Rendering

Part 1
An introduction to OpenGL

Olivier Gourmel
VORTEX Team – IRIT
University of Toulouse

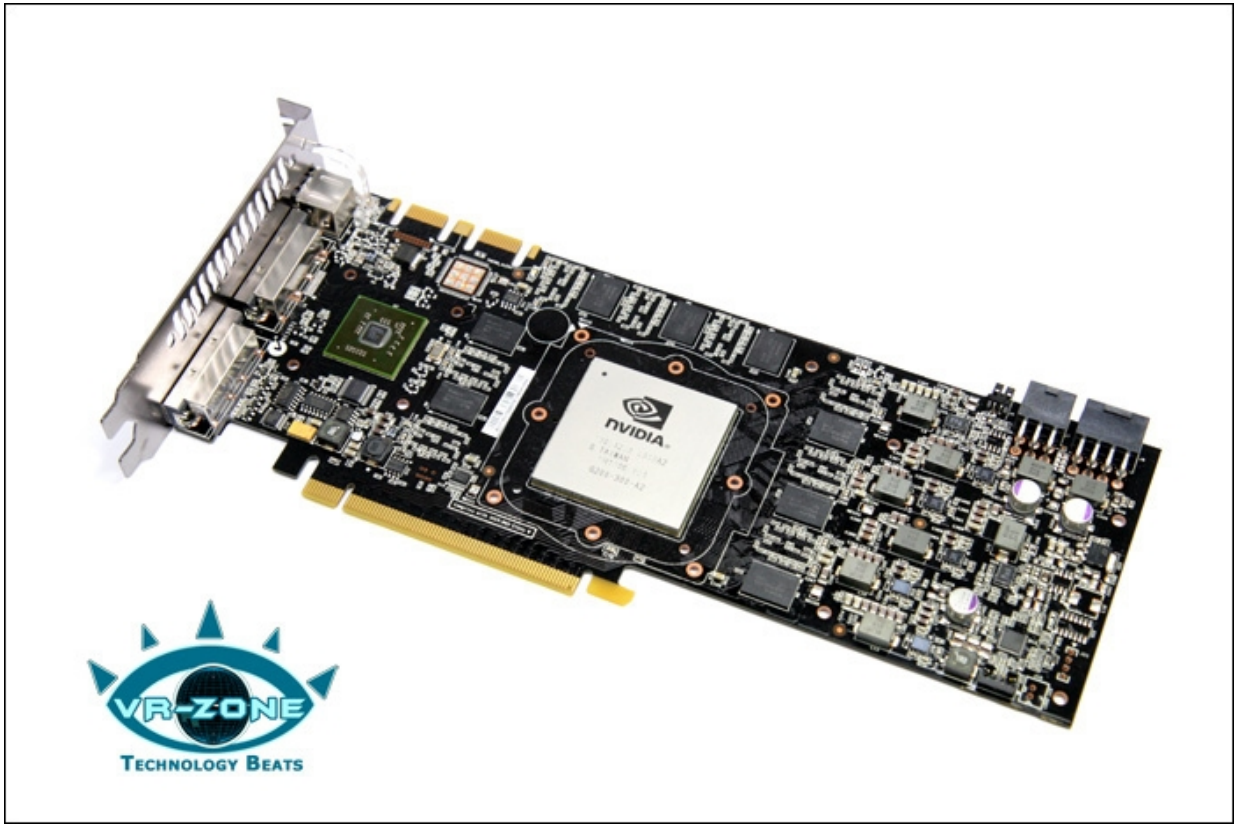
gourmel@irit.fr

VORTEX

Image synthesis

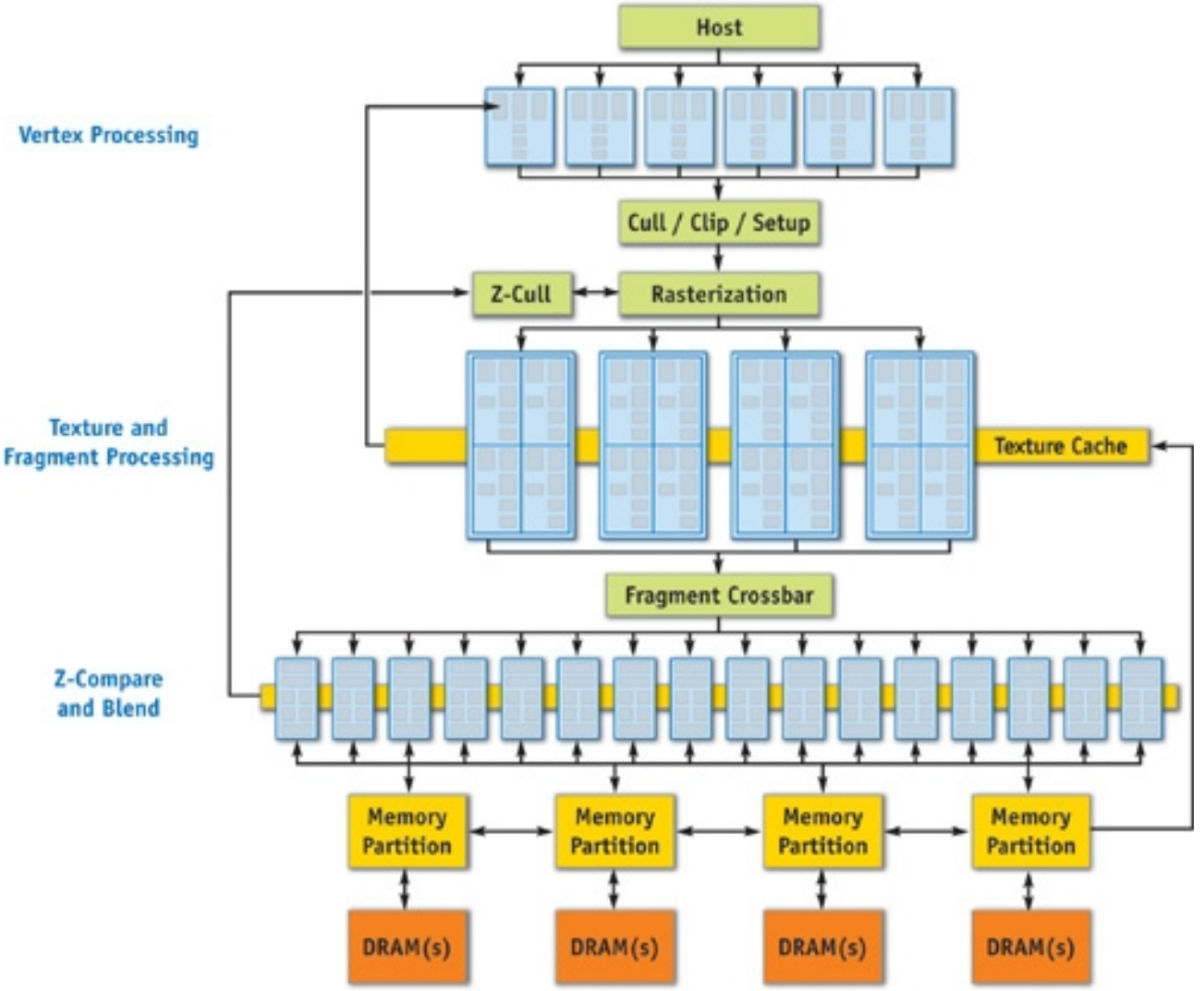
The Graphics Processing Unit (GPU):

