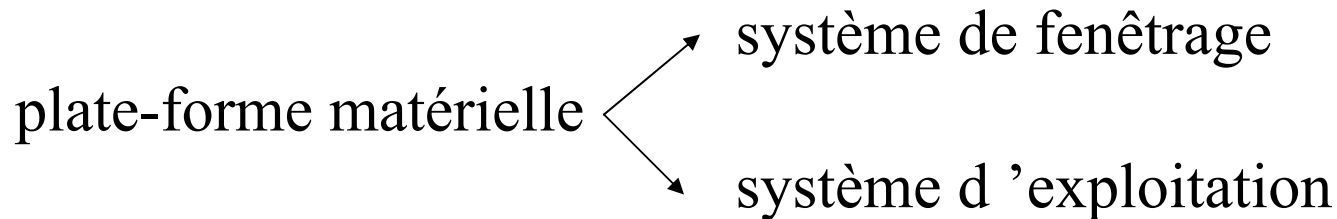


OpenGL

1. Généralités : OpenGL 1.x

- ❑ est une API (Application Programmer 's Interface) 3D ;
- ❑ est basé sur IRIS GL de Silicon Graphics ;
- ❑ est de bas niveau pour avoir l 'indépendance vis à vis de la

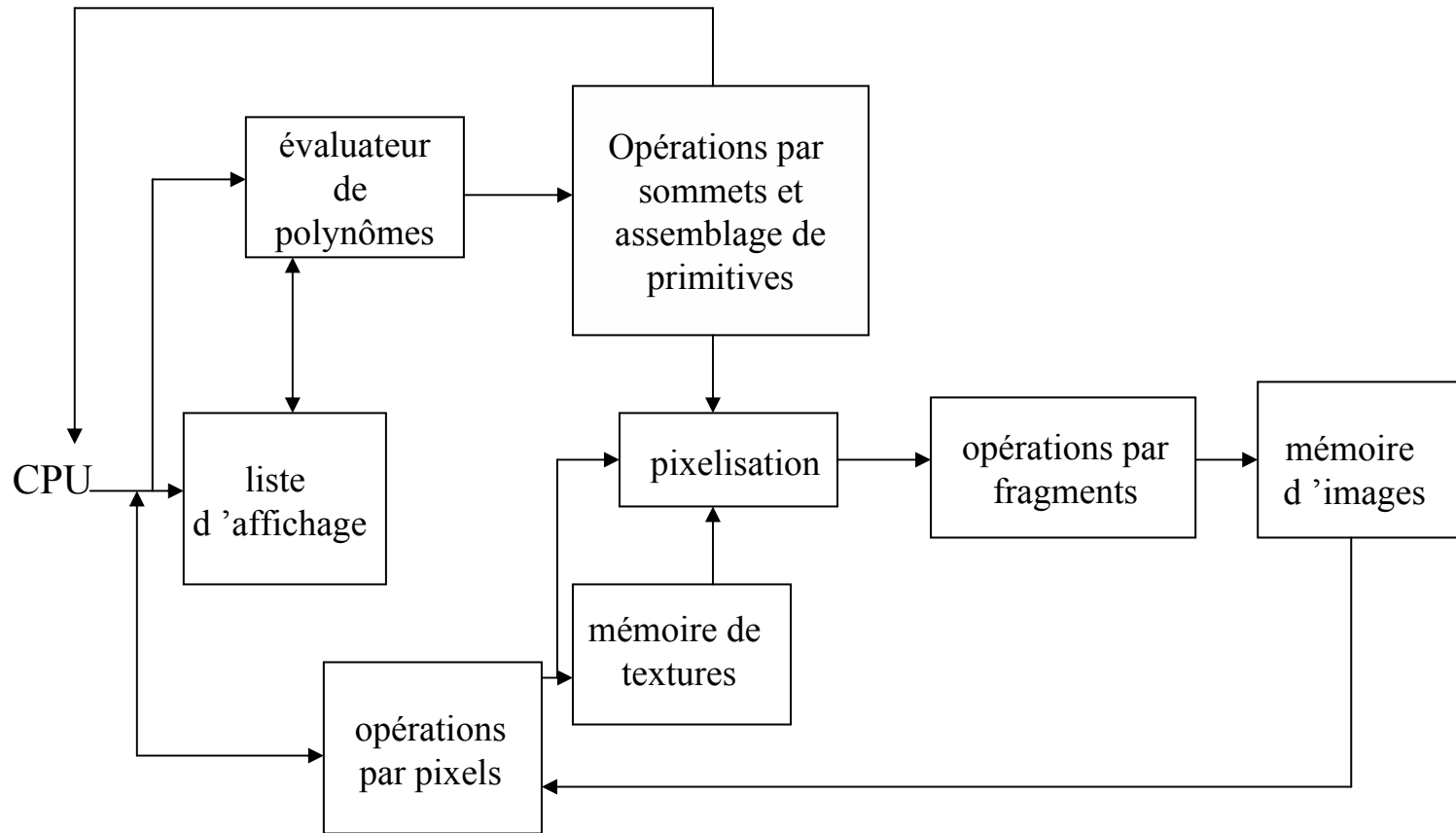


- ❑ est uniquement utilisable avec une mémoire d 'images ;

- est basé sur une architecture client-serveur : le client émet les commandes, le serveur les exécute ;
- possède un contrôle à grain fin → primitives ;
- est une machine à états : OpenGL possède un état courant pour tout (repère courant, projection courante, couleur courante, ...) ;
- n'est pas un langage de programmation (cf. RenderMan) : le modèle d'exécution des commandes est un pipeline à topologie fixe
