

# Graphics Pipeline and OpenGL

---

10 February 2009

CMPT370

Dr. Sean Ho

Trinity Western University

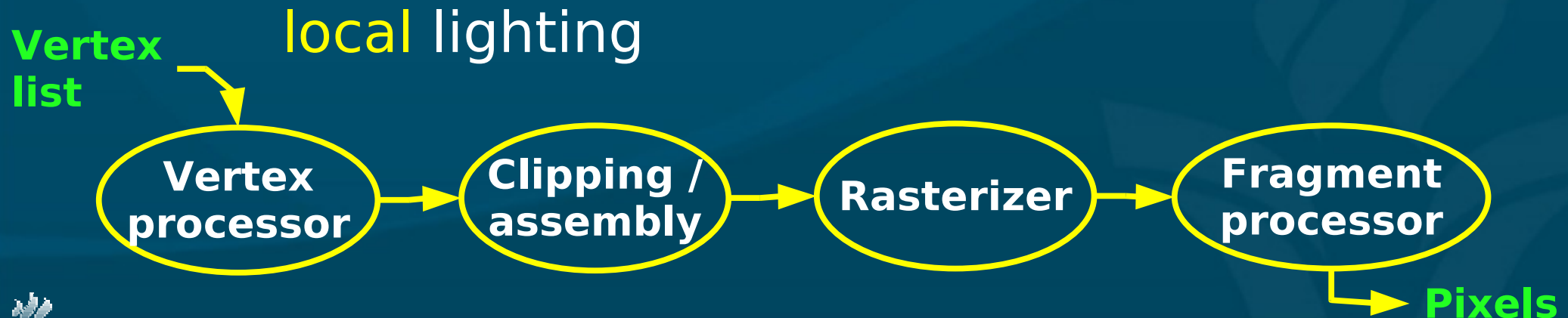
# Review last time

---

- Light and colour models
- Geometric representation: trimesh
- Off-line rendering: raytracing, radiosity
- Real-time interactive graphics pipeline:
  - Vertex processing
  - Clipping and culling
  - Rasterizing
  - Fragment processing
- Graphics API overview (OpenGL)

# Real-time graphics pipeline

- This is what your graphics card **hardware** does
- **Input**: scene **objects**, **lighting**, **camera**
  - Most of the data is the **vertex** list
- **Output**: **pixels** stored in the **framebuffer**
  - **Raster** graphics
- Usually processes **objects** one at a time:  
**local lighting**