- A highly parallel architecture specialized in the rasterization of objects composed of polygons (meshes), usually triangles.



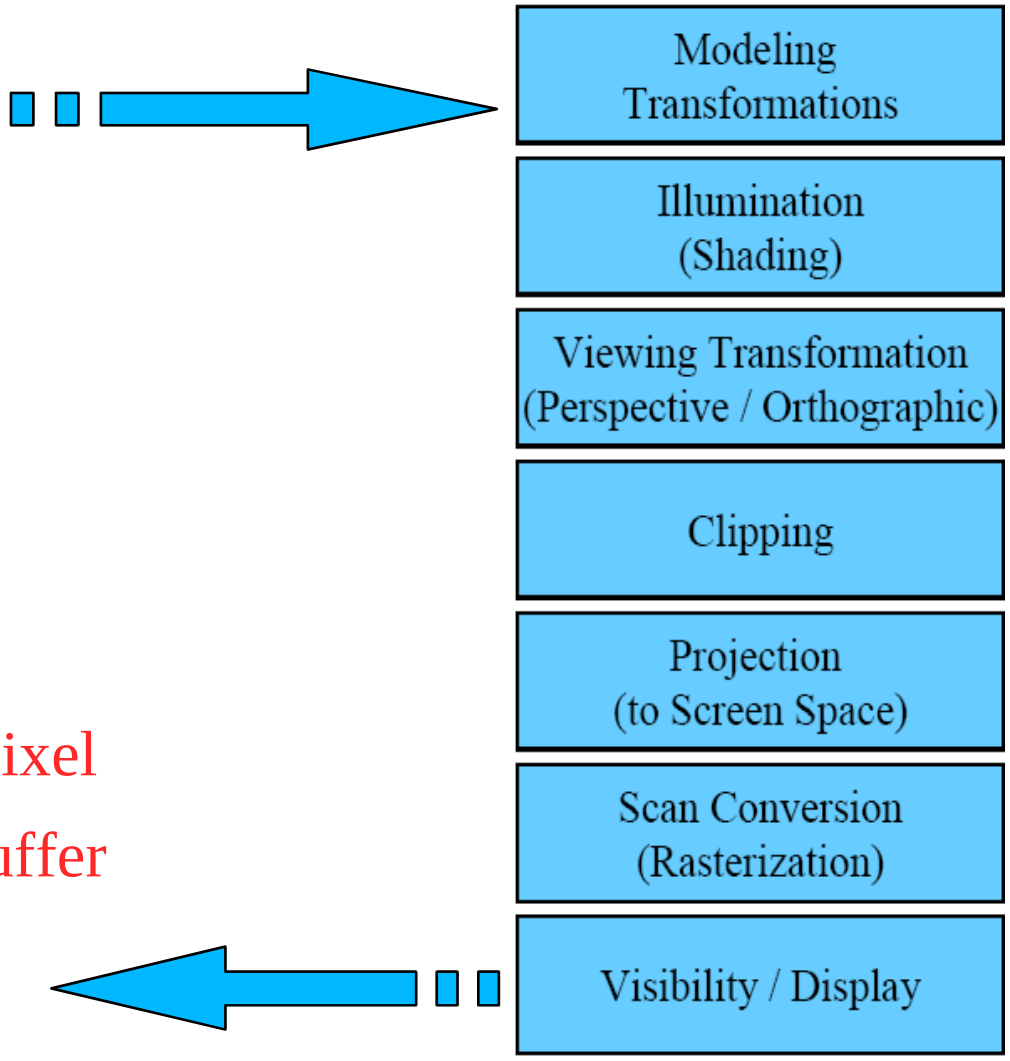
The GPU

- Architecture of a GPU (nVidia GeForce 6800)



The graphic pipeline

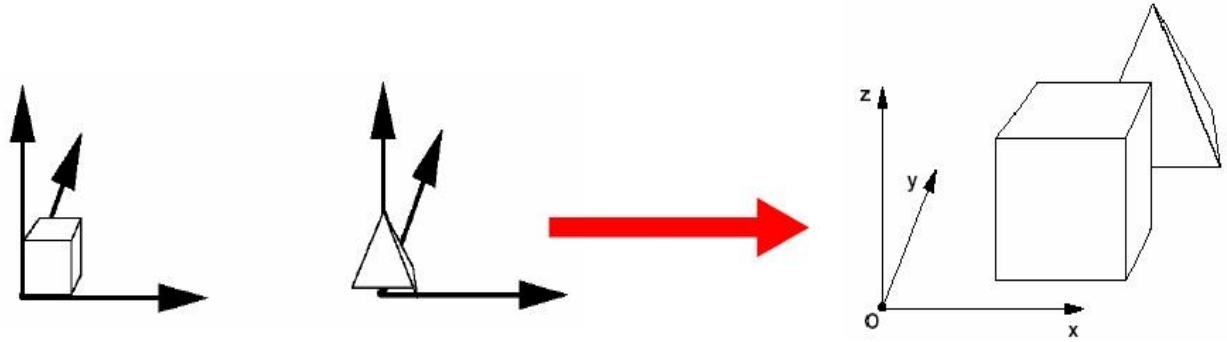
- Input:
  - Geometry
  - Lighting model
  - Camera Model
  - Output area



- Output:
  - Color value at each pixel
  - Stored in the framebuffer

The graphic pipeline

- 3D Objects:
  - Defined in their own frame (object space)
  - The object frame is located in the world space (with regards to world frame)



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

Projection (to Screen Space)

Scan Conversion (Rasterization)

