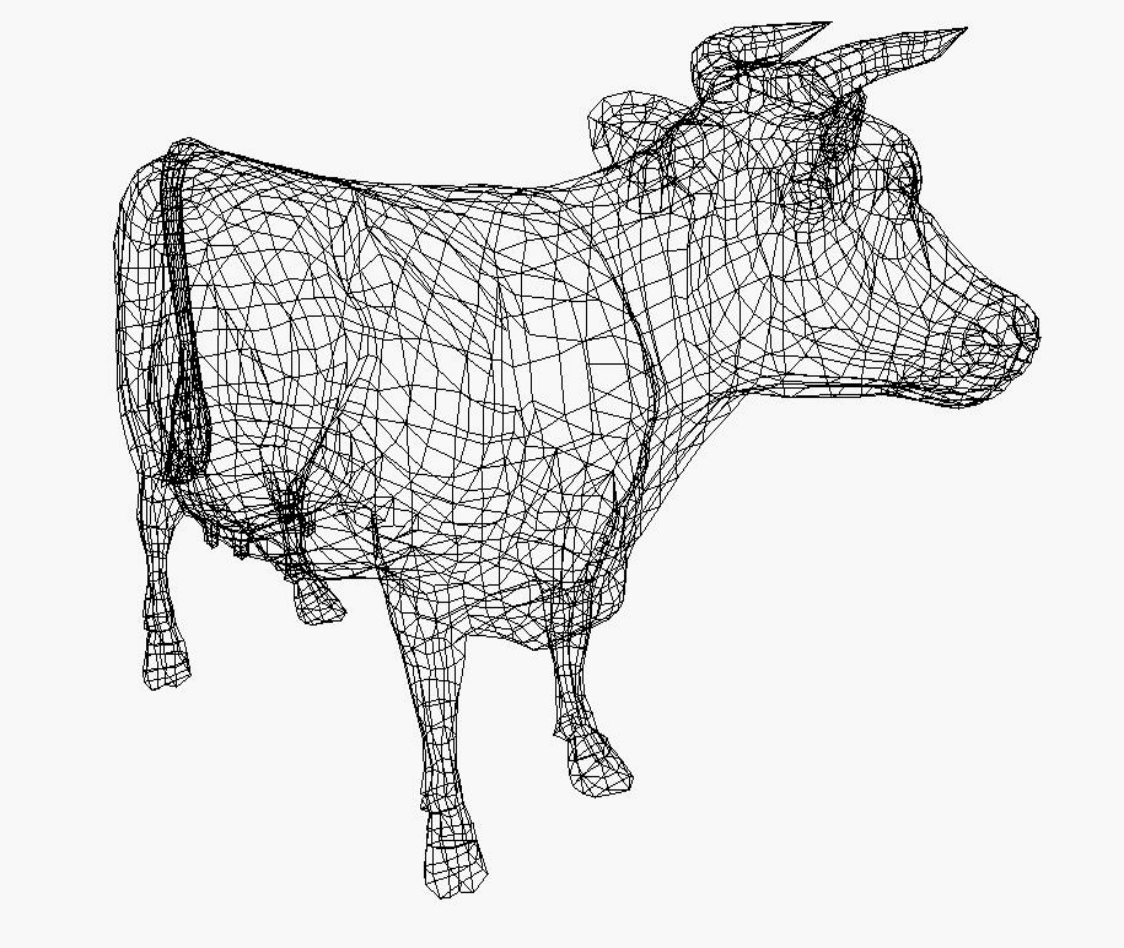
Florence Zara

http://licence-info.univ-lyon1.fr/LIFGRAPHIQUE

# Plan

- Polygones (2D)
- Polyèdres (3D)
- Format de fichier
- OpenGL

# Maillages polygonaux



Les maillages polygonaux sont la représentation la plus commune

# Polygones (2D)

- Un polygone (face) Q est défini par une série de points

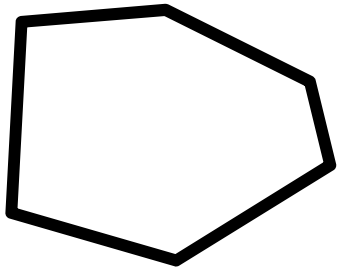
$$[p_0, p_1, p_2, \ldots, p_{n-1}, p_n]$$

$$p_i = (x_i, y_i, z_i)$$

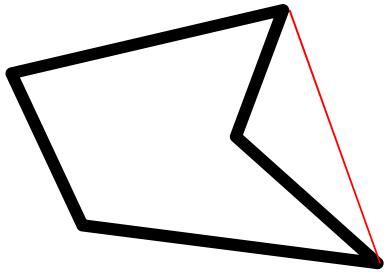
- Les points doivent être **co-planaires**
- 3 points définissent un plan. Un 4e point ne sera pas forcement sur ce plan