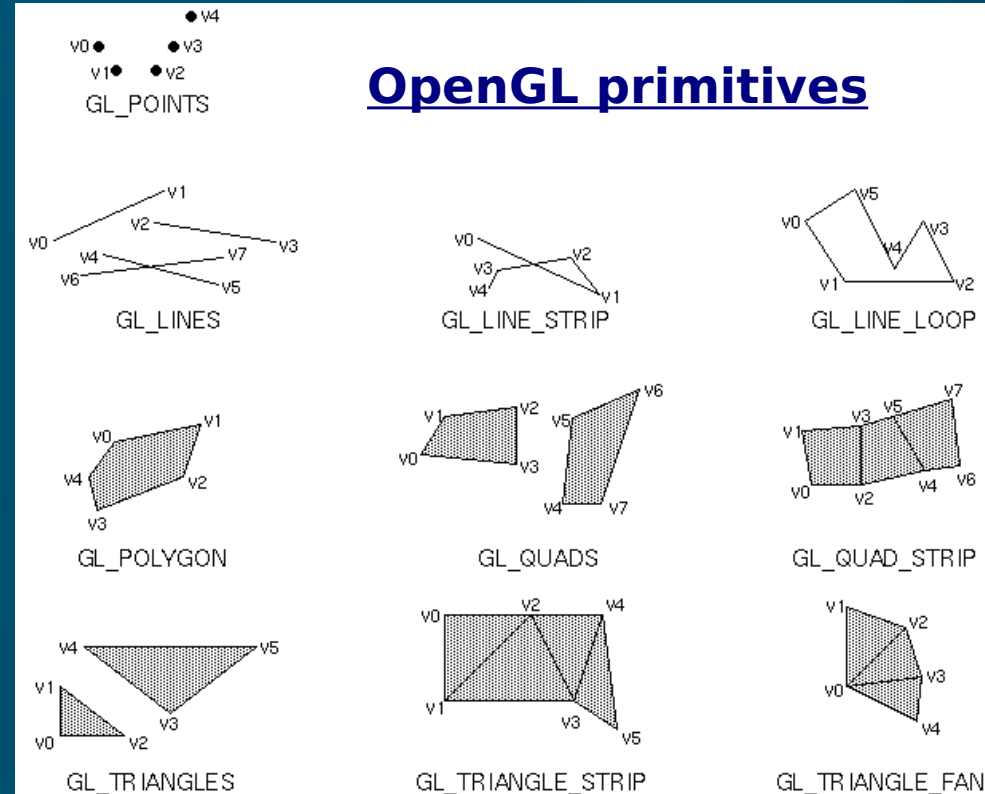
# Kinds of coordinate transforms

- The **transformations** done on vertices include:
  - **Translation**: **shift** in  $(x,y,z)$
  - **Rotation**: e.g., 3 **Euler** angles
  - **Scaling**: **uniform** or along **3 axes**
  - (Perspective, affine)
- 3D points are **projected** onto 2D image plane:
  - **Perspective** projection:
    - ◆ Projection lines **meet** at **center of projection**
  - **Parallel** projection:
    - ◆ Projection lines are all **parallel**

# Primitive assembly

- The vertex processor also **assembles** vertices into primitives:
  - **Lines/curves**
  - **Triangles/polygons/surfaces**
- Uses the **face list** to index into the vertex list

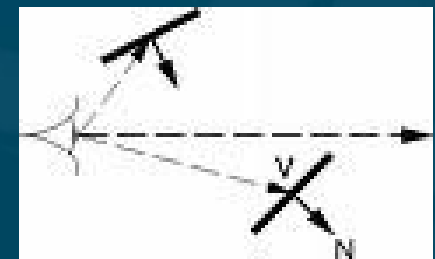
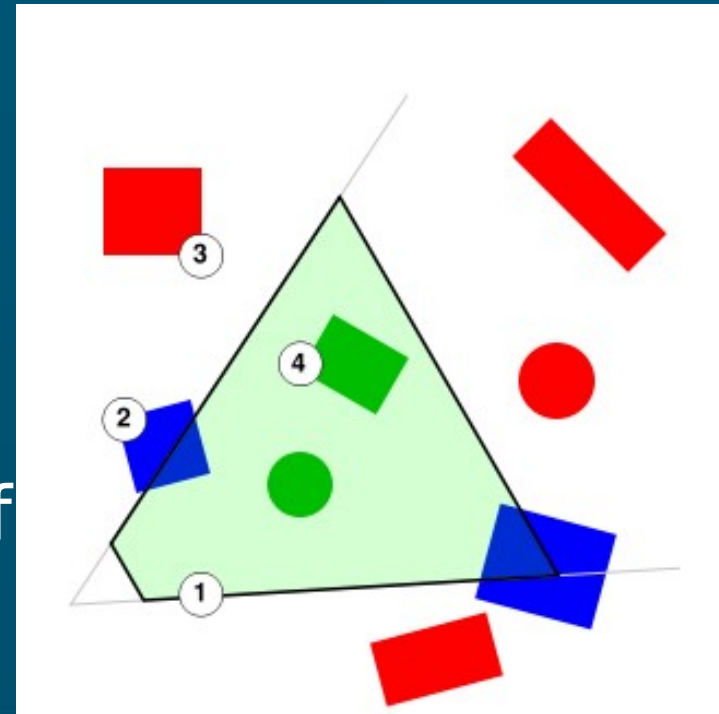
Vertex list



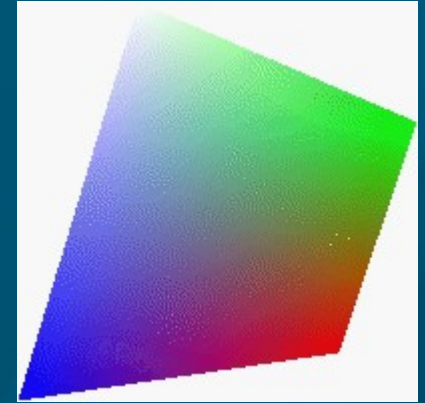
## OpenGL primitives

# Clipping and culling

- Don't render what we can't see
- Clipping
  - Remove primitives outside of the camera's view frustum
- Backface culling
  - Remove triangles facing away from camera
  - Usually cuts down # of triangles by about 50%!
- Other optimizations also possible