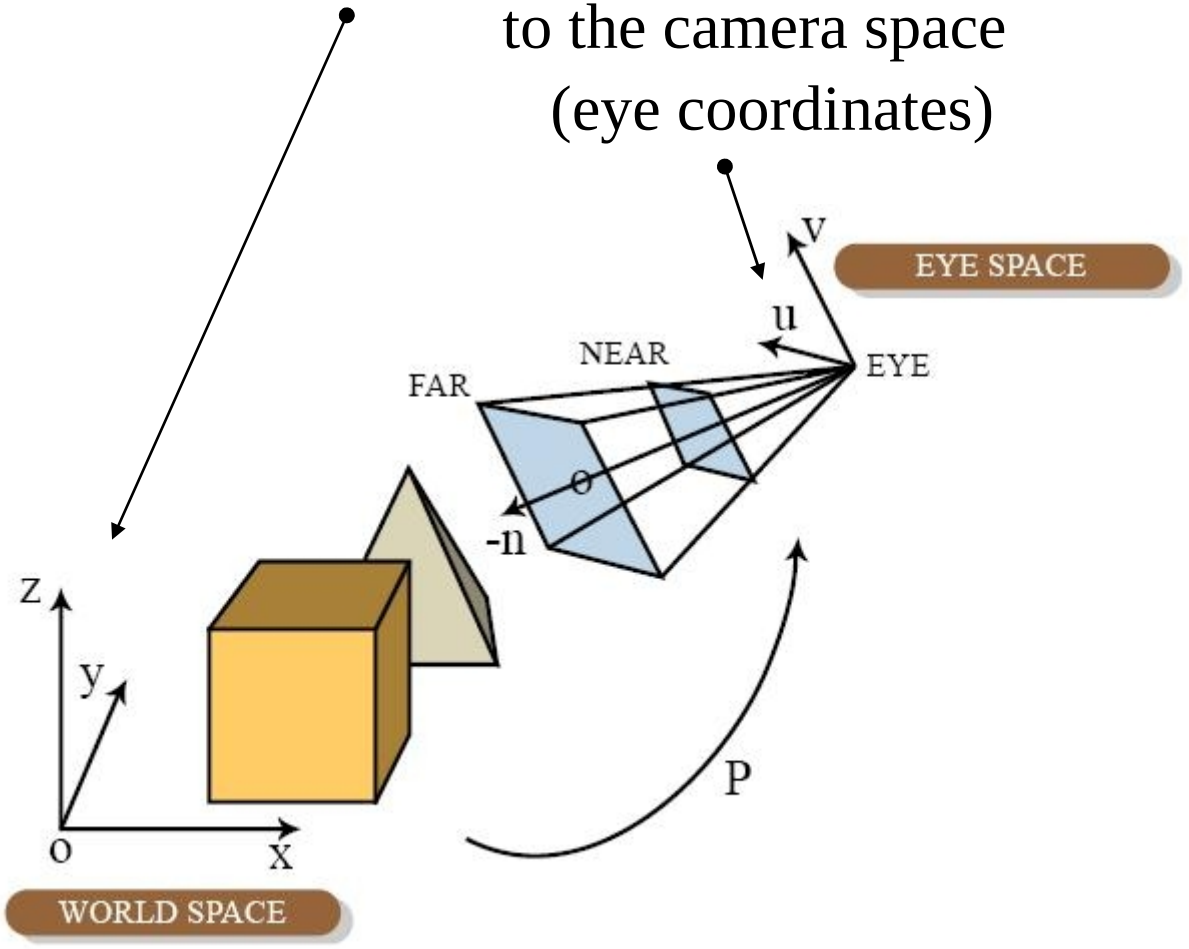
Visibility / Display

VORTEX

The graphic pipeline

Transforms vertex coordinates
from the world space

to the camera space
(eye coordinates)

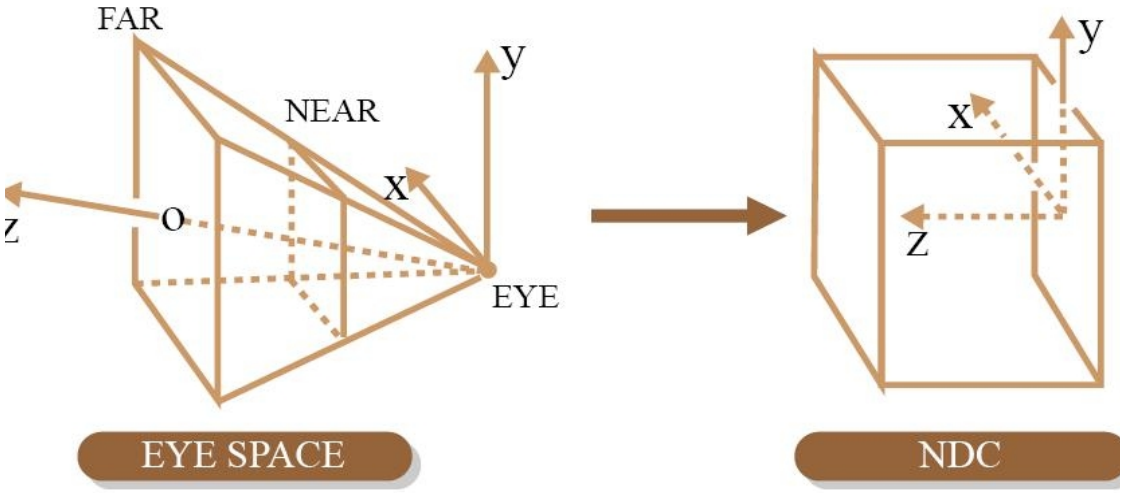


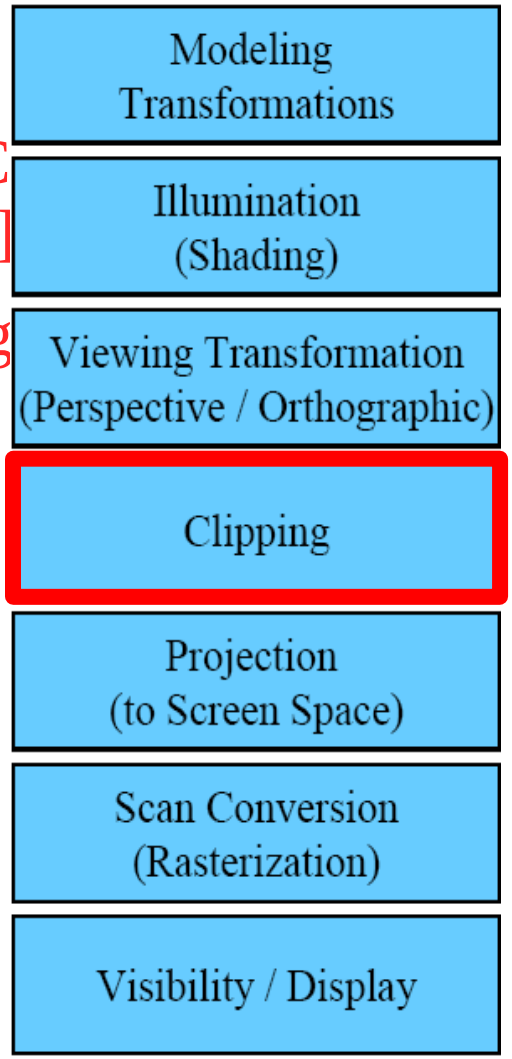
- Modeling Transformations
- Illumination (Shading)
- **Viewing Transformation (Perspective / Orthographic)**
- Clipping
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- Visibility / Display