- supporte des primitives géométriques (2D et 3D) et des primitives images ;

- ❑ supporte 2 modes de commande : le mode **immédiat** et la **liste d'affichage** ;
- ❑ ne supporte que le tampon de profondeur (z-buffer) pour l'élimination des parties cachées ;
- ❑ ne supporte que des modèles d'éclairage locaux ;
- ❑ ne comporte que des opérations de rendu

2. Principes



architecture d'OpenGL

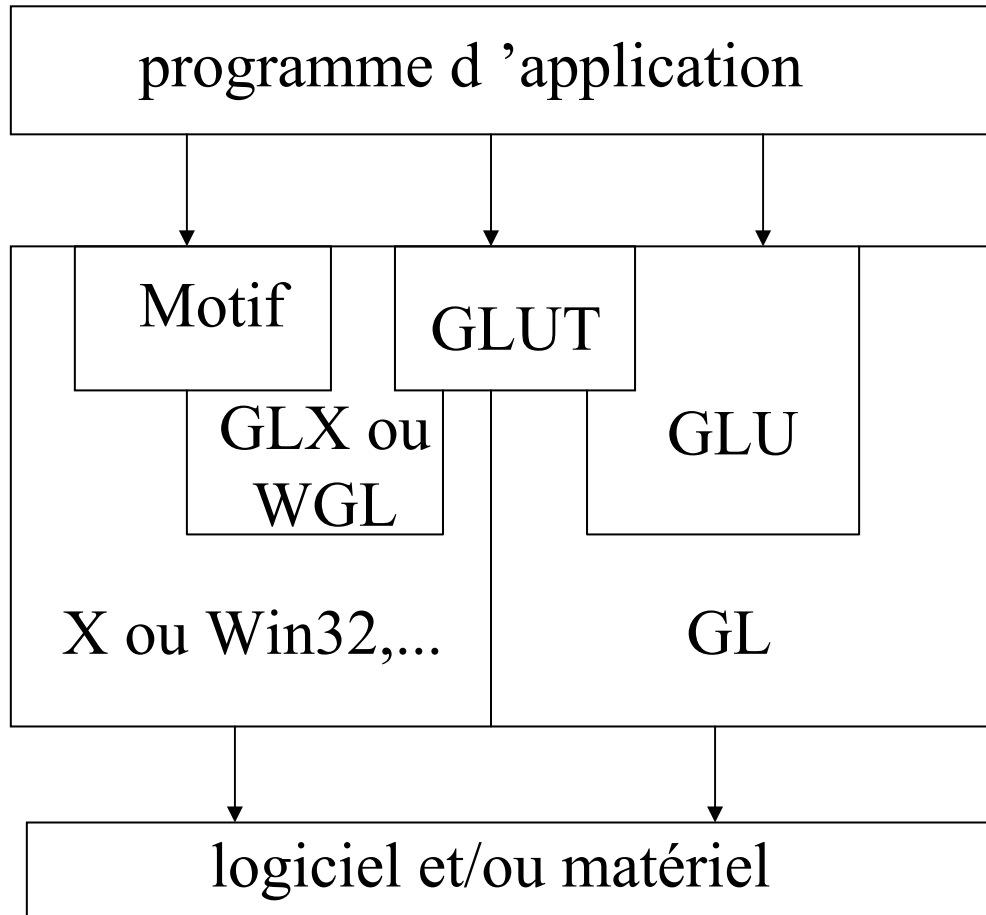
- OpenGL dessine des primitives dans une mémoire d'images en fonction d'un certain nombre d'états qui doivent être fixés ;
- primitives = point, segment de droite, polygone
bitmap, image graphique ;
- les états sont définis par des appels de fonctions ;
- en fait, deux pipelines :
 - un pipeline géométrique, basé sur les sommets
des primitives
 - un pipeline image, basé sur les pixels ;