# Rasterization



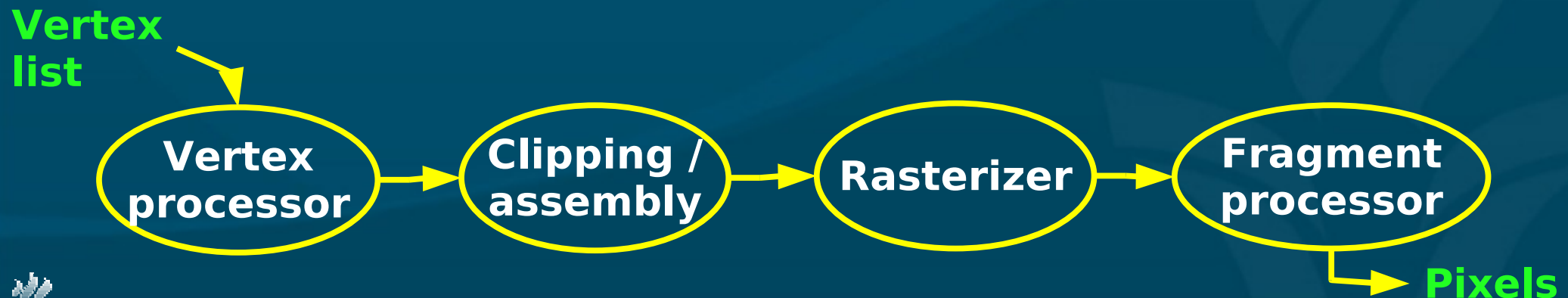
- Convert a **primitive** into a **fragment**:
  - Set of **pixels** just for that primitive
  - Each pixel has both RGB **colour** and **depth**
  - **Interpolate** vertex colours over the fragment
  - Might not correspond to pixels on screen: e.g., **occluded** pixels

Vertex  
list



# Fragment processing

- Assemble the fragments into final framebuffer
- Hidden-surface removal:
  - Some fragments may **occlude** parts of others
    - **Z-buffer** sorts pixels by distance
    - Handle **transparency**



# Programmer's interface

- A graphics **API** allows a program to interact with the graphics **pipeline**
- **Library** subroutines (see [CubeView.cxx](#))
  - Specify the scene (**models**)
  - Specify the **lighting**
  - Specify the **camera**

Application program



API: OpenGL,  
Direct3D



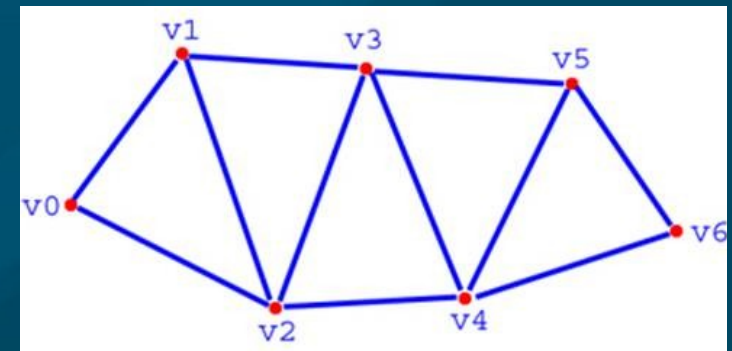
GPU: graphics card



# Graphics API: Model

- Geometry: vertices (0D)
  - Line segments, curves (1D)
  - Polygons (2D), parametric surfaces
- Material properties: colour, specularity, etc.
- Example:

```
glBegin(GL_TRIANGLE);  
    glColor3f(0.0, 1.0, 0.0);  
    glVertex3f(0.0, 0.0, 0.0);  
    glVertex3f(1.0, 0.0, 0.0);  
    glVertex3f(0.0, 1.0, 0.0);  
glEnd();
```