VORTEX

The graphic pipeline

- Clipping :
  - Transforms eye coordinates to Normalized Device Coordinates (NDC) (maps each coordinates to range [-1, 1]
  - Suppresses geometry outside the viewing volume

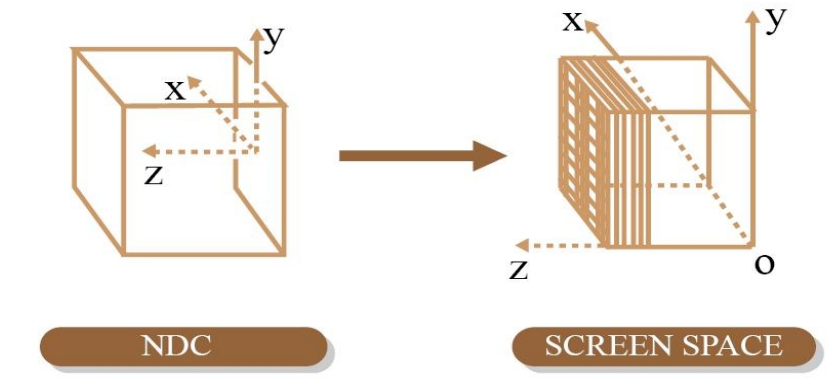


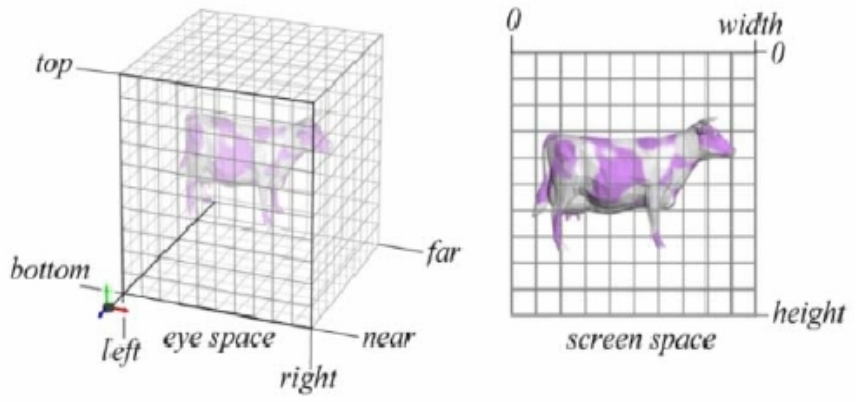


VORTEX

The graphic pipeline

- Projection to screen space (2D) :





Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

Projection (to Screen Space)

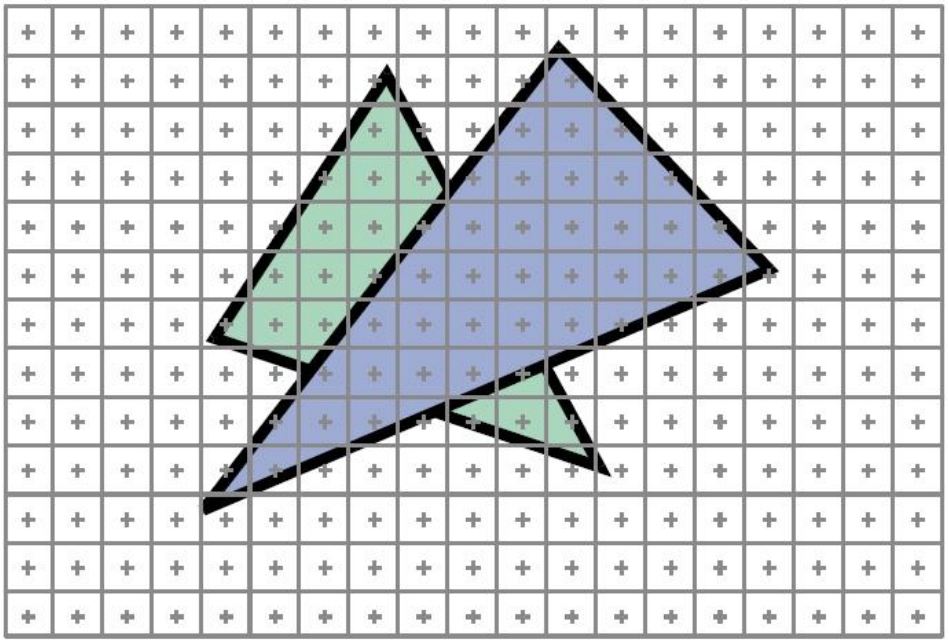
Scan Conversion (Rasterization)

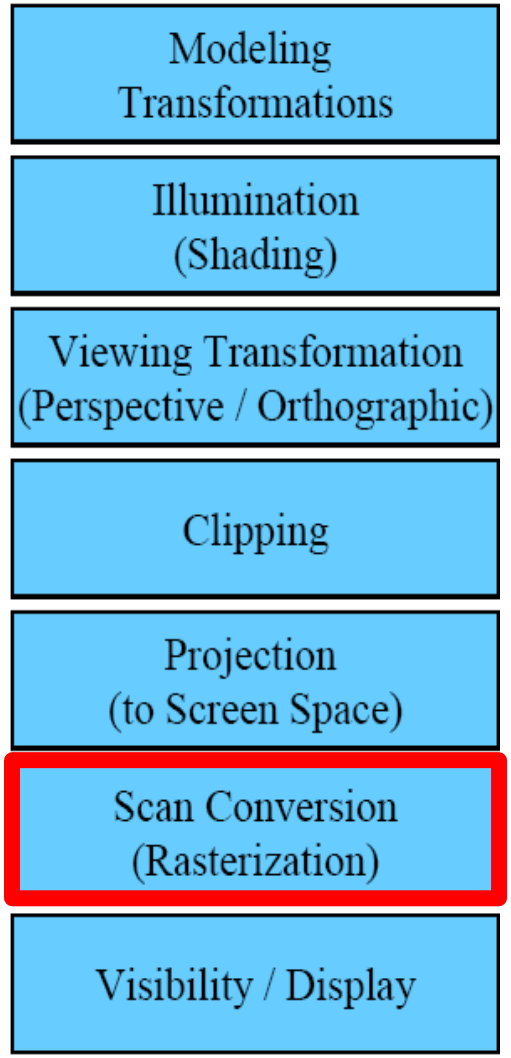
Visibility / Display

VORTEX

The graphic pipeline

- Rasterization : for each triangle
  - Computes the pixels overlapped by this triangle (fragments)
  - Interpolates the vertex attributes (color, normals, depth...) at each fragment