□ comme OpenGL est indépendant du système de fenêtrage, des bibliothèques additionnelles sont utilisées pour intégrer OpenGL dans de tels systèmes :

GLX pour XWindow

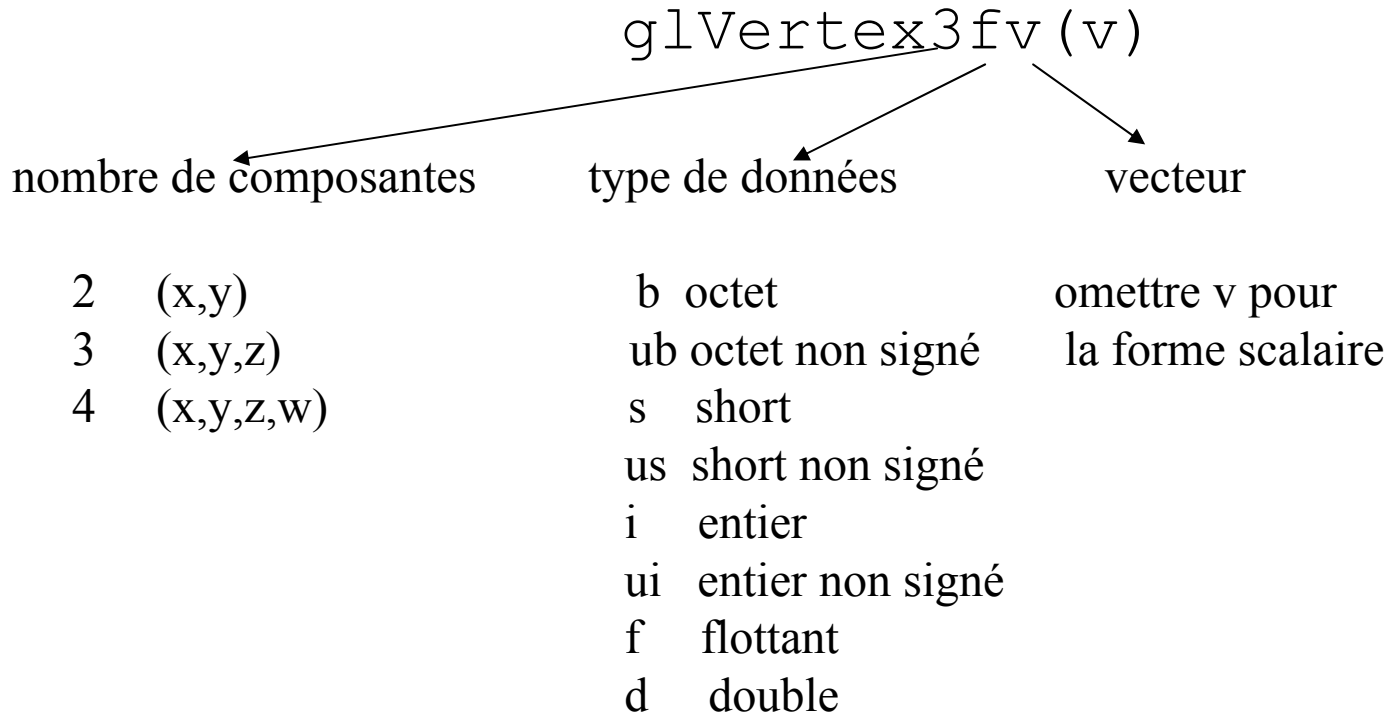
WGL pour Windows

GLUT indépendant du système de fenêtrage



3. Syntaxe

- ❑ les noms des fonctions OpenGL commencent par gl ;
- ❑ chaque nouveau mot commence par une majuscule ;
- ❑ le suffixe indique le nombre et le type des arguments ;



□ les primitives sont spécifiées à l'aide de

```
glBegin (primType) ;  
glEnd;
```

les types possibles sont :

GL_POINTS

GL_LINES

GL_POLYGON

GL_TRIANGLES

GL_QUADS

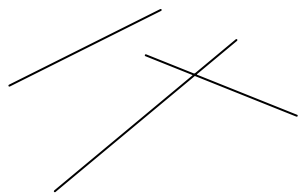
GL_LINE_STRIP

GL_LINE_LOOP

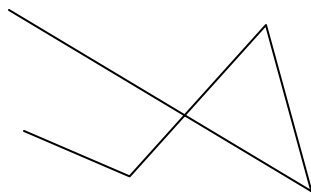
GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

