OpenGL & the Graphics Pipeline

Lecture notes provided by

Chris Handley

Jim Clark SGI

- PhD in Utah
- Invented the ‘geometry engine’ while at Stanford.
- Founded SGI in 1982
- Iris 1000 - 1984
- Indigo 1990
- Tezro 1995

NVIDIA

- Founded 1993 by Jen-Hsun Huang
- Started with quadratic surface rendering for game consoles all but killed by introduction of Direct X
- 1996 hired David Kirk
- 1997 Riva 128
- 1999 GeForce 256
- 2007 CUDA

ATI

- Founded 1985 by Kwok Yuen Ho (Hong Kong)
- Made graphics boards for IBM and Commodore PC
- 1996 3D accelerator for notebooks
- 2000 Radeon
- 2006 bought by AMD

What is OpenGL™?

- A software interface to graphics hardware.
- Consists of about 120 functions that specify the objects and operations needed to produce high-quality colour images of three-dimensional objects.
- OpenGL Utility Library (GLU) and the OpenGL Extension to the X Window System™ (GLX) provide useful supporting features.

Availability

- C/C++, Objective-C, C#, D, Haskell, Java, Lua, Perl, Python, R, Ruby, Tcl, Ada, FORTRAN, ...
- Unix/Linux, MacOS, any recent Windows flavour
  - –OS/2 and BeOS, ...

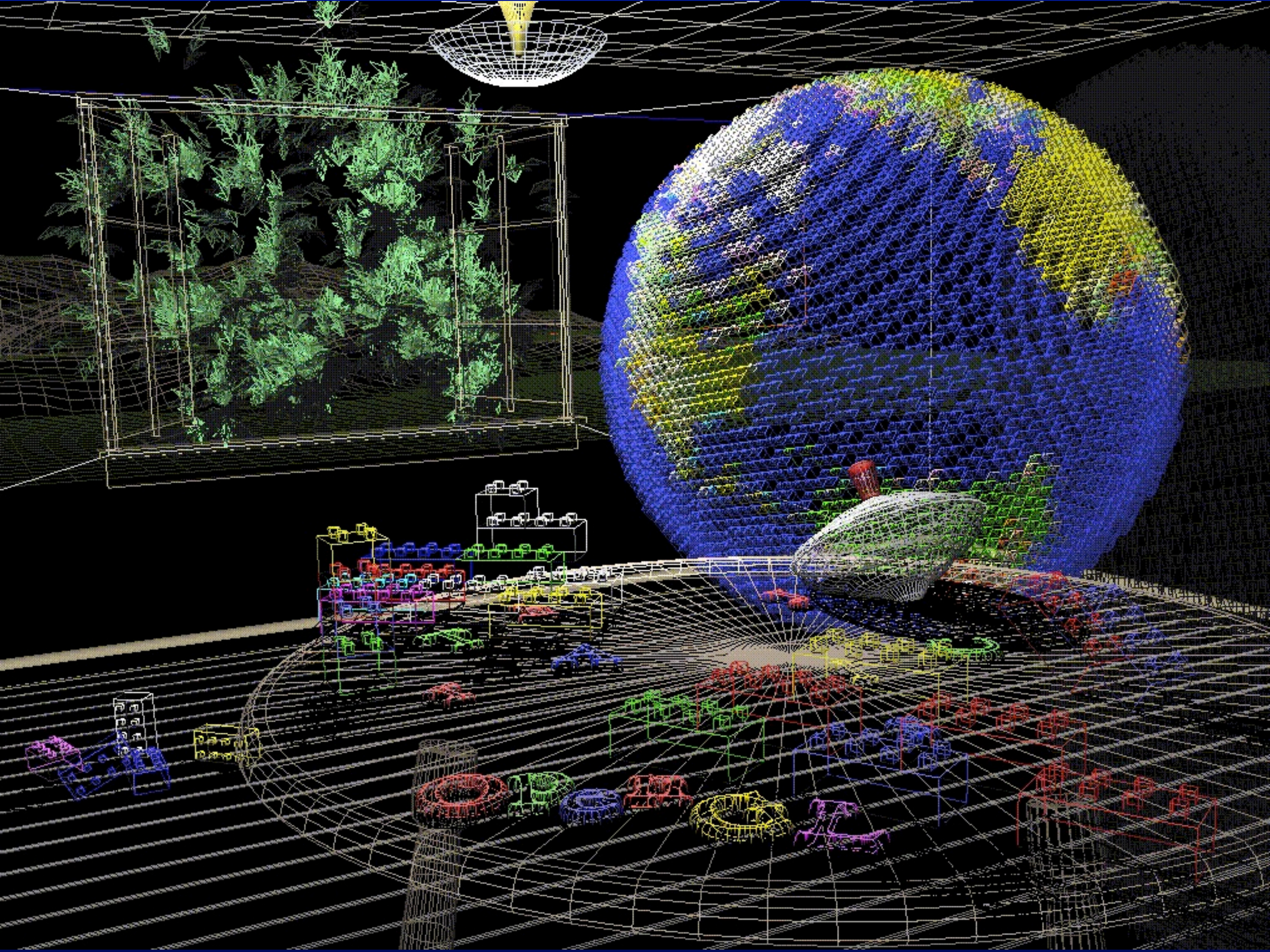
How does it work?

