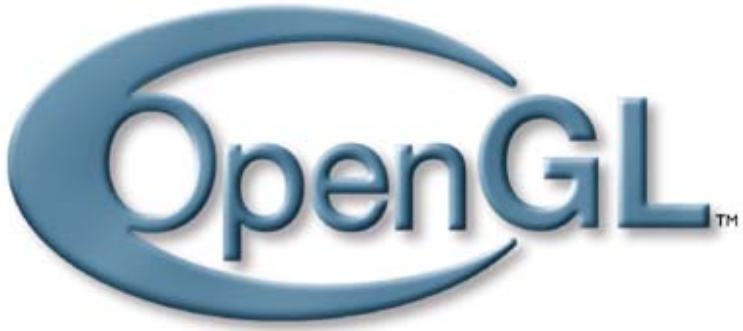



# Introduction to OpenGL

Ensimag 3D Graphics, 2013

# Planning

Available at

https://intranet.ensimag.fr/KIOSK/Matieres/4MMG3D/index.html

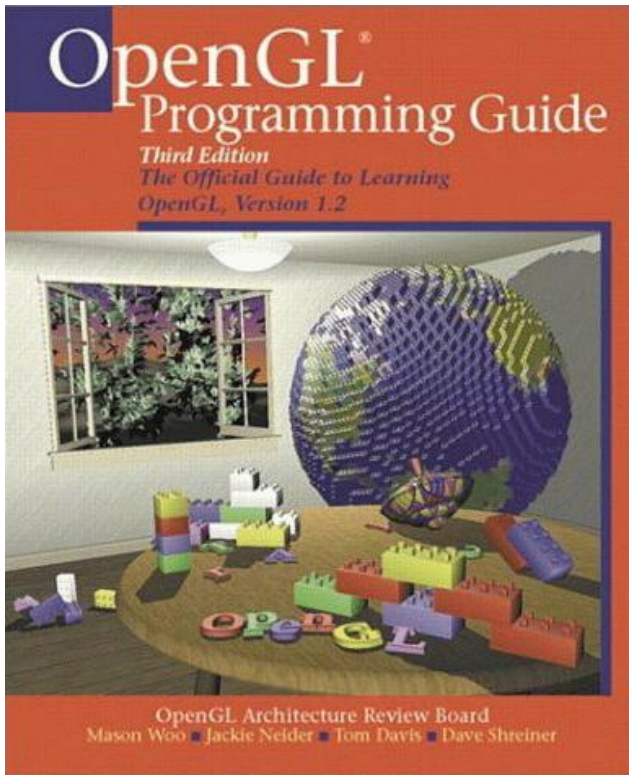
# Plan

# References

D. Shreiner, M. Woo, J. Neider, T. Davis
OpenGL Programming Guide

aka the red book

http://opengl-redbook.com



# Plan

# What is it?

- API (Application Programming Interface) for graphics hardware
- Non-dependant on the architecture or programming language
- Developped in 1989 (GL) by Silicon Graphics, extended to other architectures in 1993 (OpenGL)
- About 250 commands

# Things it can NOT do

- Can NOT create nor manage a *viewer*
- Can NOT manage complex objects : only 3 types of geometric primitives (points, lines, polygons)

# Things it can NOT do

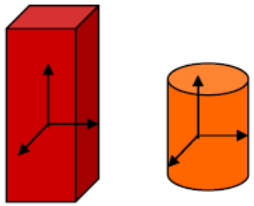
- Can NOT create nor manage a *viewer*
- Can NOT manage complex objects : only 3 types of geometric primitives (points, lines, polygons)

- $\Rightarrow$ **Additional libraries needed :**
  - **GLU :** *openGL Utility library* : more complex 3D models
  - **GLUT :** *openGL Utility Toolkit* : viewer
  - **QGLViewer :** *Qt library handling OpenGL*
  - ...