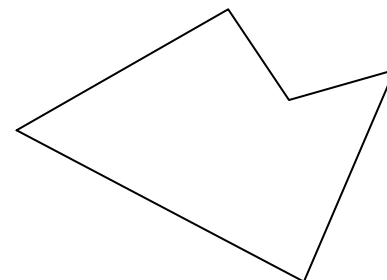
GL_QUAD_STRIP



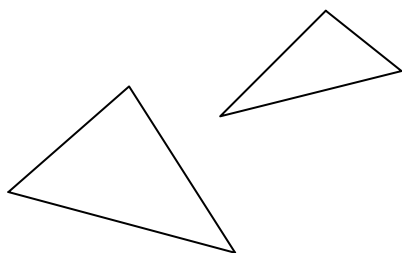
GL_LINES



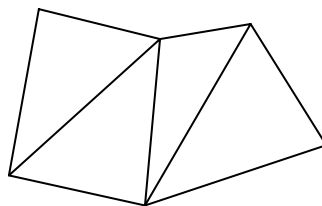
GL_LINE_STRIP



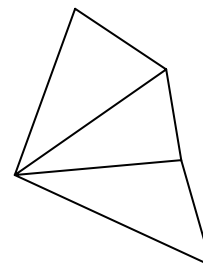
GL_LINE_LOOP



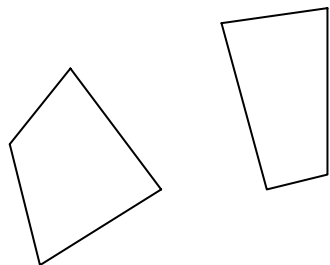
GL_TRIANGLES



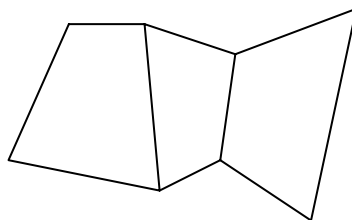
GL_TRIANGLE_STRIP



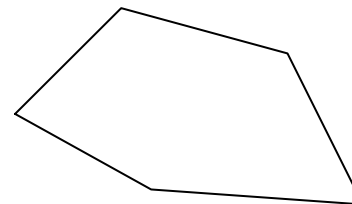
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

```
void main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode(mode);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(resize);
    glutKeyboardFunc(key);
    glutIdleFunc(idle);