# Convexe / concave

- *Convexe*
  - *un polygone est convexe s'il n'est pas croisé et si toutes ses diagonales sont entièrement à l'intérieur de la surface délimitée par le polygone*



- *Concave*
  - *SI préfère les polygones convexes*
  - *SI préfère les triangles !!*
    - *Conversion facile d'un polygone convexe en triangles*
    - *Difficile pour les concaves*



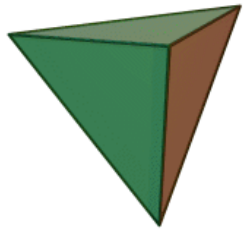
# Polyèdres (3D)

- Les polygones sont souvent groupés pour former des polyèdres
  - Une arête joint 2 sommets
  - Une arête joint 2 faces / polygones
  - Les faces ne s'intersectent pas

- Ils sont nommés selon leur nombre de faces

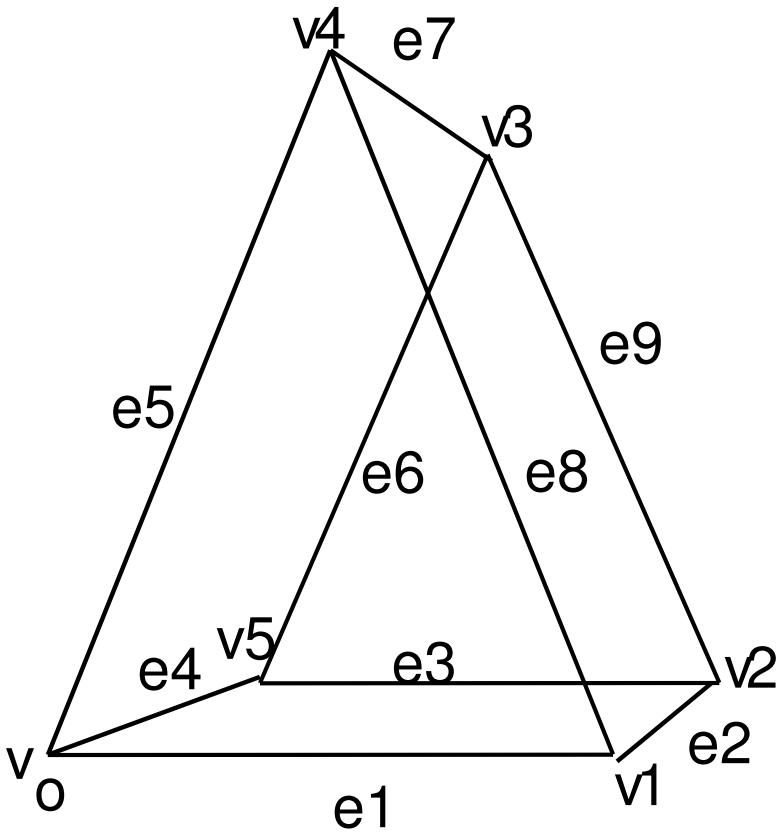
# Polyèdres (3D)

- Exemple de polyèdres :
  - Tétraèdres (4 faces)
  - Pentaèdres (5 faces)
  - Hexaèdres (6 faces)
    - Hexaèdre régulier : cube



- Caractéristique d'Euler pour les polyèdres convexes
  - V-E+F=2  (V= #Vertex, E= #Edge, F= #Face)

# Exemple




- F0=v0v1v4
- F1=v5v3v2
- F2=v1v2v3v4
- F3=v0v4v3v5
- F4=v0v5v2v1
- F5 =v0v5v3v4

- V=6,F=5, E=9
- V-E+F=2 (ok !)