# Plan

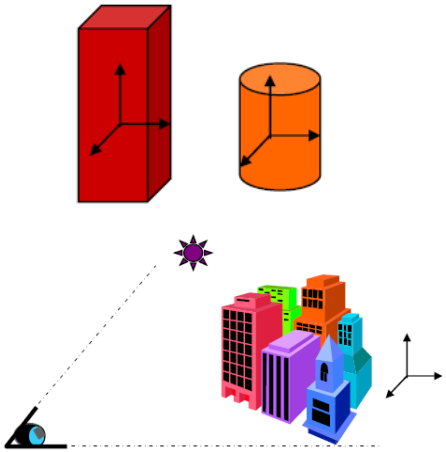
# Plan

# Graphics Pipeline



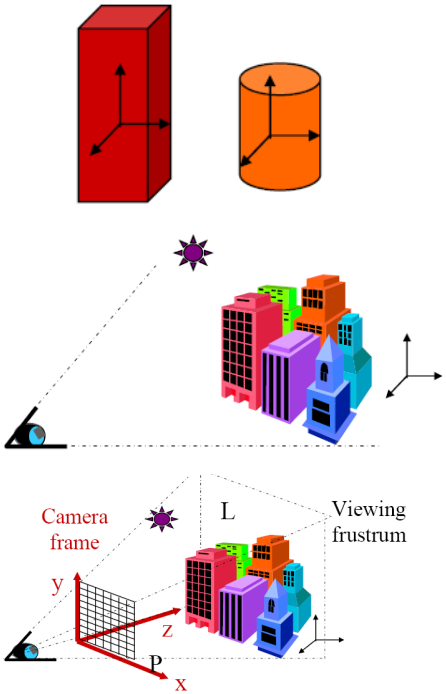
1. Create 3D models (modeling)

# Graphics Pipeline



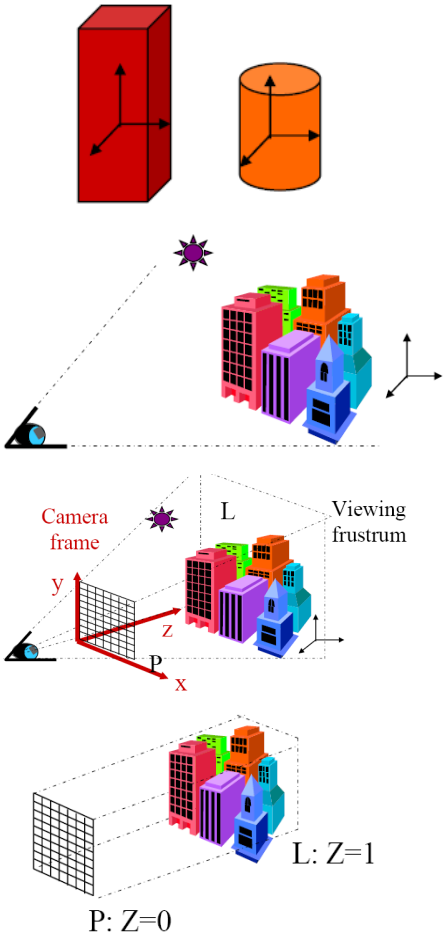
1. Create 3D models (modeling)
2. Build the scene from instances of models placed in a world frame (modeling transformation)

# Graphics Pipeline



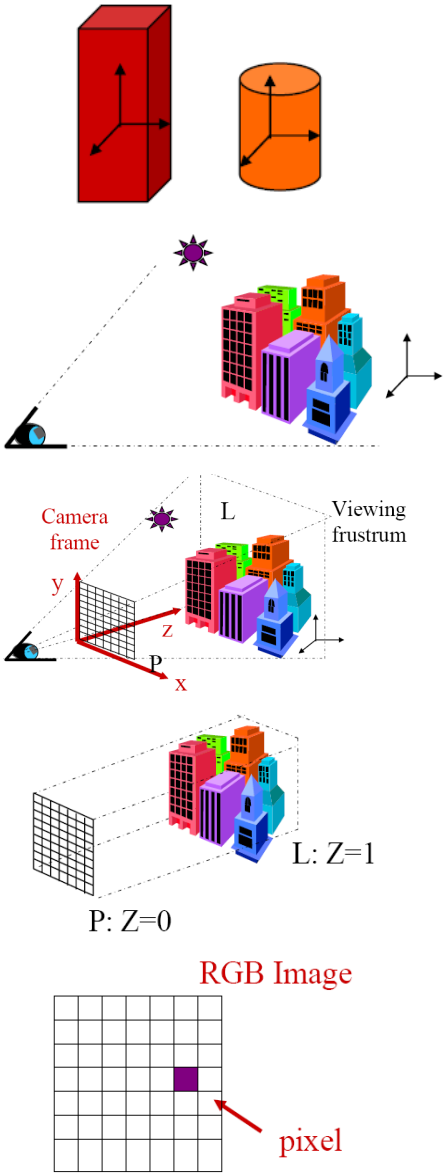

1. Create 3D models (modeling)
2. Build the scene from instances of models placed in a world frame (modeling transformation)
3. Convert to camera frame (culling, frustrum)