    glutMainLoop();
}
```

```
void init(void);
{
    glClearColor(0.0,0.0,0.0,1.0);
    glClearDepth(1.0);
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
}
```

```

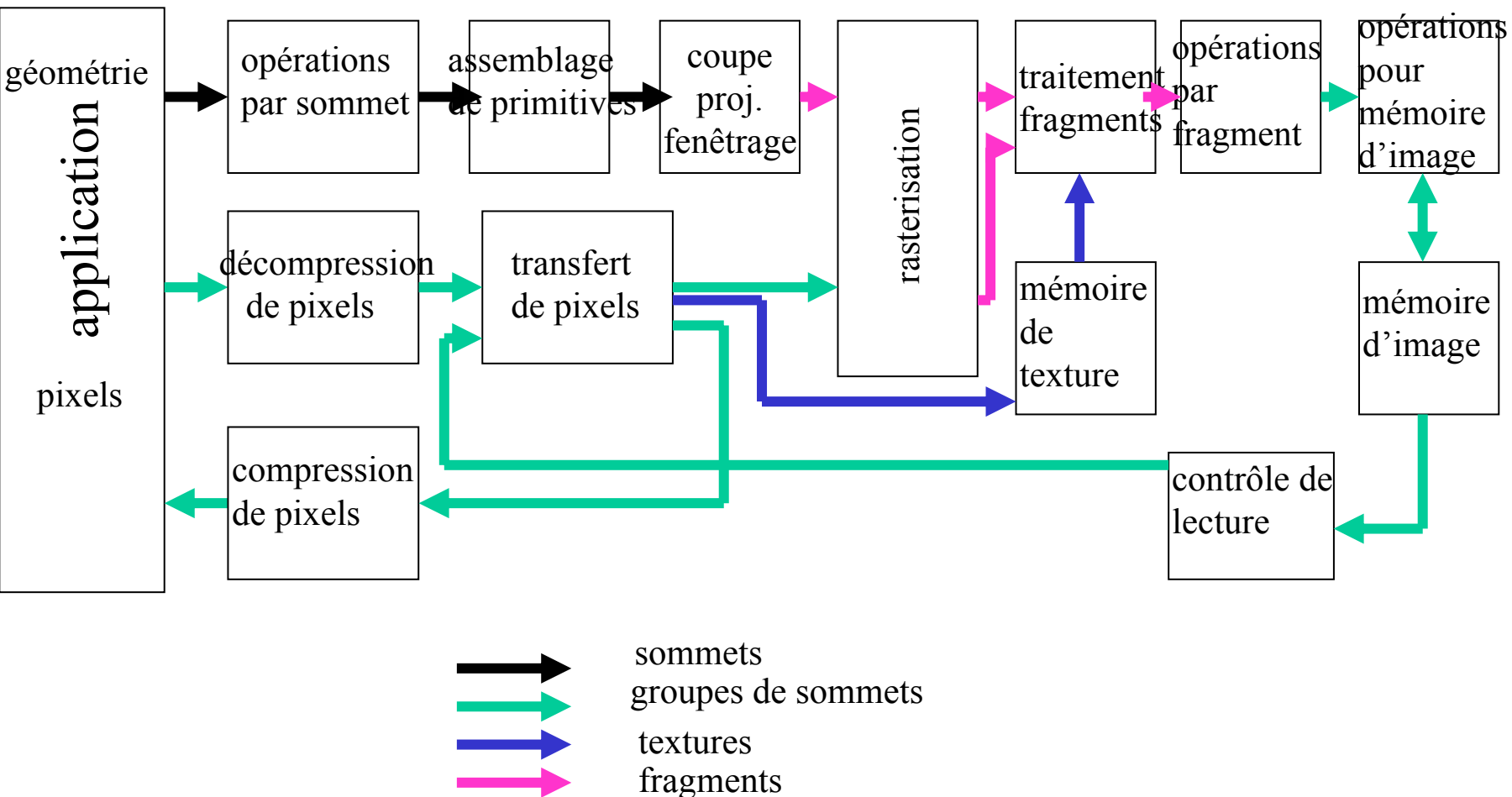
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLE_STRIP);
        glVertex3fv(v[0]);
        glVertex3fv(v[1]);
        glVertex3fv(v[2]);
        glVertex3fv(v[3]);
    glEnd();
    glutSwapBuffers();
}

void keyboard(char key, int x, int y);
{
    switch(key)
    {
        case 'q' : case 'Q' :
            exit(EXIT_SUCCESS);
        case 'r' : case 'R' :
            rotate = GL_TRUE;
            break;
    }
}