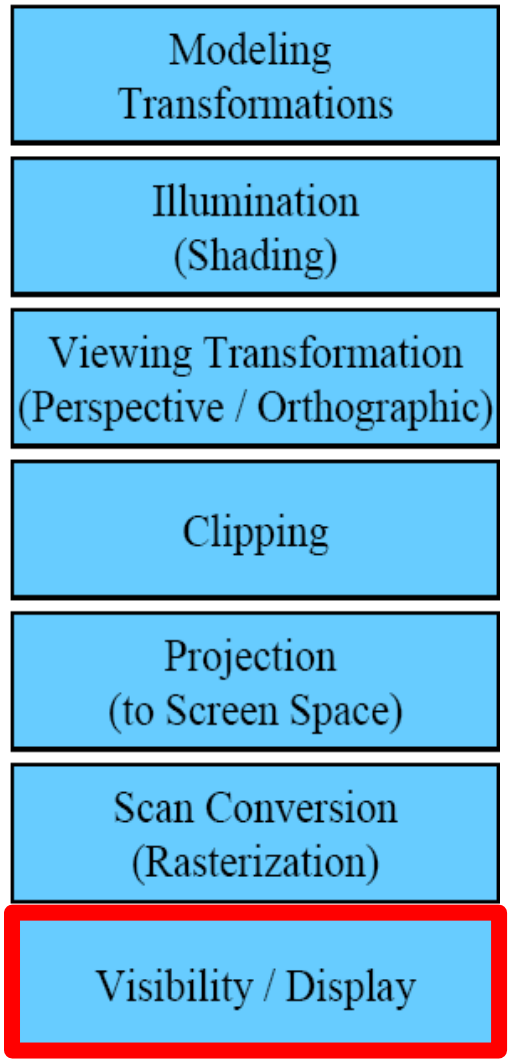




The graphic pipeline

- Eliminates hidden fragments using a *depth buffer*

- A depth buffer stores the depth of the last fragment written at each pixel.



The graphic pipeline

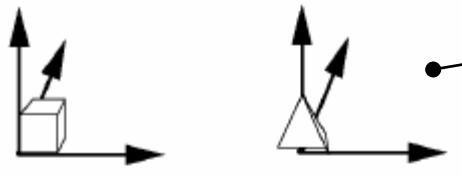
Object space

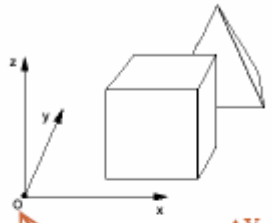
World space

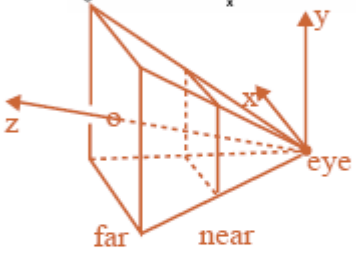
Eye space

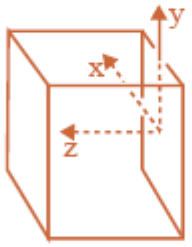
Normalized device coordinates

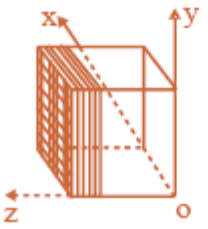
Screen space











Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

Projection (to Screen Space)

Scan Conversion (Rasterization)

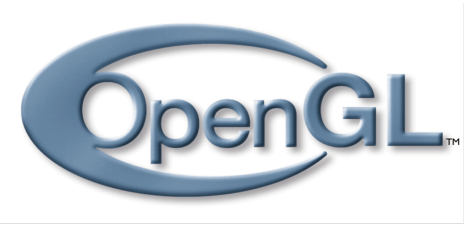
Visibility / Display

VORTEX

Introduction to OpenGL

- OpenGL : Open Graphics Library
  - Standard specification (Architecture Review Board (ARB))
  - API : software interface with graphic hardware
    - 200 commands to write interactive 3D programs
  - State machine
    - State variables : viewing frustum, drawing color, material properties, light properties, etc.

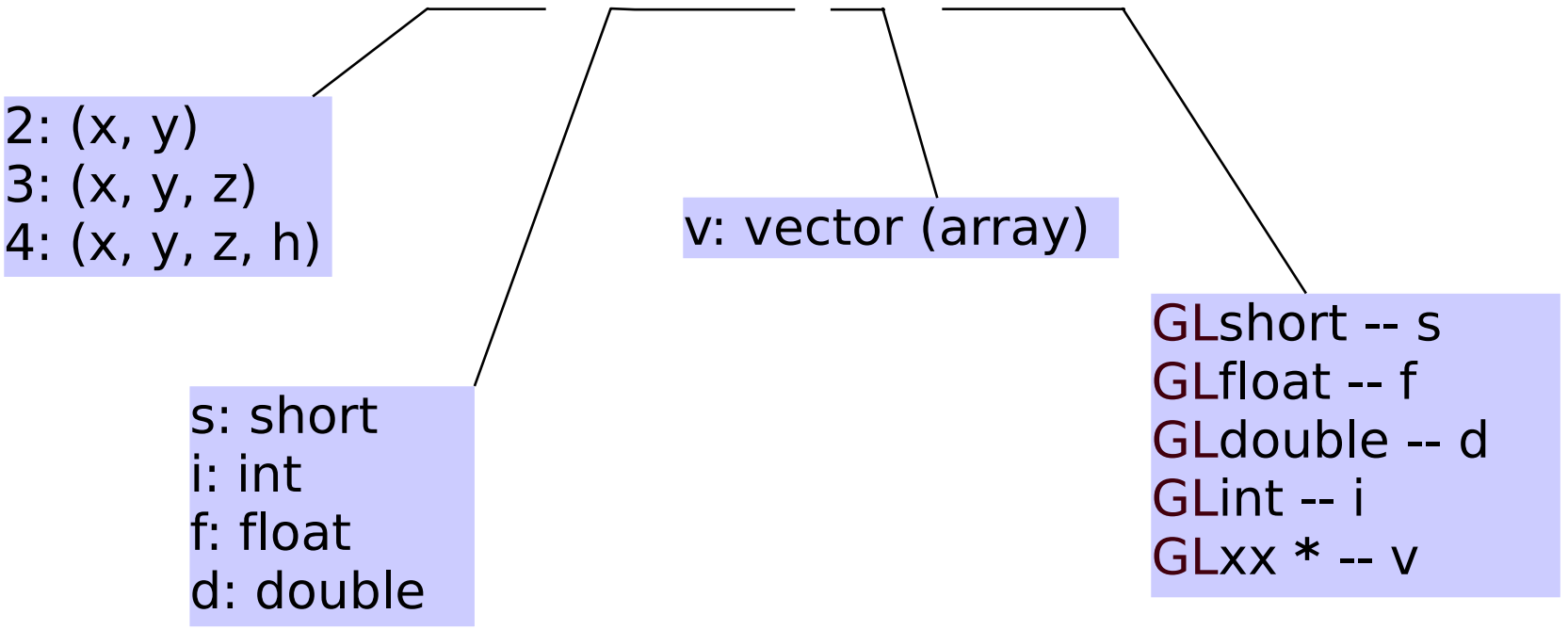
- OpenGL is not
  - A window manager
  - A 3D modeling tool