# Structure de données

- Exhaustif (Tableau de listes de sommets)
  - faces[1] = (x0,y0,z0),(x1,y1,z1),(x3,y3,z3)
  - faces[2] = (x2,y2,z2),(x0,y0,z0),(x3,y3,z3)
  - etc.

- Très coûteux en mémoire car chaque sommet apparaît au moins 3 fois !
- Si un point bouge, un trou apparaît car il n'y a pas de notion de connectivité !

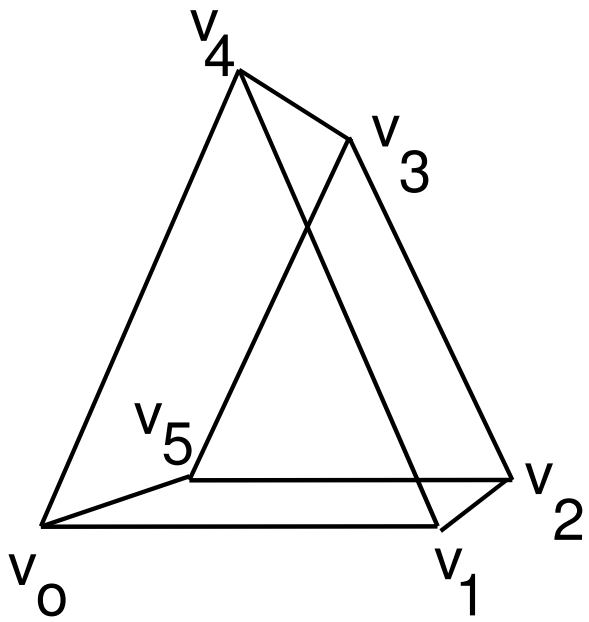
# Structure de données

Indexed Face Set

- Tableau de sommets (Vertex array)
  - vertices[0] = (x0,y0,z0)
  - vertices[1]= (x1,y1,z1)
  - etc.

- Tableau de faces (liste d'indices dans le tableau de sommets)
  - faces[0] = 0,2,1
  - faces[1] = 2,3,1
  - etc.

# Ordre des sommets




- Polygone $v_0,v_1,v_4$ != $v_0$, $v_4$, $v_1$
- Le vecteur normal pointe dans des directions opposées

  $v_0v_1 \times v_0v_4 = - v_0v_4 \times v_0v_1$
- Habituellement un polygone n'est visible que depuis les points de son demi-espace positif
- **Back-face culling** (si le point de vue n'est pas devant le polygone, on ne l'affiche pas)

  (*cf.* OpenGL)

# Format de fichier OBJ (AliasWavefront)

- Fichiers OBJ sont au format ASCII
- # Commentaire jusqu'à la fin de la ligne

- v *float float float*
  - Vertex (sommet). Le 1er vertex a pour numéro 1
- vn *float float float*
  - Vecteur normal. La 1ere normale a pour numéro 1
- vt *float float*
  - Coordonnée texture. La 1ere coordonnée a pour numéro 1
- f *int int int* ...
- f *int/int int/int int/int* ...
- f *int/int/int int/int/int int/int/int* ...
  - Face. Les numéros correspondent respectivement au indice de Vertex/CoordTexture/VecteurNormal

# Format OBJ : exemple (cube)

```
# Ceci est un cube
# 8 sommets
v 1 1 1
v 1 1 -1
v 1 -1 1
v 1 -1 -1
v -1 1 1
v -1 1 -1
v -1 -1 1
v -1 -1 -1
# 6 faces
f 1 3 4 2
f 5 7 8 6
f 1 5 6 2
f 3 7 8 4
f 1 5 7 3
f 2 6 8 4
```

- Pour la description d'une face des champs peuvent éventuellement être vide

  Par exemple :

  ```
  f  1//7  2/3/5  3//  4/6/8
  ```

# Formats : OBJ et VRML

- OBJ
  - Format efficace, simple à gérer et standard
  - Trouve facilement des fichiers sur Internet

- VRML (fichier.WRL)
  - VRML=Virtual Reality Modeling Language
  - Egalement, listes de sommets et de polygones

```
Coordinate3 {
        point       [       -2.250000 3.110000 -0.350000,
                            -2.170000 3.070000 -0.520000,
                            ...
                    ]
}
IndexedFaceSet {
        coordIndex [        0, 1, 2, 2, -1,
                            3, 2, 4, 4, -1,
                            ...
                     ]
}
```