```

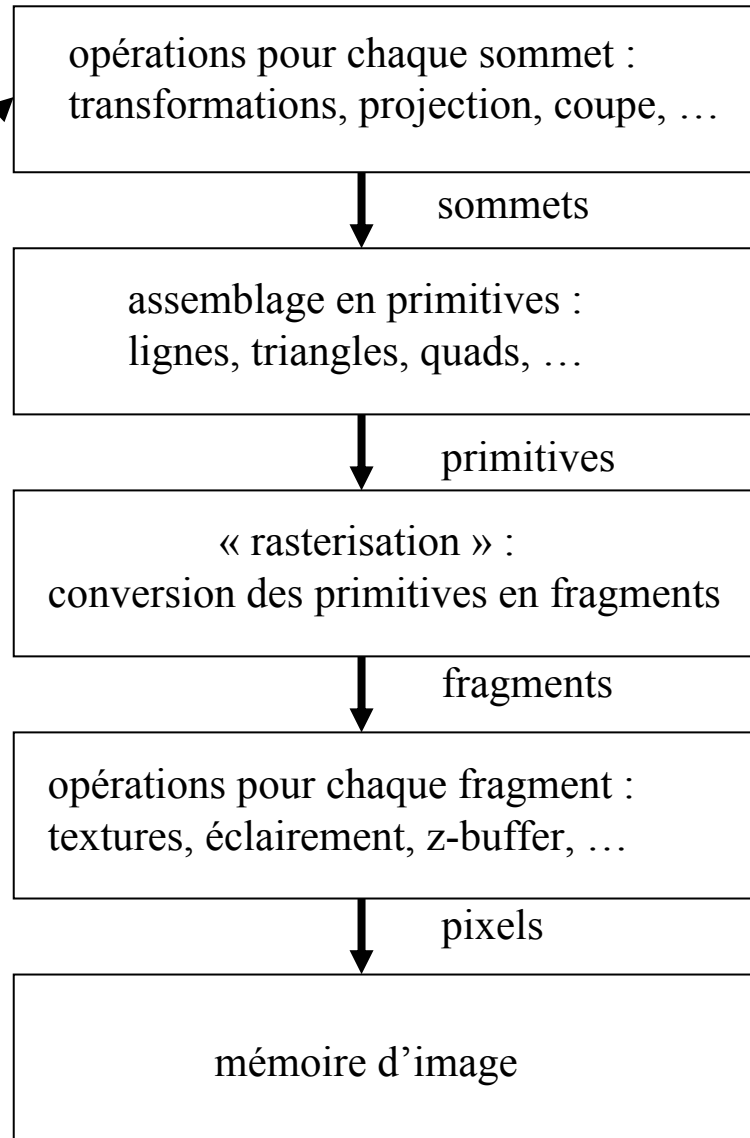
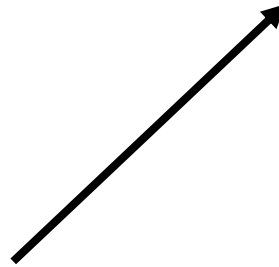
4. L'avenir : OpenGL 2.0