# Graphics Pipeline




1. Create 3D models (modeling)
2. Build the scene from instances of models placed in a world frame (modeling transformation)
3. Convert to camera frame (culling, frustrum)
4. Convert to screen frame (projection)

# Graphics Pipeline




1. Create 3D models (modeling)
2. Build the scene from instances of models placed in a world frame (modeling transformation)
3. Convert to camera frame (culling, frustrum)
4. Convert to screen frame (projection)
5. Compute image (rasterization)

# Plan

# OpenGL Pipeline

Create OpenGL context

# OpenGL Pipeline

Create OpenGL context
Loop :

# OpenGL Pipeline

Create OpenGL context
Loop :

1. Manage mouse/keyboard events

# OpenGL Pipeline

Create OpenGL context
Loop :

1. Manage mouse/keyboard events
2. Display

# OpenGL Pipeline

Create OpenGL context
Loop :

1. Manage mouse/keyboard events
2. Display
    1. Clear screen

# OpenGL Pipeline

Create OpenGL context
Loop :

1. Manage mouse/keyboard events
2. Display
    1. Clear screen
    2. Viewpoint

# OpenGL Pipeline

Create OpenGL context
Loop :

1. Manage mouse/keyboard events
2. Display
    1. Clear screen
    2. Viewpoint
    3. For each object :

# OpenGL Pipeline

Create OpenGL context
Loop :

1. Manage mouse/keyboard events
2. Display
    1. Clear screen
    2. Viewpoint
    3. For each object :
        1. Place object

# OpenGL Pipeline

Create OpenGL context
Loop :

1. Manage mouse/keyboard events
2. Display
    1. Clear screen
    2. Viewpoint
    3. For each object :
        1. Place object
        2. Draw

# It's a state machine!

State machine = each parameter retains its value and is used with that value until being explicitely changed

# It's a state machine!

State machine = each parameter retains its value and is used with that value until being explicitely changed

Parameters can be :

- **modes :** shading mode, matrix manipulated . . .

# It's a state machine!

State machine = each parameter retains its value and is used with that value until being explicitely changed

Parameters can be :

- **modes :** shading mode, matrix manipulated ...
- **booleans :** lights on/off, blend colors, ...

# It's a state machine!

State machine = each parameter retains its value and is used with that value until being explicitely changed

Parameters can be :

- **modes :** shading mode, matrix manipulated ...
- **booleans :** lights on/off, blend colors, ...
- **scalar values :** colors, viewpoint, ...

# Plan

# OpenGL Syntax - 1/2

Reminder :

- **modes :** `gl[MODE]Mode(GL_VALUE)`

# OpenGL Syntax - 1/2

Reminder :

- **modes :** `gl[MODE]Mode(GL_VALUE)`
- **booleans :** `glEnable(GL_VALUE)`

# OpenGL Syntax - 1/2

Reminder :

- **modes :** `gl[MODE]Mode(GL_VALUE)`
- **booleans :** `glEnable(GL_VALUE)`
  $\Rightarrow$ OpenGL constants start with **GL_**

# OpenGL Syntax - 1/2

Reminder :

- **modes :** `gl[MODE]Mode(GL_VALUE)`
- **booleans :** `glEnable(GL_VALUE)`
  $\Rightarrow$ OpenGL constants start with **GL_**
- **scalar values :**

```
glColor3f(1.0,1.0,1.0);
```

  - **gl** : OpenGL command ...

# OpenGL Syntax - 1/2

Reminder :

- **modes :** `gl[MODE]Mode(GL_VALUE)`
- **booleans :** `glEnable(GL_VALUE)`
  $\Rightarrow$ OpenGL constants start with **GL_**
- **scalar values :**

```
glColor3f(1.0,1.0,1.0);
```

  - **gl** : OpenGL command ...
  - **3** : ... that has 3 arguments ...

# OpenGL Syntax - 1/2

Reminder :

- **modes :** `gl[MODE]Mode(GL_VALUE)`
- **booleans :** `glEnable(GL_VALUE)`
  $\Rightarrow$ OpenGL constants start with **GL_**
- **scalar values :**

```
glColor3f(1.0,1.0,1.0);
```

  - **gl** : OpenGL command ...
  - **3** : ... that has 3 arguments ...
  - **f** : ... of type float.