OpenGL™

# OpenGL : généralité

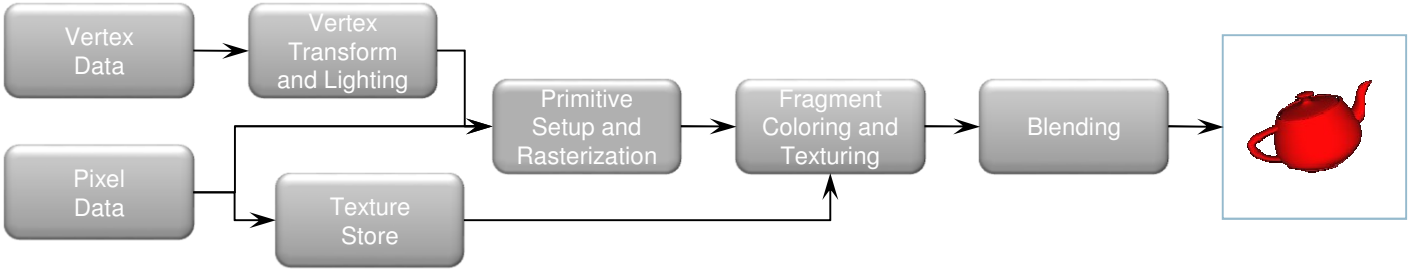
- OpenGL est une API 3D (Application Programmer ’s Interface)
  - basé sur IRIS GL de Silicon Graphics
  - de bas niveau pour avoir l’indépendance vis à vis
    - système de fenêtrage
    - plate-forme matérielle
    - système d’exploitation

- Architecture client-serveur : le client émet

les commandes, le serveur les exécute

# Les débuts …

- OpenGL 1.0 : 1994
- Pipeline fixe
  - Toutes les operations étaient cablées et fixes

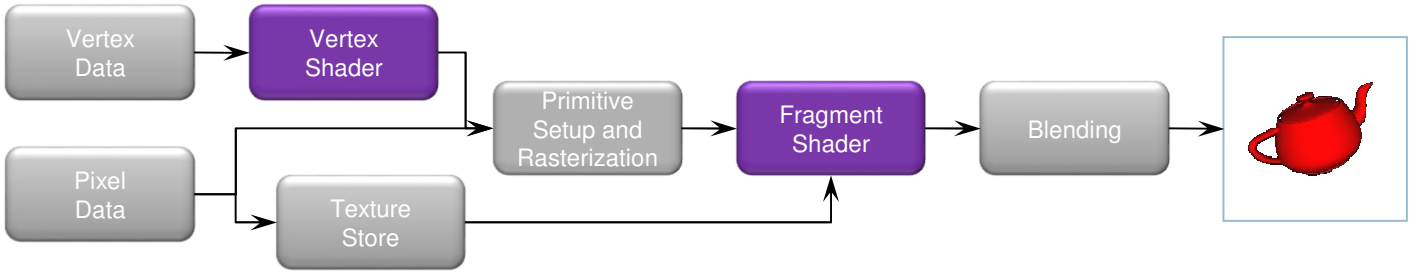



- Le pipeline a évolué
  - Mais en restant fixe

OpenGL versions 1.1 ➔ 2.0 (Sept. 2004)

# OpenGL 3.1

- Fin du pipeline fixe
  - programs = shader
    - Vertex program : transforme les sommets
    - Fragment program : transforme les fragments=pixel