GL\_TRIANGLE\_STRIP

# Graphics API: Lighting

## ■ Type of light:

- Ambient (uniform, everywhere)
- Directional (e.g., sunlight)
- Spotlight (cone with falloff)
- Point vs. area light



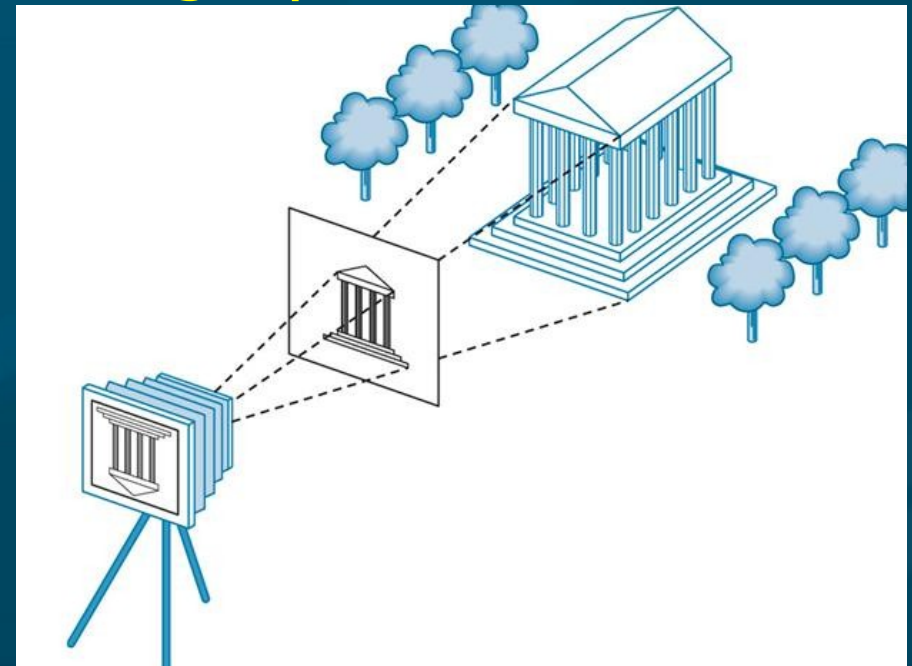
## ■ Material properties:

- Ambient colour
- Diffuse colour
- Specular colour
- Emissive colour



# Graphics API: Camera

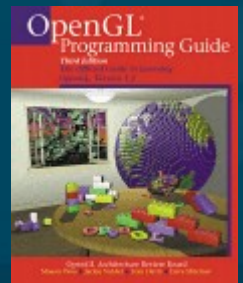
- 6DOF camera model:
  - Position of center of projection (3DOF)
  - Orientation (3DOF)
- Also: location and size of image plane
- Could also consider modelling lens distortion



# History of OpenGL



- Silicon Graphics (**SGI**) was one of the first to implement graphics pipeline in **hardware** (**1982**)
  - In-house library: **GL**
- **OpenGL** (**1992**): platform-independent API
  - **Architectural Review Board** / Working Group
    - ◆ SGI, IBM, HP, MS, Nvidia, 3DLabs
  - Stable, widely-accepted **standard** (now **2.1**)
    - ◆ New changes just for new hardware capabilities
  - Lots of **documentation** and sample code
    - ◆ “Red book” and “Blue book”