# OpenGL Syntax - 1/2

Reminder :

- **modes :** `gl[MODE]Mode(GL_VALUE)`
- **booleans :** `glEnable(GL_VALUE)`
  $\Rightarrow$ OpenGL constants start with **GL_**
- **scalar values :**

```
glColor3f(1.0,1.0,1.0);
```

  - **gl** : OpenGL command ...
  - **3** : ... that has 3 arguments ...
  - **f** : ... of type float.

```
glColor3fv(color_array);
```

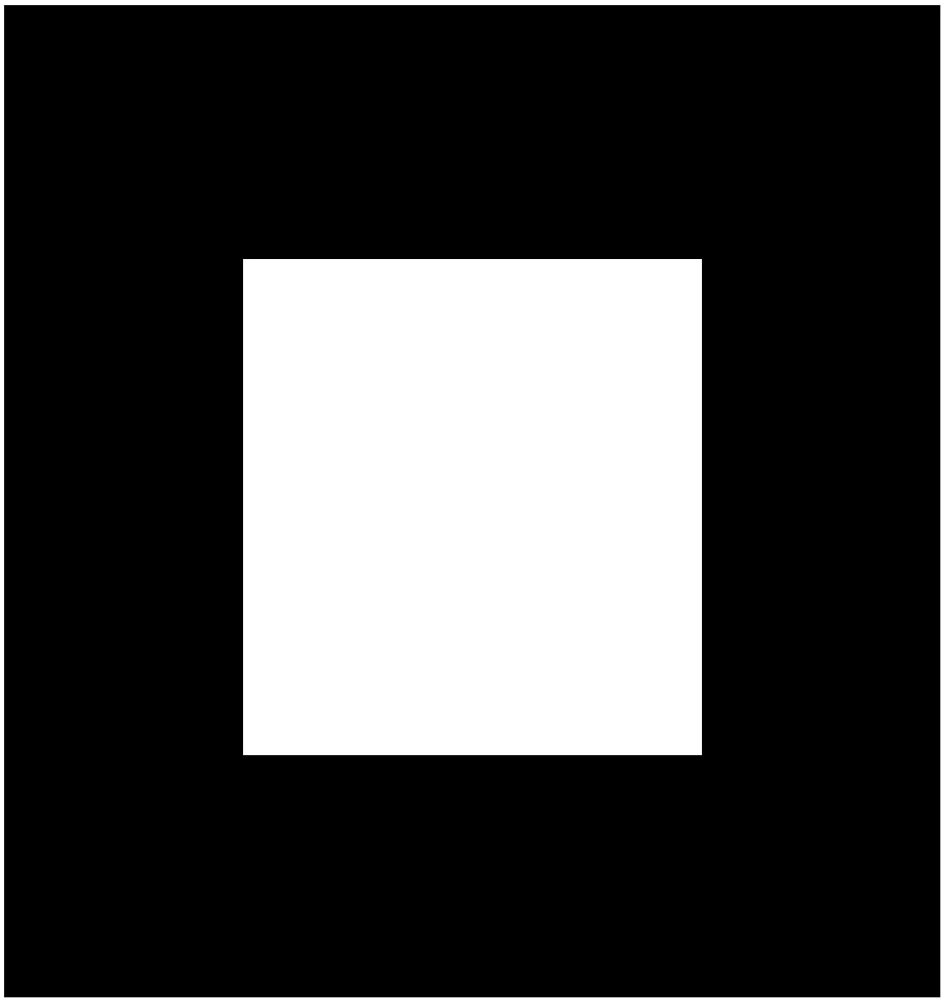
$\Rightarrow$ The argument is a **v**ector (or array) of 3 floats
(`GLfloat color_array[] = { 1.0,0.0,0.0 } ;` )

# OpenGL Syntax - 2/2

## OpenGL suffixes and types

| | | | |
|---|---|---|---|
| b | integer (8 bits) | signed char | GLbyte |
| s | integer (16 bits) | short | GLshort |
| i | integer (32 bits) | int ou long | GLint |
| f | real (32 bits) | float | GLfloat |
| d | real (64 bits) | double | GLdouble |
| ub | unsigned integer (8 bits) | unsigned char | GLubyte |
| us | unsigned integer (16 bits) | unsigned long | GLushort |
| ul | unsigned integer (32 bits) | unsigned int ou long | GLuint |

# Basic example - the Square



# Basic example - the Code

Display :

# Basic example - the Code

Display :

```
void display () {
```

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen

```
    glClear(GL_COLOR_BUFFER_BIT);
```

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen

   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```

2. Viewpoint

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen

```
glClear(GL_COLOR_BUFFER_BIT);
```

2. Viewpoint

```
glMatrixMode(GL_PROJECTION);
glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
```

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :
   1. Place object

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :
   1. Place object

   ```
   glMatrixMode(GL_MODELVIEW);
   ```

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :
   1. Place object
      ```
      glMatrixMode(GL_MODELVIEW);
      ```
   2. Modify state machine