Basic commands

- General form of an OpenGL function:

```
void gl...{2,3,4}{s,i,f,d}[v] (TYPE coords);
```



Basic commands

- Vertex: basic component of a primitive

  glVertex {2,3}{s,i,f,d}[v]()

  - Defined between a glBegin() and a glEnd()
  - Specification of a mode (type of the primitive)

```
glBegin(mode);
glVertex*(coordinates);
glVertex*(coordinates);
.
.
glVertex*(coordinates);
glEnd();
```

MODES
GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_POLYGON
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP

Basic commands

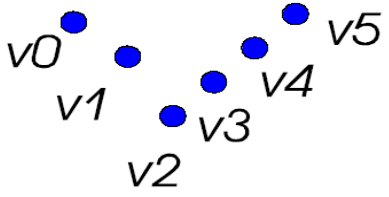
- glVertex* (coordinates)
  Samples:

```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex3f( 2.3f, 12.0f, -4.8f);

GLdouble dvect[3]={5.0, 9.0, 1435.0};
glVertex3dv(dvect);
```

```
typedef struct {
    GLfloat x, y, z;
} Point3D;

Point3D pt = {0.0, 4.0, 6.0};
glVertex3fv((GLfloat *) &pt);
```

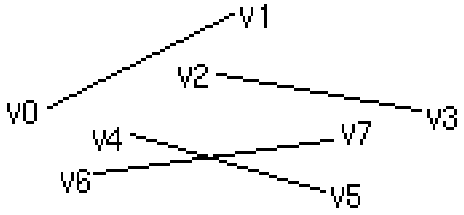
Basic commands

- Primitive types:
  - Points

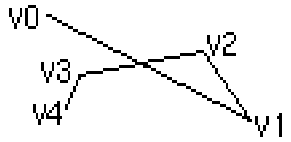


GL_POINTS

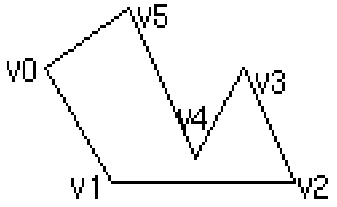
  - Line segments
    - Independent line segments
    - Polylines
    - Empty polygons



GL_LINES



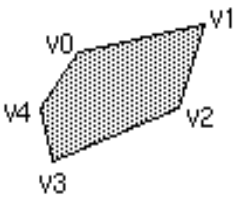
GL_LINE_STRIP



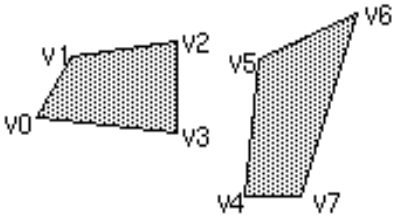
GL_LINE_LOOP

Basic operations

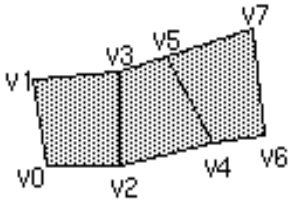
- Primitive types
  - Filled polygons
    - polygons, independent triangles or quads
    - triangle fan, triangle strip, quad strip



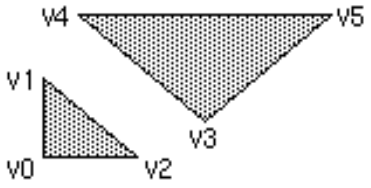
GL_POLYGON



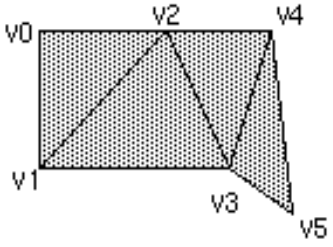
GL_QUADS



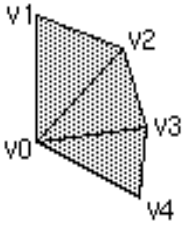
GL_QUAD_STRIP



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Basic operations

- Polygons orientation
  - Default : front face if oriented counterclockwise on the screen
  - Can be changed :

glFrontFace(mode)

    - Mode : GL_CCW (counterclockwise), GL_CW (clockwise)