# OpenGL libraries

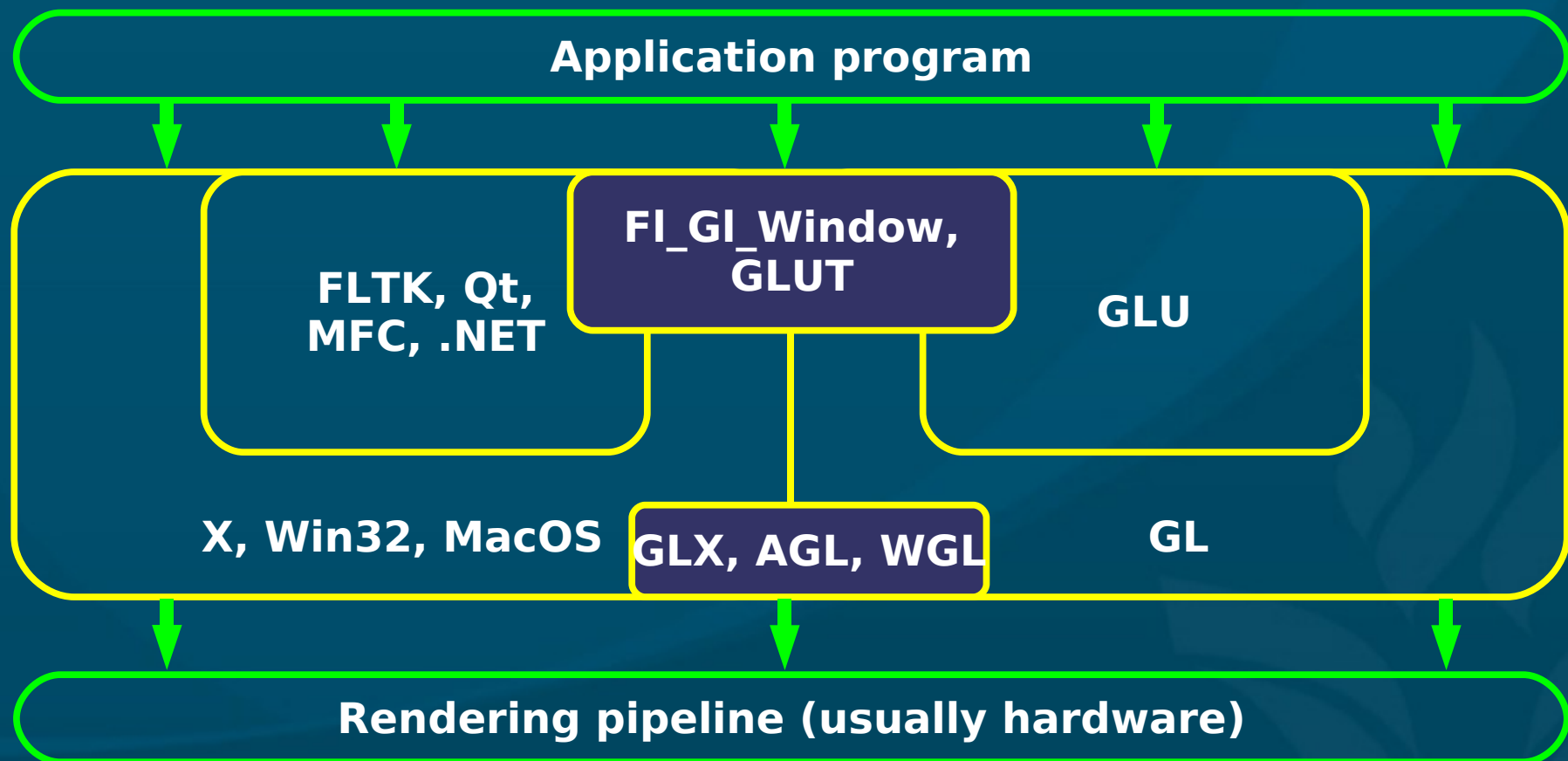
- OpenGL **core** library
  - On Windows: **opengl32.dll**
  - On Linux/Unix: **libGL.a, libGL.so**
- OpenGL **utility** library: **GLU**
  - **Higher**-level drawing routines that use OpenGL core primitives
- OpenGL utility **toolkit**: **GLUT**
  - Basic **GUI** toolkit, very small footprint
  - **FLTK** is much more powerful
  - Also freeglut, OpenGLUT, GLUI