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :
   1. Place object
      ```
      glMatrixMode(GL_MODELVIEW);
      ```
   2. Modify state machine
      ```
      glColor3f(1.0,1.0,1.0);
      ```

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :
   1. Place object
      ```
      glMatrixMode(GL_MODELVIEW);
      ```
   2. Modify state machine
      ```
      glColor3f(1.0,1.0,1.0);
      ```
   3. Draw

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :
   1. Place object
      ```
      glMatrixMode(GL_MODELVIEW);
      ```
   2. Modify state machine
      ```
      glColor3f(1.0,1.0,1.0);
      ```
   3. Draw
      ```
      glBegin(GL_POLYGON);
          glVertex3f(0.25,0.25,0.0);
          glVertex3f(0.75,0.25,0.0);
          glVertex3f(0.75,0.75,0.0);
          glVertex3f(0.25,0.75,0.0);
      glEnd();
      ```

# Basic example - the Code

Display :

```
void display () {
```

1. Clear screen
   ```
   glClear(GL_COLOR_BUFFER_BIT);
   ```
2. Viewpoint
   ```
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);
   ```
3. For each object :
   1. Place object
      ```
      glMatrixMode(GL_MODELVIEW);
      ```
   2. Modify state machine
      ```
      glColor3f(1.0,1.0,1.0);
      ```
   3. Draw
      ```
      glBegin(GL_POLYGON);
          glVertex3f(0.25,0.25,0.0);
          glVertex3f(0.75,0.25,0.0);
          glVertex3f(0.75,0.75,0.0);
          glVertex3f(0.25,0.75,0.0);
      glEnd();
      glFlush(); // Execute OpenGL commands in hold
      ```

```
}
```

# Plan

# Plan

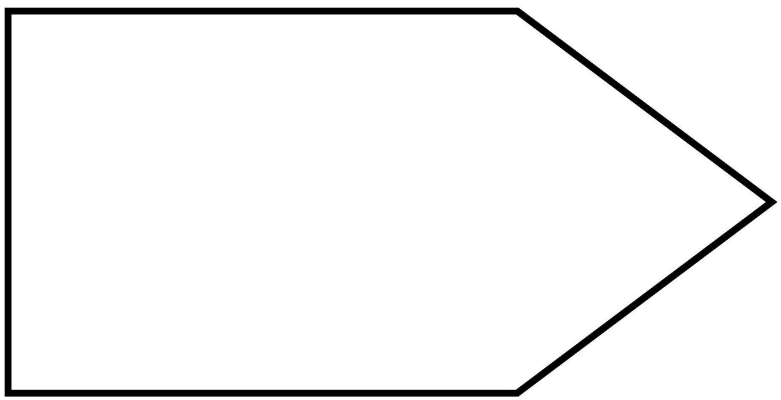
# Procedural modeling

Complex object : combination of elementary elements :