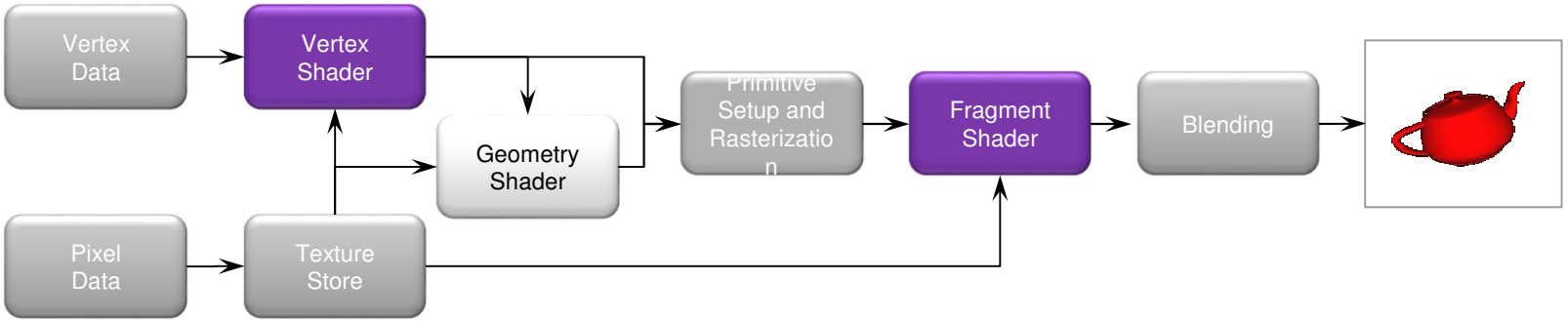



- En plus, les données sont stockés que sur GPU
  - Vertex ➔ buffer objects

# Plus programmable

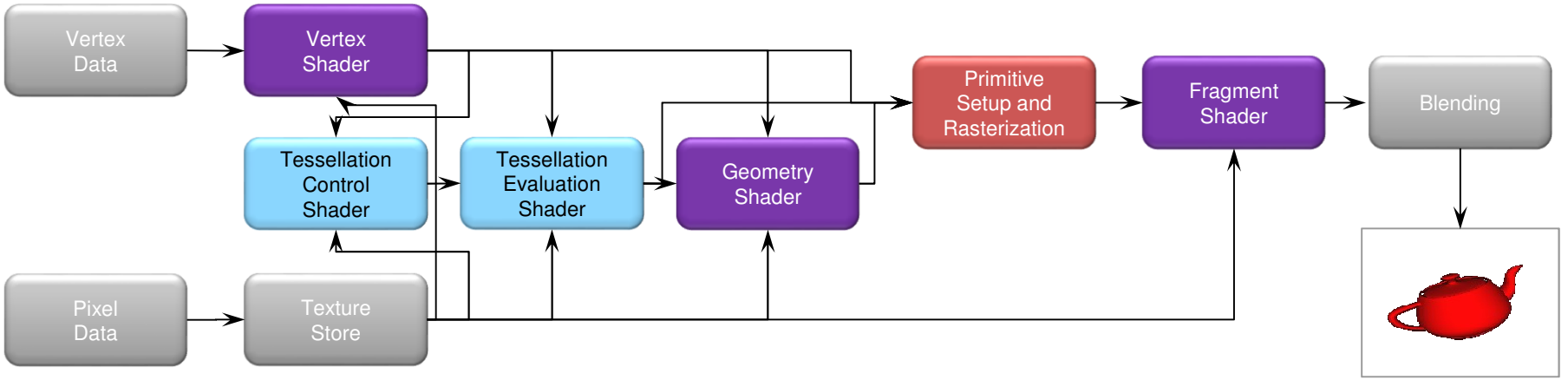
- OpenGL 3.2 (released August 3rd, 2009)
  - Ajoute un étage de shader : geometry shaders