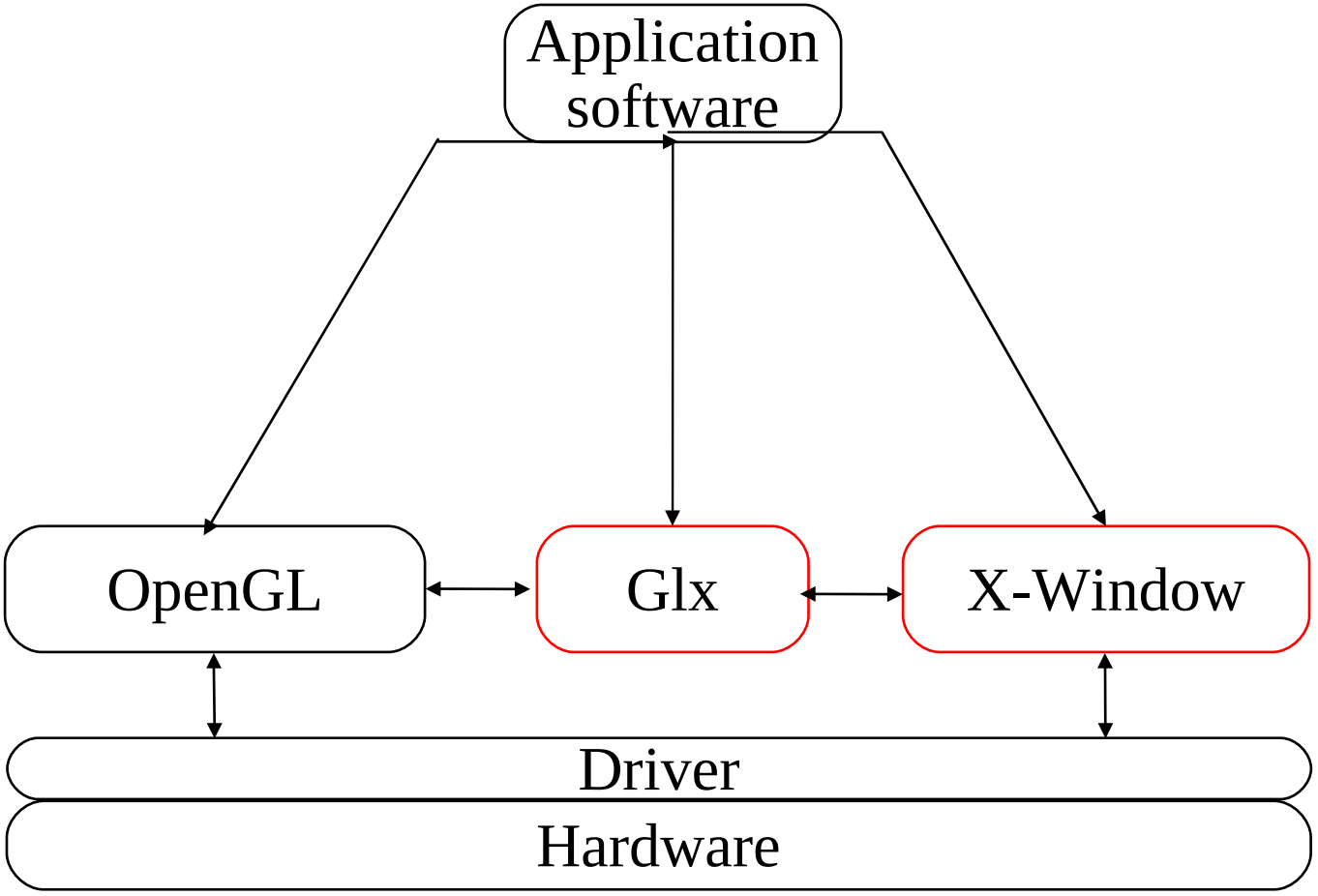
- Drawing mode of the faces

glPolygonMode(face, mode)

  - Face : GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - Mode : GL_POINT, GL_LINE, GL_FILL