1. Points (vertices) : coordinates in a given reference frame
2. Lines : segments
3. Polygons : simple convex polygons

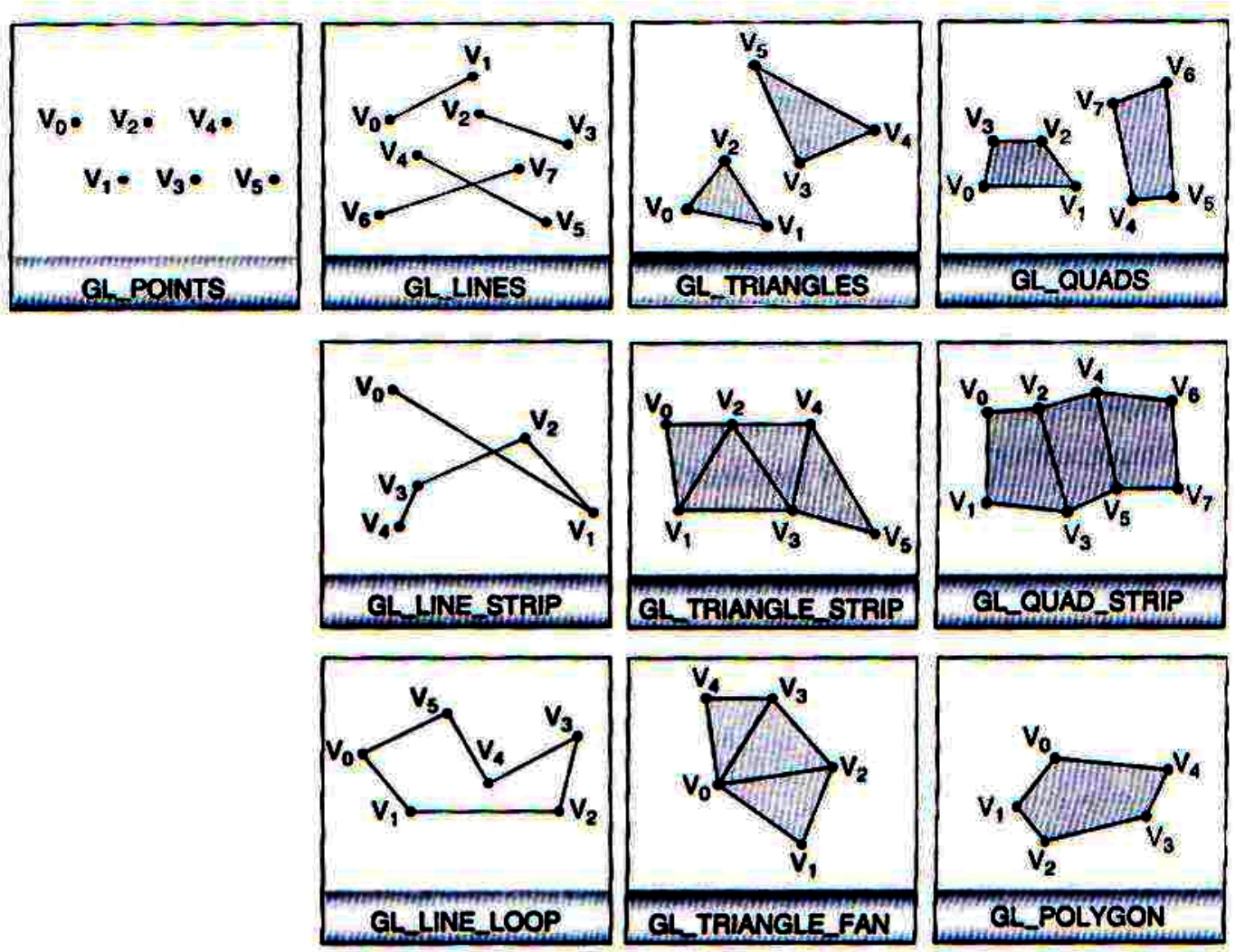
# Example - planar pentagon

```
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();
glFlush();
```



# Plan

# OpenGL primitives



# Parameters

- Point size (in pixels) : `glPointSize(2.0);`
- Line width (in pixels) : `glLineWidth(3.0);`

# Parameters

- Point size (in pixels) : `glPointSize(2.0);`
- Line width (in pixels) : `glLineWidth(3.0);`
- Line drawing : many stippling styles
- Different renderings for front and back faces :

```
glPolygonMode(GL_FRONT,GL_FILL);
glPolygonMode(GL_BACK,GL_LINE);
```

# Parameters

- Point size (in pixels) : `glPointSize(2.0);`
- Line width (in pixels) : `glLineWidth(3.0);`
- Line drawing : many stippling styles
- Different renderings for front and back faces :
  `glPolygonMode(GL_FRONT,GL_FILL);`
  `glPolygonMode(GL_BACK,GL_LINE);`
- Culling : `glCullFace(GL_BACK);` : back-faces non-visible

# Parameters

- Point size (in pixels) : `glPointSize(2.0);`
- Line width (in pixels) : `glLineWidth(3.0);`
- Line drawing : many stippling styles
- Different renderings for front and back faces :
  `glPolygonMode(GL_FRONT,GL_FILL);`
  `glPolygonMode(GL_BACK,GL_LINE);`
- Culling : `glCullFace(GL_BACK);` : back-faces non-visible
- The color, normal, . . . , at each vertex can be specified
- . . .

# Parameters

- Point size (in pixels) : `glPointSize(2.0);`
- Line width (in pixels) : `glLineWidth(3.0);`
- Line drawing : many stippling styles
- Different renderings for front and back faces :
  `glPolygonMode(GL_FRONT,GL_FILL);`
  `glPolygonMode(GL_BACK,GL_LINE);`
- Culling : `glCullFace(GL_BACK);` : back-faces non-visible
- The color, normal, . . . , at each vertex can be specified
- . . .