➔ Modifie les primites geometriques (triangles) dans le pipeline

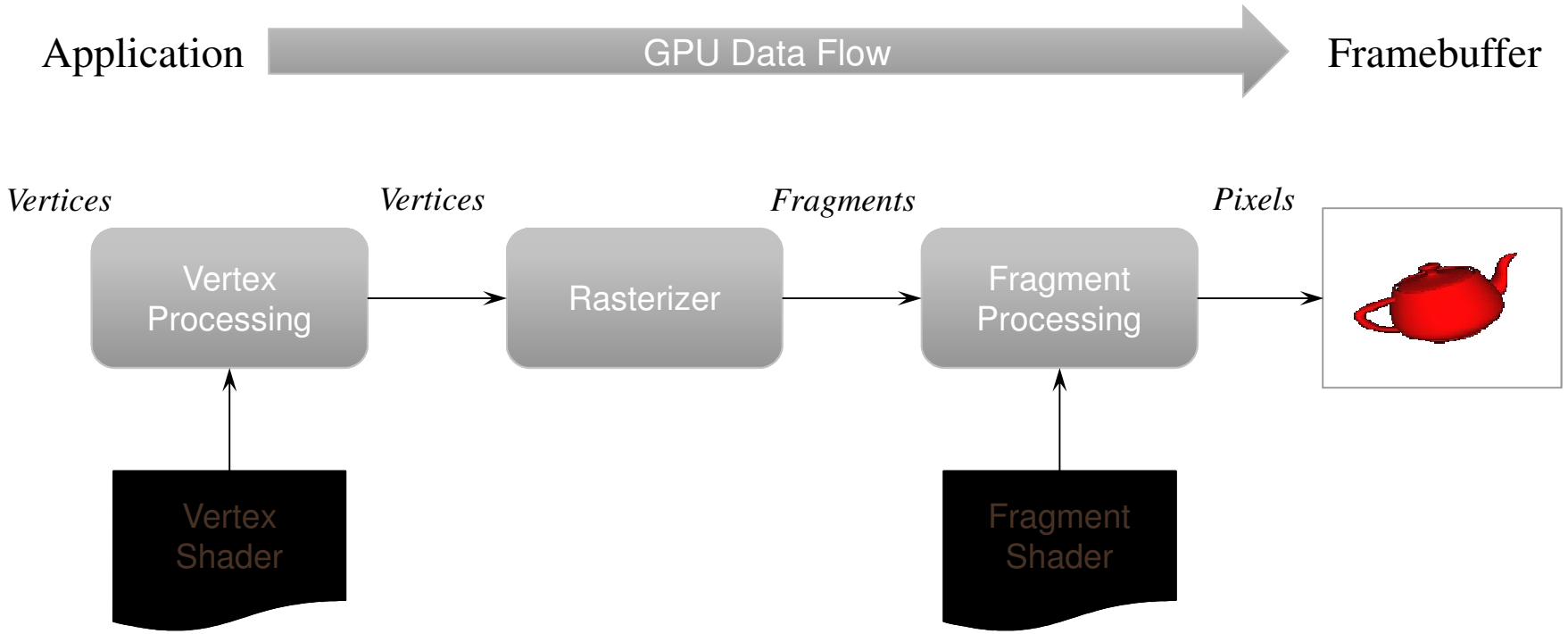


# Dernier Pipelines

- OpenGL 4.1 (released July 25th, 2010) ajoute d'autres étages
  - *tessellation-control*
  - *tessellation-evaluation* shaders
- Puis Compute shader



# Un pipeline simplifié



# OpenGL ES et WebGL

- OpenGL ES 2.0
  - Pour système embarqué comme les mobiles
  - Basé sur  OpenGL 3.1
  - Avec Shader
- WebGL
  - JavaScript implementation of ES 2.0
  - Tourne sur des navigateurs récents

# OpenGL : librairies associées

- Comme OpenGL est indépendant du système de fenêtrage, des librairies additionnelles sont utilisées pour intégrer OpenGL dans de tels systèmes :
  - GLX pour XWindow
  - WGL pour Windows
  - GLUT
  - SDL,
  - Qt
  - MFC (windows)
  - etc.

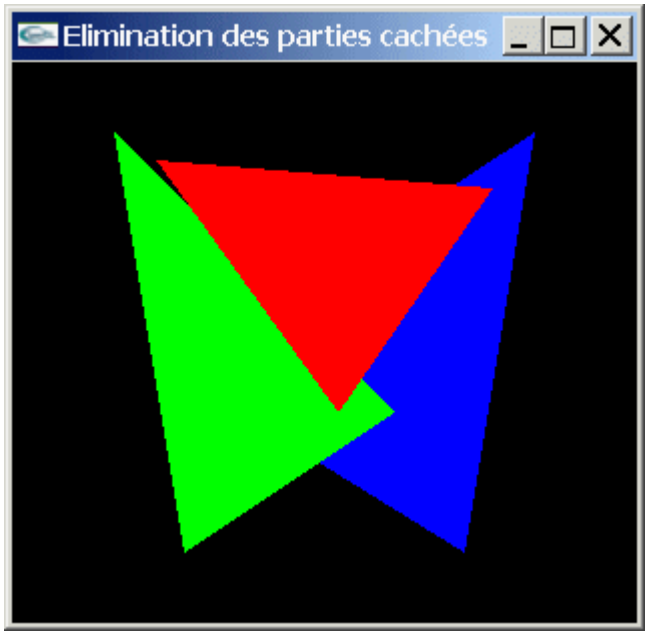
# OpenGL : machine à états

- OpenGL possède des états courants
- On peut activer / désactiver les états avec
  - glEnable()
  - glDisable()
- Tous les états ont des valeurs par défaut (pas besoin de tous les définir en début de programme !)