# Interface with window system

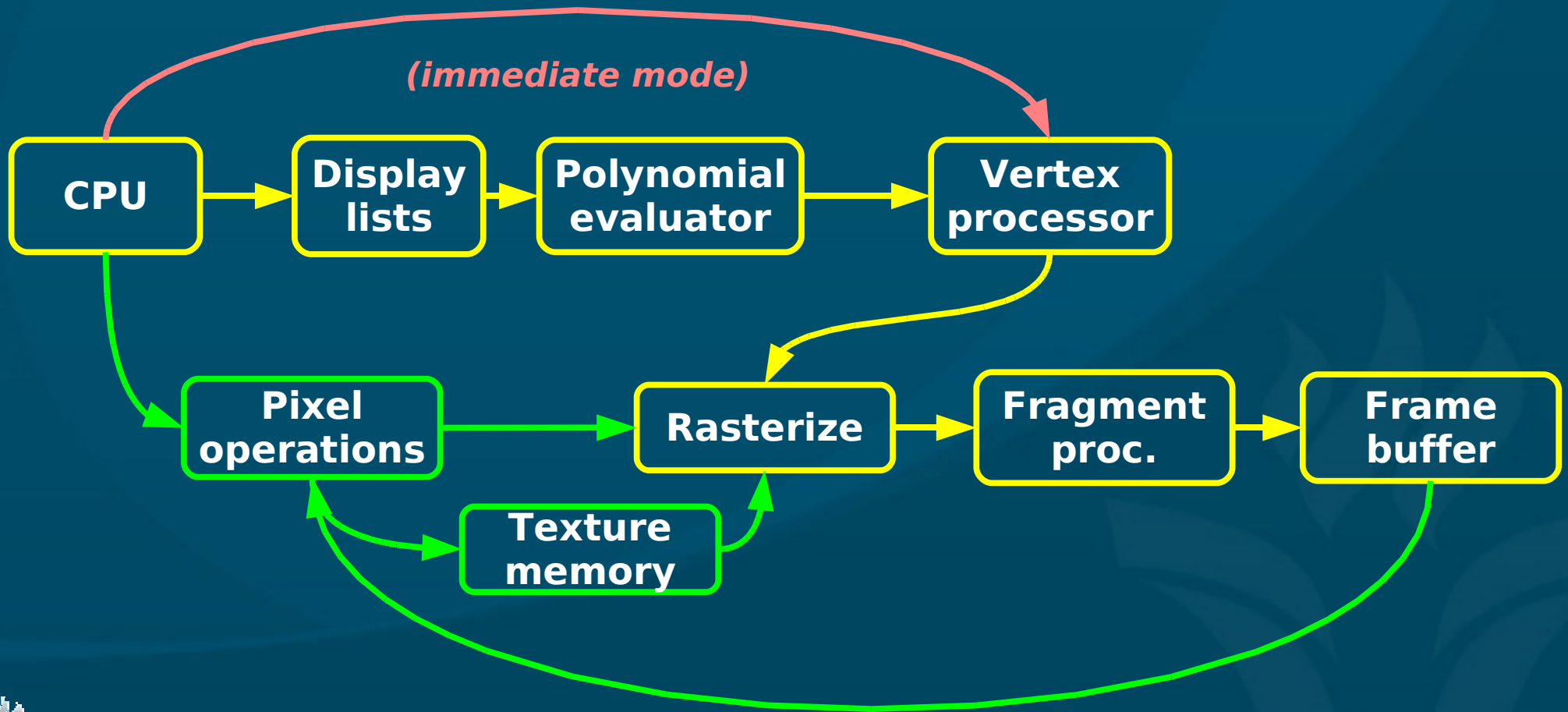
- OpenGL links with the native **window system**
  - **Windows**: **WGL**, **GDI**
  - **MacOS**: **AGL** (for Carbon), **NSOpenGL** (Cocoa)
  - **Unix**: **GLX**
- This makes it possible to run an OpenGL program over the **network**:
  - Run **Xwin32** on local PC (has **GLX**)
  - **ssh -x carmel**
  - run program on carmel, **primitives** get sent to local PC, uses local **hardware** acceleration