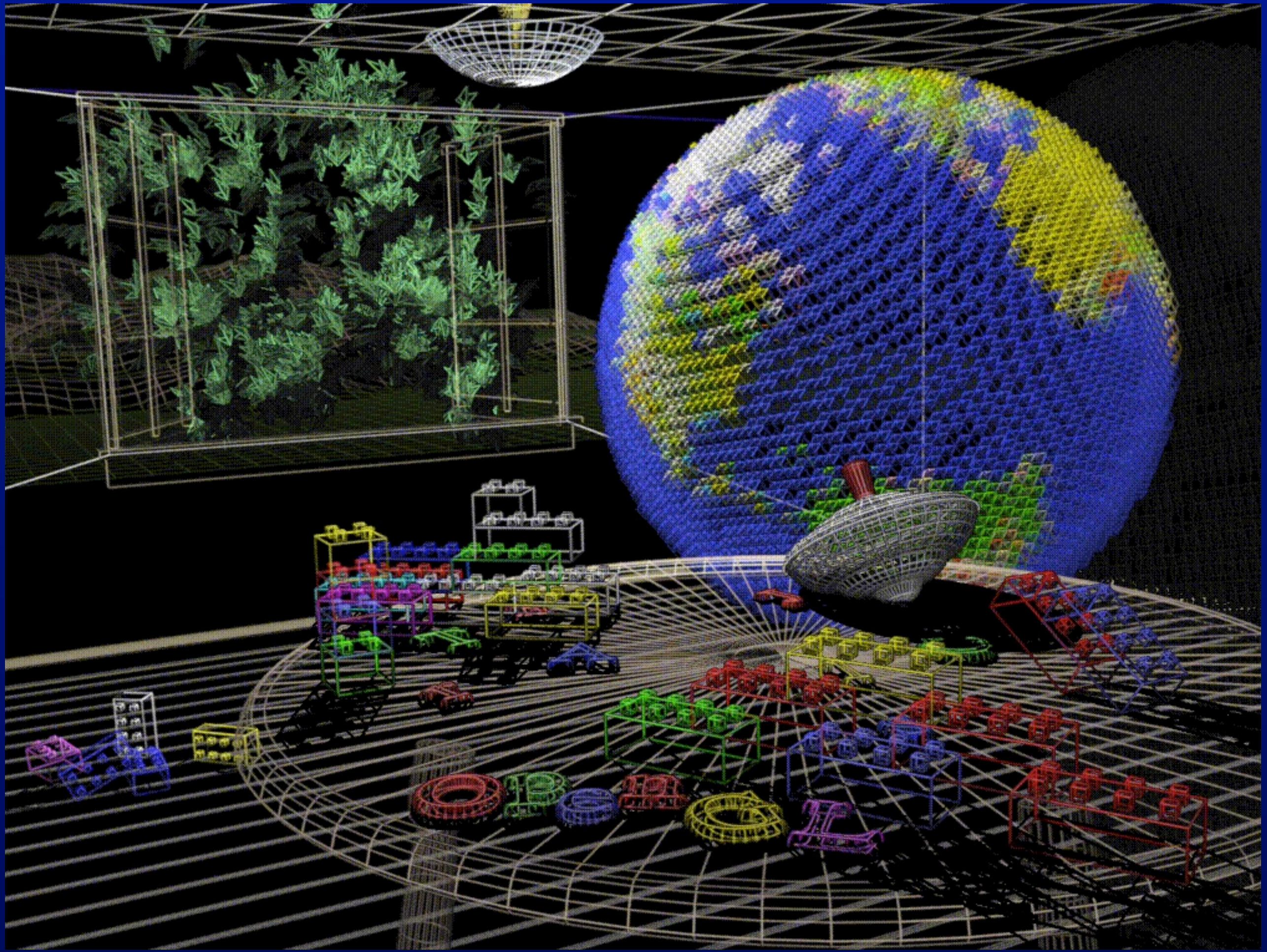
- Client-server model —
  - –Server does the drawing,
  - –Client does the display,
  - –May, in fact, be the same machine.