- Get current values : `glGetFloatv(GL_LINE_WIDTH);`

# Immediate definition of an objet

Each vertices and polygons are directly defined.
Example of a triangle :

```
glBegin(GL_POLYGON);
   glNormal3fv(n0);
   glVertex3fv(v0);
   glNormal3fv(n1);
   glVertex3fv(v1);
   glNormal3fv(n2);
   glVertex3fv(v2);
glEnd();
```

Beware of the order : parameter (i.e. normal) before coordinates (state machine)

# Non immediate methods : Arrays

- Vertex-related data (coordinates, normals, colors, ...) can also be stored in arrays in the CPU memory

# Non immediate methods : Arrays

- Vertex-related data (coordinates, normals, colors, ...) can also be stored in arrays in the CPU memory
- Polygons refer to vertices through indices in these arrays

# Non immediate methods : Arrays

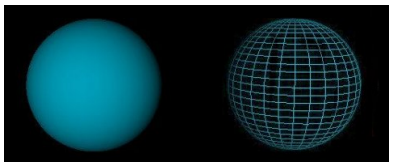
- Vertex-related data (coordinates, normals, colors, ...) can also be stored in arrays in the CPU memory
- Polygons refer to vertices through indices in these arrays
- Object are then defined with a reduced number of primitives
  - `glDrawArrays(GL_QUADS, 0, 24)`
  - `glDrawElements(GL_POLYGON, 5, GL_UNSIGNED_INT, vertices)`

# Non immediate methods : Arrays

- Vertex-related data (coordinates, normals, colors, ...) can also be stored in arrays in the CPU memory
- Polygons refer to vertices through indices in these arrays
- Object are then defined with a reduced number of primitives
  - `glDrawArrays(GL_QUADS, 0, 24)`
  - `glDrawElements(GL_POLYGON, 5, GL_UNSIGNED_INT, vertices)`
- Advanced methods (non studied in this course)
  - Display lists
  - Vertex Buffer Objects (VBO) : arrays are directly stored in the graphics card memory.

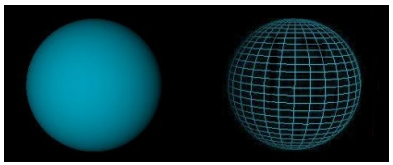
# Plan

# GLUT primitives - 1/2

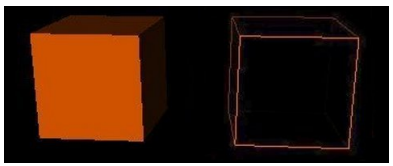


```
glutSolidSphere(radius, slices, stacks)
glutWireSphere(radius, slices, stacks)
```

# GLUT primitives - 1/2

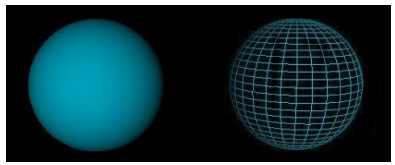


```
glutSolidSphere(radius, slices, stacks)
glutWireSphere(radius, slices, stacks)
```

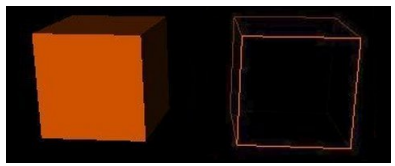


```
glutSolidCube(size)
glutWireCube(size)
```

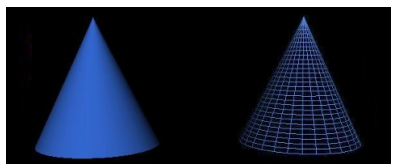
# GLUT primitives - 1/2



```
glutSolidSphere(radius, slices, stacks)
glutWireSphere(radius, slices, stacks)
```

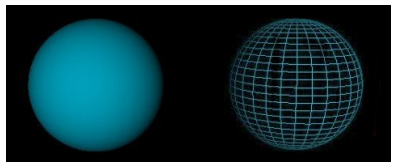


```
glutSolidCube(size)
glutWireCube(size)
```

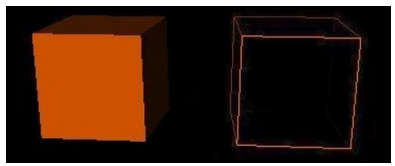


```
glutSolidCone(base, height, slices, stacks)
glutWireCone(base, height, slices, stacks)
```