Introduction to OpenGL

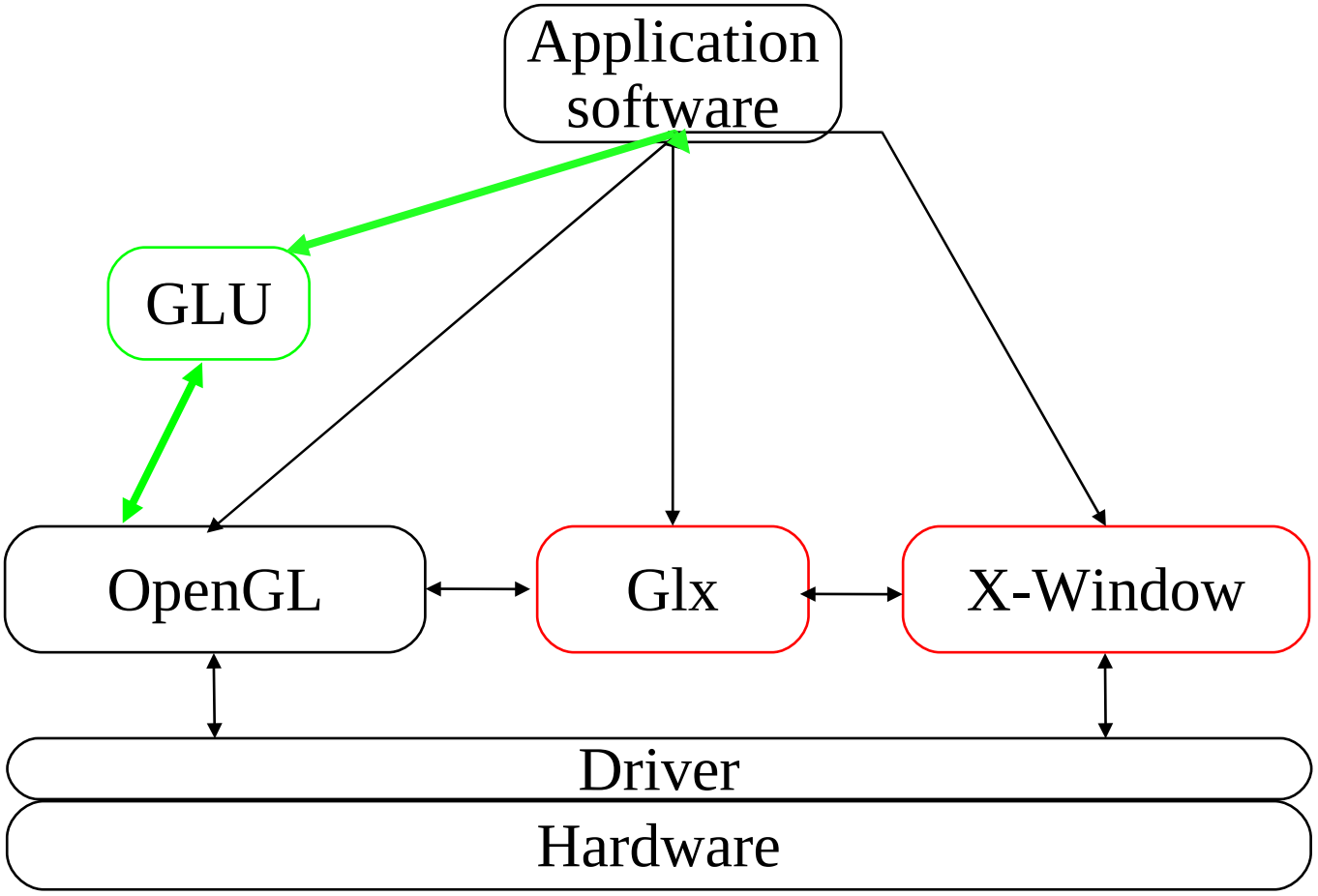
- Programming interfaces



VORTEX

Introduction to OpenGL

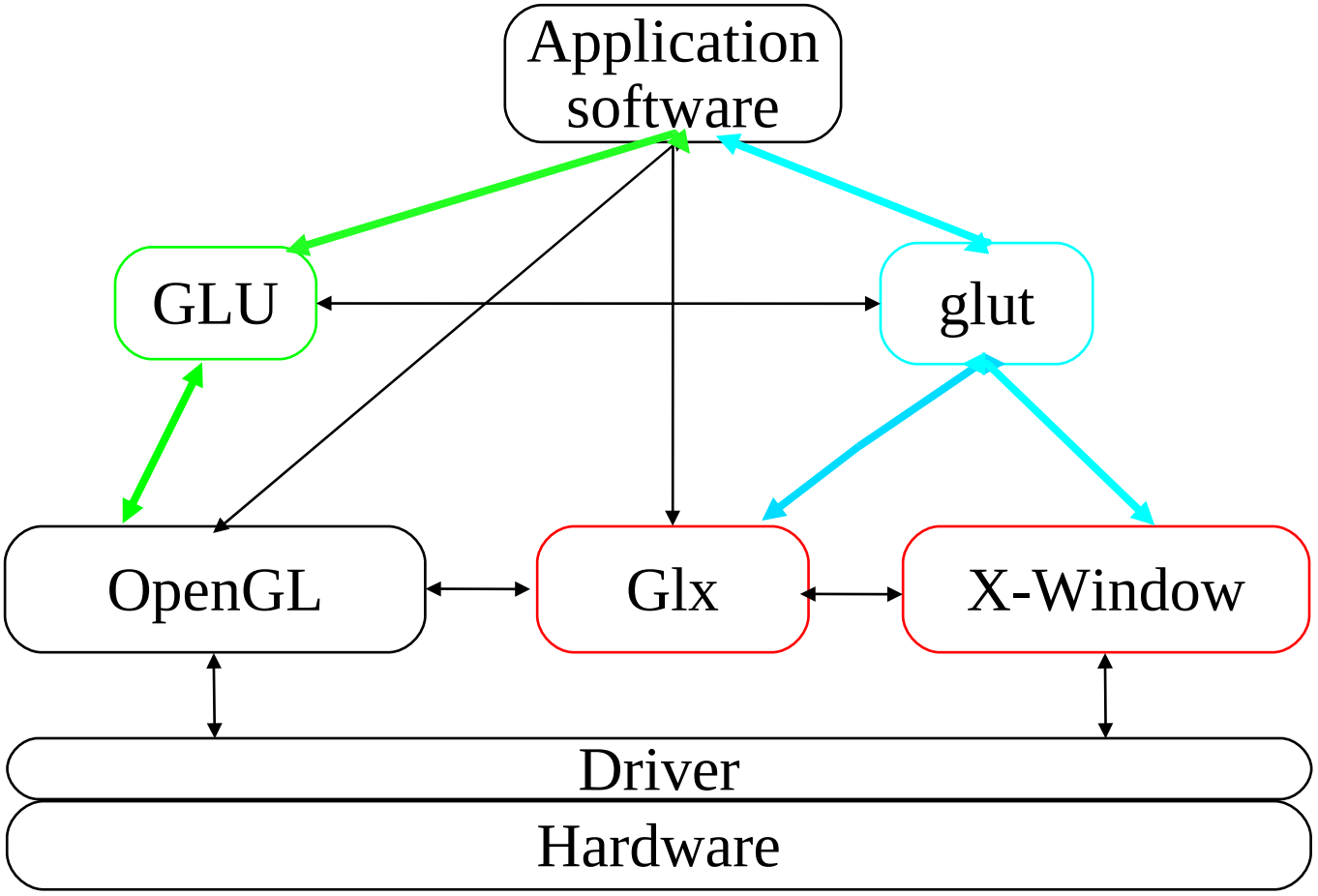
- Programming interfaces



VORTEX

Introduction to OpenGL

- Programming interfaces



VORTEX

Introduction to GLUT

- GL Utility Toolkit (≠ OpenGL)
  - A library to create and handle windows easily
  - Can be used to handle events as well (key strokes, etc..)
  - Available on various platforms
    - Linux, Windows, MacOS ...

Introduction to GLUT

- Initializing and creating windows
  - Just call the functions :
    - void glutInit(int * argc, char **argv);
    - void glutInitDisplayMode(unsigned int mode);
    - void glutInitWindowSize(int width, int height);
    - void glutInitWindowPosition(int x, int y);
- Then :
    - int glutCreateWindow(char *title);

Introduction to GLUT

- Can be used to handle double buffering
  - void glutSwapBuffers(void);
- Can be used to redraw the window at any time
  - void glutPostRedisplay(void);
- (At last) launch the main loop (waits for an event)
  - void glutMainLoop(void);

Introduction to GLUT

- Handles user interactions
  - with the use of Callback procedures:
  - void glutDisplayFunc(void (*f)(void));
  - void glutReshapeFunc(void (*f)(int width, int height));
  - void glutKeyboardFunc(void (*f)(unsigned int key, int x, int y));
  - void glutMouseFunc(void (*f)(int button, int state, int x, int y));
  - void glutMotionFunc(void (*f)(int x, int y));
  - void glutSpecialFunc(void (*f)(int key, int x, int y));
  - void glutIdleFunc(void (*f) (void));

Introduction to GLUT

- GLUT has some predefined 3D objects:
  - void glut{Wire Solid}Cube(GLdouble size);
  - void glut{Wire Solid}Sphere(GLdouble radius, GLint slices, GLint stacks);
  - void glut{Wire Solid}Cone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
  - void glut {Wire Solid}Torus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
  - void glut {Wire Solid}Teapot(GLdouble size);
  - void glut {Wire Solid}Tetrahedron(void);
  - void glut {Wire Solid}Decahedron(void);

Tutorial: Draw a simple polygon

- Write an OpenGL program that draws a filled polygon in white on a black background
  - Function that draws the polygon
    - void display(void)
  - Function that quits the application whenever <q> is stroked
    - void key (unsigned char c, int mouseX, int mouseY)
  - Function handling the possible change in size of the window
    - void reshape(int width, int height)
  - Main
  - Include and constants