# Software architecture



# OpenGL pipeline architecture



# OpenGL state machine

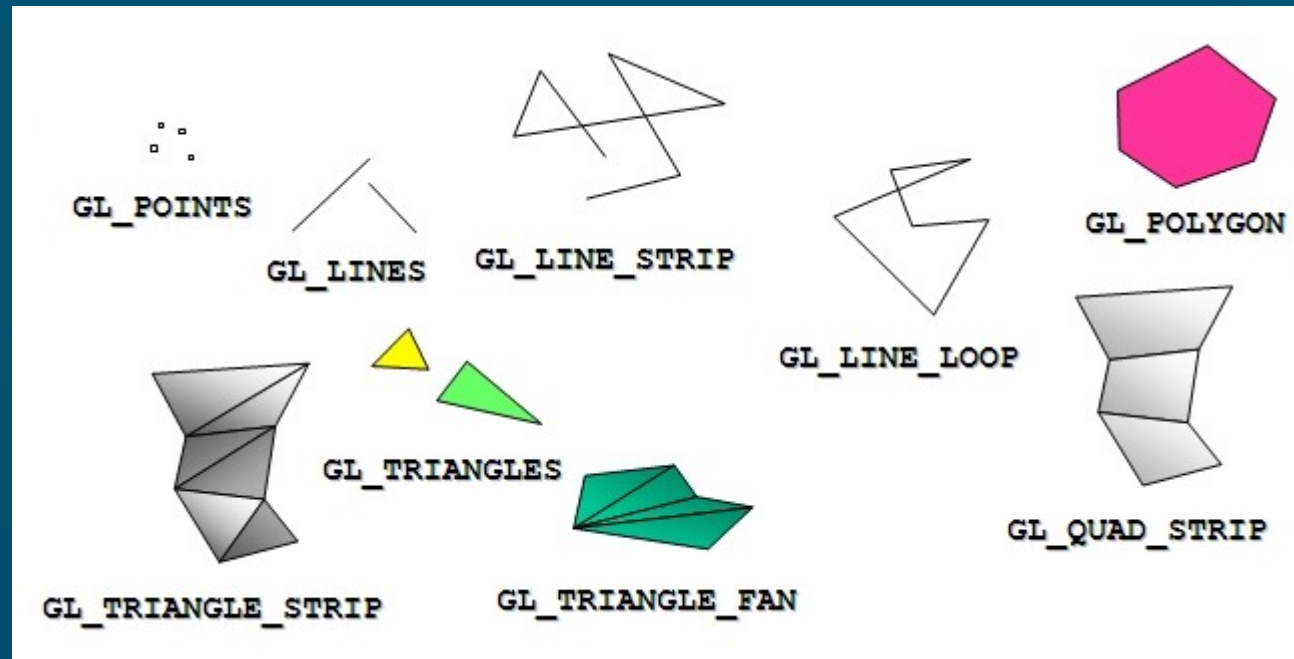
- Two kinds of OpenGL functions
  - Generate primitives
    - ◆ Vertex, line, triangle, polygon, etc.
  - Change state
    - ◆ Current colour
    - ◆ Material properties (shininess, etc.)
    - ◆ Transformations: view, model
- Each primitive gets whatever state was current when it was drawn

# OpenGL functions

- Most core OpenGL **functions** look like this:
  - **glVertex3f**( x, y, z )
    - ◆ gl: belongs to core **OpenGL** library (glu for **GLU**)
    - ◆ **Vertex**: name of function
    - ◆ **3f**: argument **type**: 3 floats
- Not **overloaded**, for efficiency
  - **glVertex3fv**( vec )
    - ◆ takes a **pointer** to an array of 3 floats
  - **glVertex3i**( x, y, z ): **ints**
  - **glVertex3d**( x, y, z ): **doubles**

# OpenGL primitives

- `glBegin(GL_*)` starts a set of **primitives**



- **Polygons** must be **simple**: edges cannot **cross**
- Must be **convex**
- Must be **flat**: all vertices in the same **plane**

# Drawing in OpenGL (see CubeView)

- Start the set of primitives:
  - ◆ `glBegin( GL_TRIANGLES );`
- Set the colour and other attributes:
  - ◆ `glColor3f( 0.0, 0.0, 1.0 );`
- Create the vertices:
  - ◆ `glVertex3f( 0.0, 0.1, 0.2 );`
  - ◆ `glVertex3f( 0.1, 0.0, 0.2 );`
  - ◆ `glVertex3f( 0.0, 0.0, 0.2 );`
- End the set of primitives:
  - ◆ `glEnd();`

# Projection matrix

- The coordinates of `glVertex` are in **world** coords
  - OpenGL converts to **camera** coords, then to **screen** coords
- The **projection** matrix specifies the camera:
  - ◆ `glMatrixMode( GL_PROJECTION );`
- Specify the **viewing volume**:
  - ◆ `glLoadIdentity();`
  - ◆ `glOrtho(left, right, bottom, top, near, far)`
  - **Orthographic** (parallel) projection