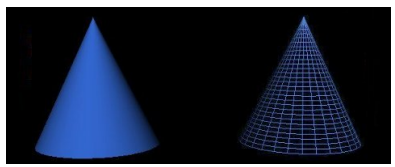
# GLUT primitives - 1/2



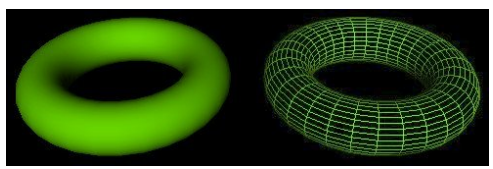
```
glutSolidSphere(radius, slices, stacks)
glutWireSphere(radius, slices, stacks)
```



```
glutSolidCube(size)
glutWireCube(size)
```



```
glutSolidCone(base, height, slices, stacks)
glutWireCone(base, height, slices, stacks)
```



```
glutSolidTorus(innerRadius, outerRadius,
nsides, rings)
glutWireTorus(innerRadius, outerRadius,
nsides, rings)
```

# GLUT primitives - 2/2

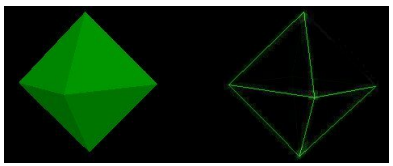
`glutSolidTetrahedron()`

`glutWireTetrahedron()`

# GLUT primitives - 2/2

`glutSolidTetrahedron()`

`glutWireTetrahedron()`



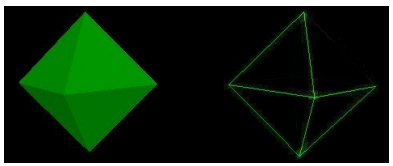
`glutSolidOctahedron()`

`glutWireOctahedron()`

# GLUT primitives - 2/2

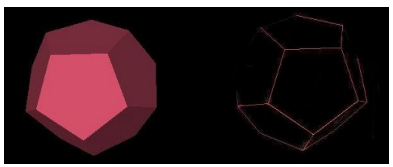
`glutSolidTetrahedron()`

`glutWireTetrahedron()`



`glutSolidOctahedron()`

`glutWireOctahedron()`



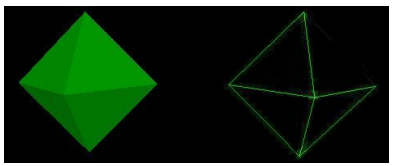
`glutSolidDodecahedron()`

`glutWireDodecahedron()`

# GLUT primitives - 2/2

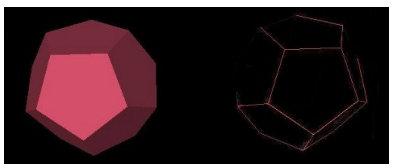
`glutSolidTetrahedron()`

`glutWireTetrahedron()`



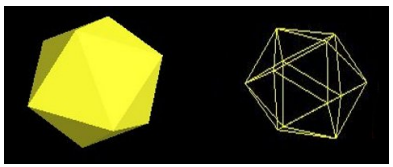
`glutSolidOctahedron()`

`glutWireOctahedron()`



`glutSolidDodecahedron()`

`glutWireDodecahedron()`



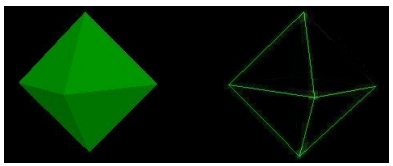
`glutSolidIcosahedron()`

`glutWireIcosahedron()`

# GLUT primitives - 2/2

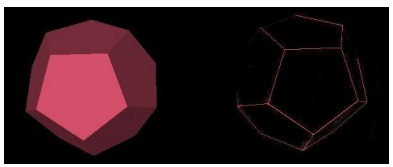
`glutSolidTetrahedron()`

`glutWireTetrahedron()`



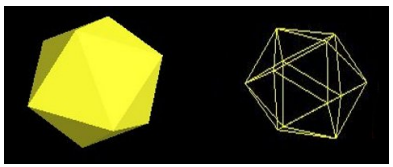
`glutSolidOctahedron()`

`glutWireOctahedron()`



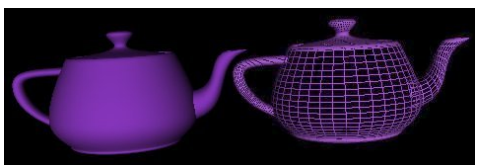
`glutSolidDodecahedron()`

`glutWireDodecahedron()`



`glutSolidIcosahedron()`

`glutWireIcosahedron()`



`glutSolidTeapot(size)`

`glutWireTeapot(size)`

Remark : There is no `Cylinder` primitive.

# Plan

# Conclusion :

- Done :
  - General process
  - Modeling : geometric primitives
- Highlights :
  - State machine
  - Primitives
  - the redbook
- To do :
  - lab
  - modeling complex objects with primitives