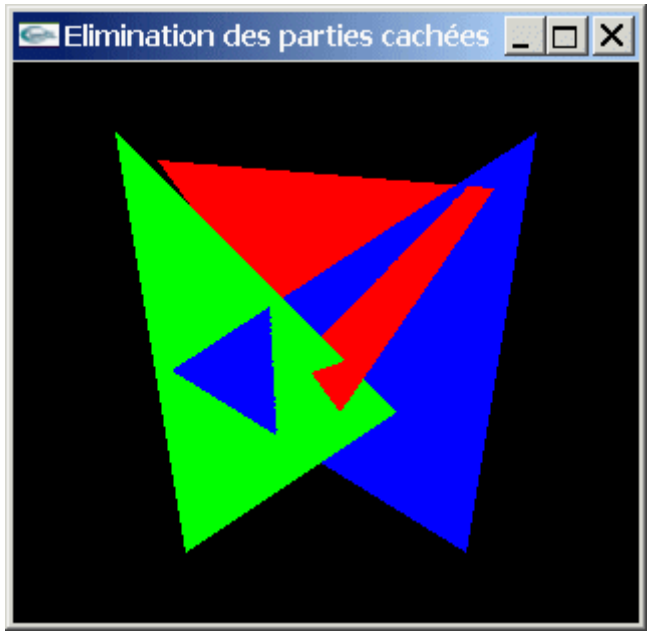
# Exemple avec le Z-buffer

- Pour activer le test des Z avec le Z-buffer

```
glEnable(GL_DEPTH_TEST) ;
```




non activé

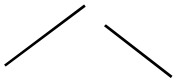



activé

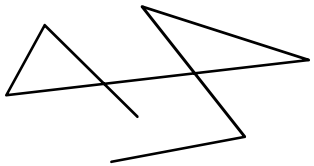
# OpenGL : Primitives

- Vertices (sommets)

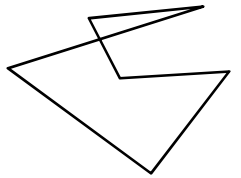
GL_POINTS



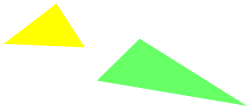
GL_LINES



GL_LINE_STRIP



GL_LINE_LOOP



GL_TRIANGLES

GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

# gKitLight …

- GL-shader (sans le pipeline fixe)
  - Besoin d'une base de code
  - sinon il faut tout redéfinir

- Regardons
  - Viewer.h / .cpp

- Démo du code

# Exercice : affichage d'une pyramide

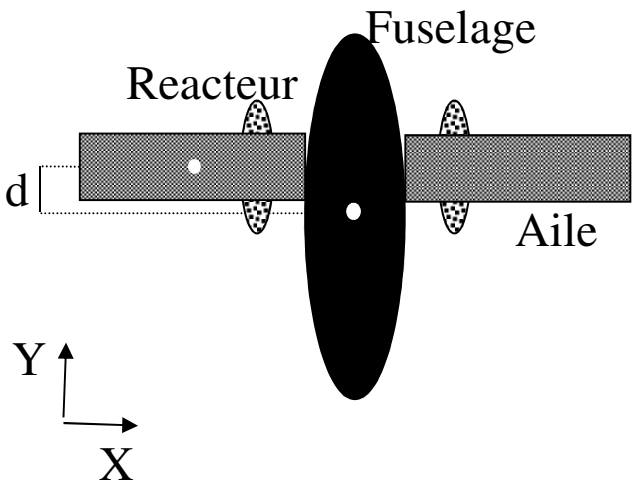
- Pyramide
  - Avec 4 polygones : base triangulaire
  - Avec des TRIANGLE_STRIP
  - Avec une structure indexée

- Idem avec un Cube (cf. TD)

# Exercice

- On souhaite dessiner un avion (on ne dessine pas la queue de l'avion)
- On dispose de 2 fonctions
  - *drawCube* : dessine un cube centré en 0
  - *drawSphere* : dessine une sphère centrée en 0

**Vue de dessus**




**Vue de derrière**