Rappel : architecture de OpenGL 1.x



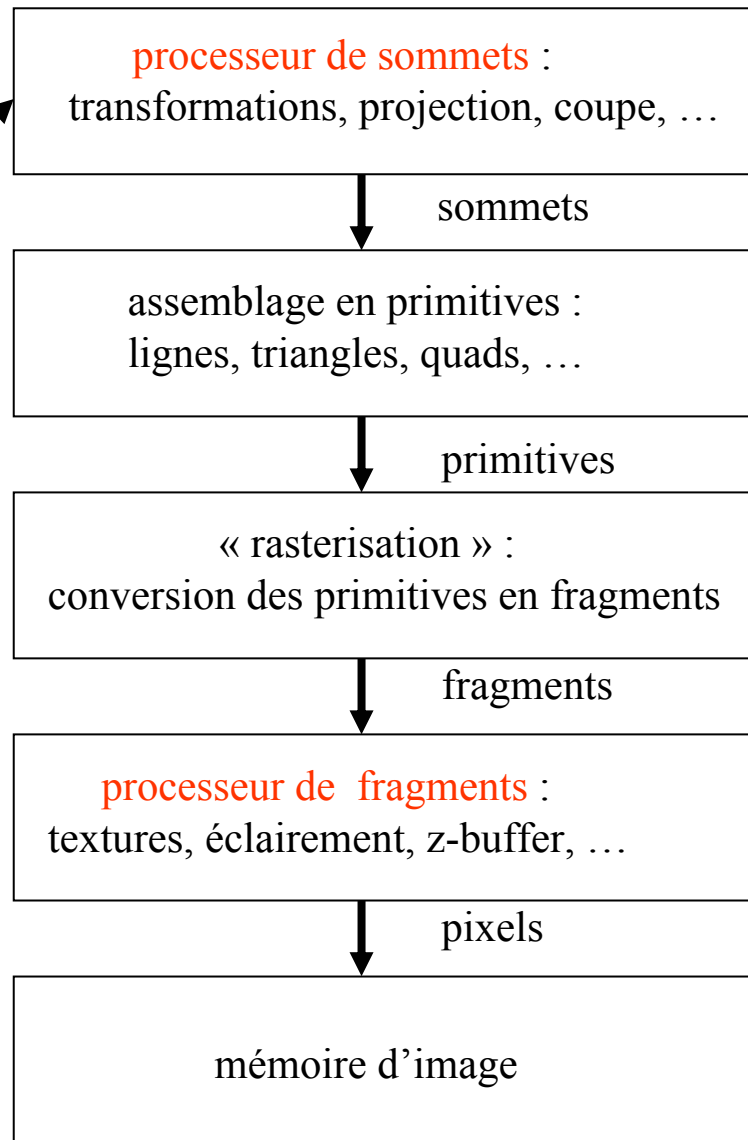
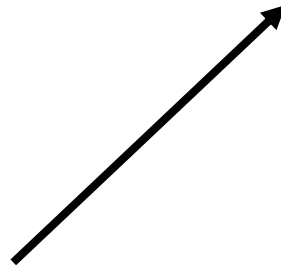
pipeline classique :

ensemble de
sommets fournis
par
l'application



⇒ pipeline programmable

ensemble de
sommets fournis
par
l'application



Possibilités des processeurs de sommets

- flexibilité pour éclairage, matériaux et géométrie
- processeur de sommets remplace :
 - transformations de sommets
 - transformations de normales, normalisation, ...
 - éclairage
 - application de la couleur des matériaux
 - « clamping » de couleurs
 - génération de coordonnées de texture
 - transformation de coordonnées de texture

Possibilités des processeurs de fragments

- flexibilité pour texturation et opérations portant sur les pixels
- processeur de sommets remplace :
 - opérations sur les valeurs interpolées
 - accès aux textures
 - application de textures
 - brouillard
 - somme de couleurs

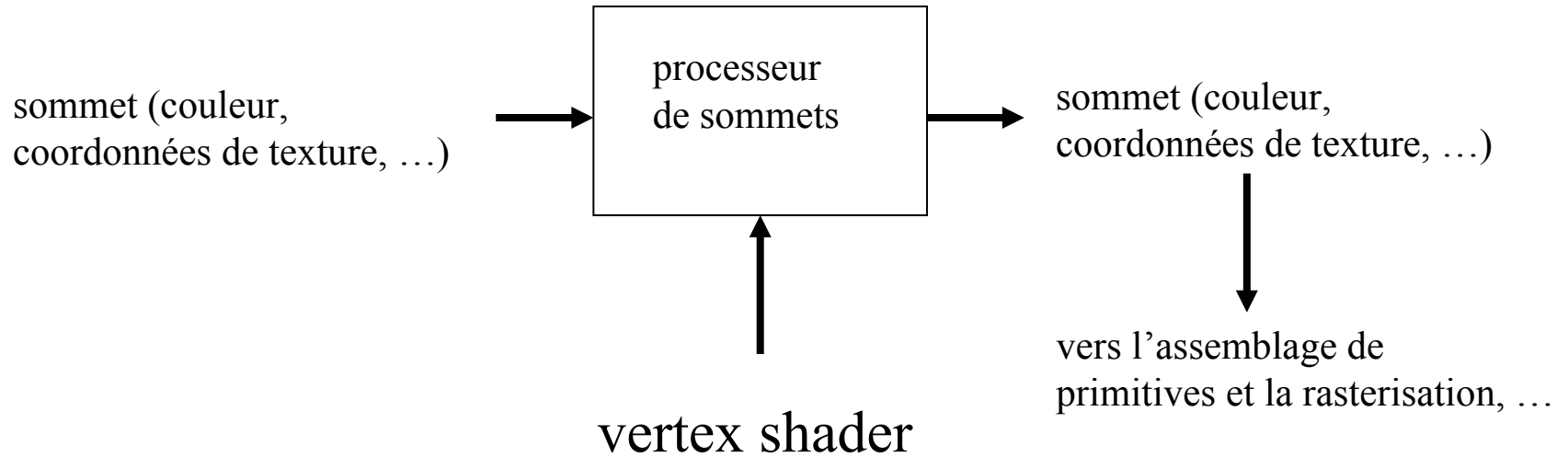
⇒ « shaders »

Shader = programme utilisateur exécuté dans le pipeline graphique lors d'étapes spécifiques

⇒ programmes du processeur de sommets
vertex shader

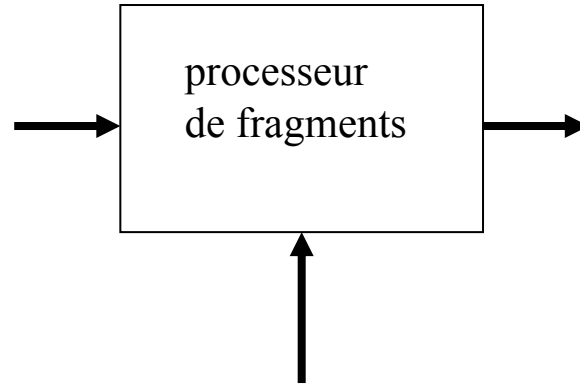
⇒ programmes du processeur de fragments
fragment (ou pixel) shader

vertex shader



pixel shader

fragment (couleur,
texture, position, ...) issu
de la rasterisation d'une
primitive



fragment (couleur,
texture, position, ...)

↓
vers la mémoire d'image

pixel shader

architecture de OpenGL 2.0