What doesn't it have?

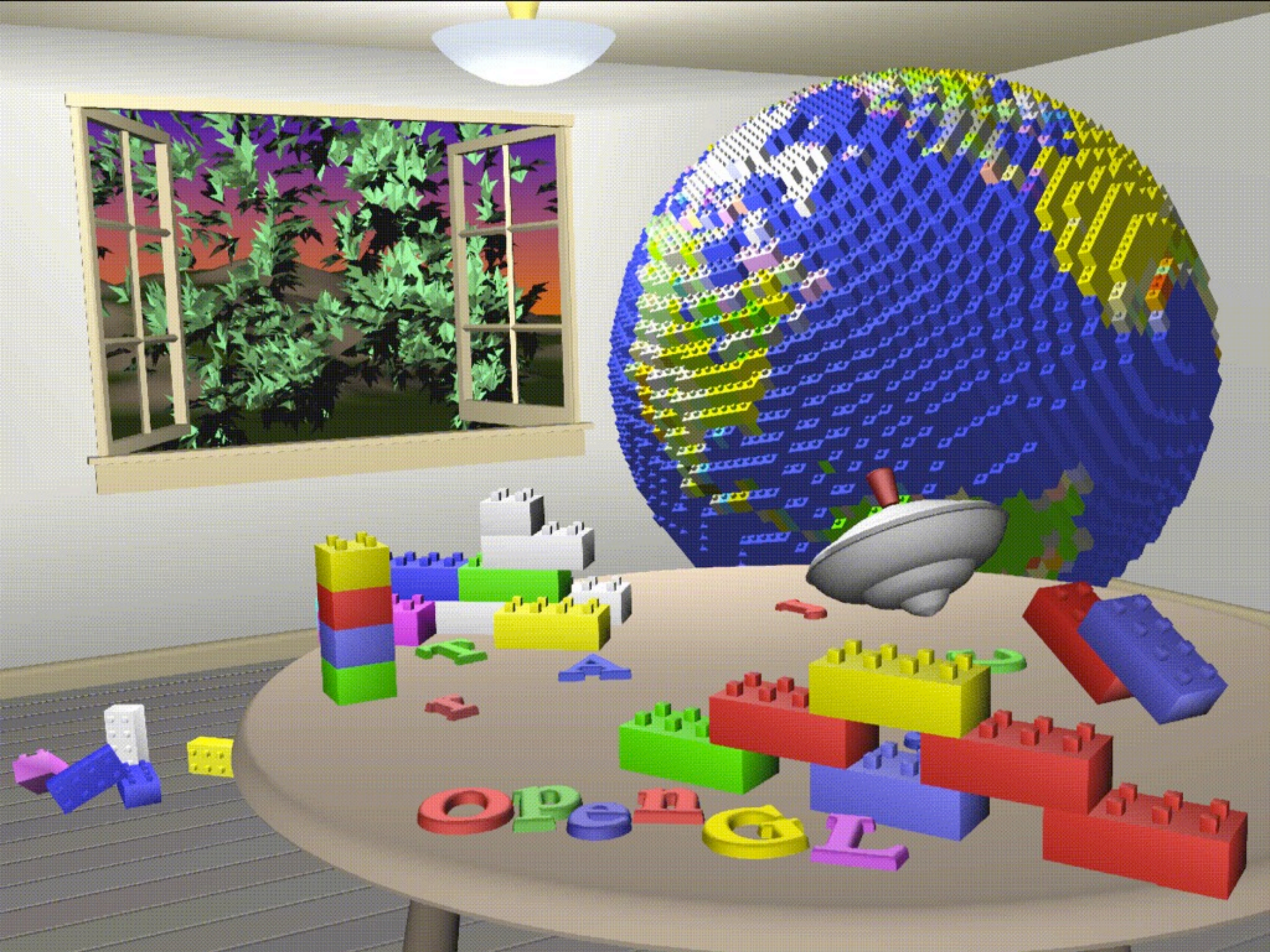
- Commands for performing windowing tasks or obtaining user input.
- High-level commands for describing models of three-dimensional objects.

Well, what *does* it do?

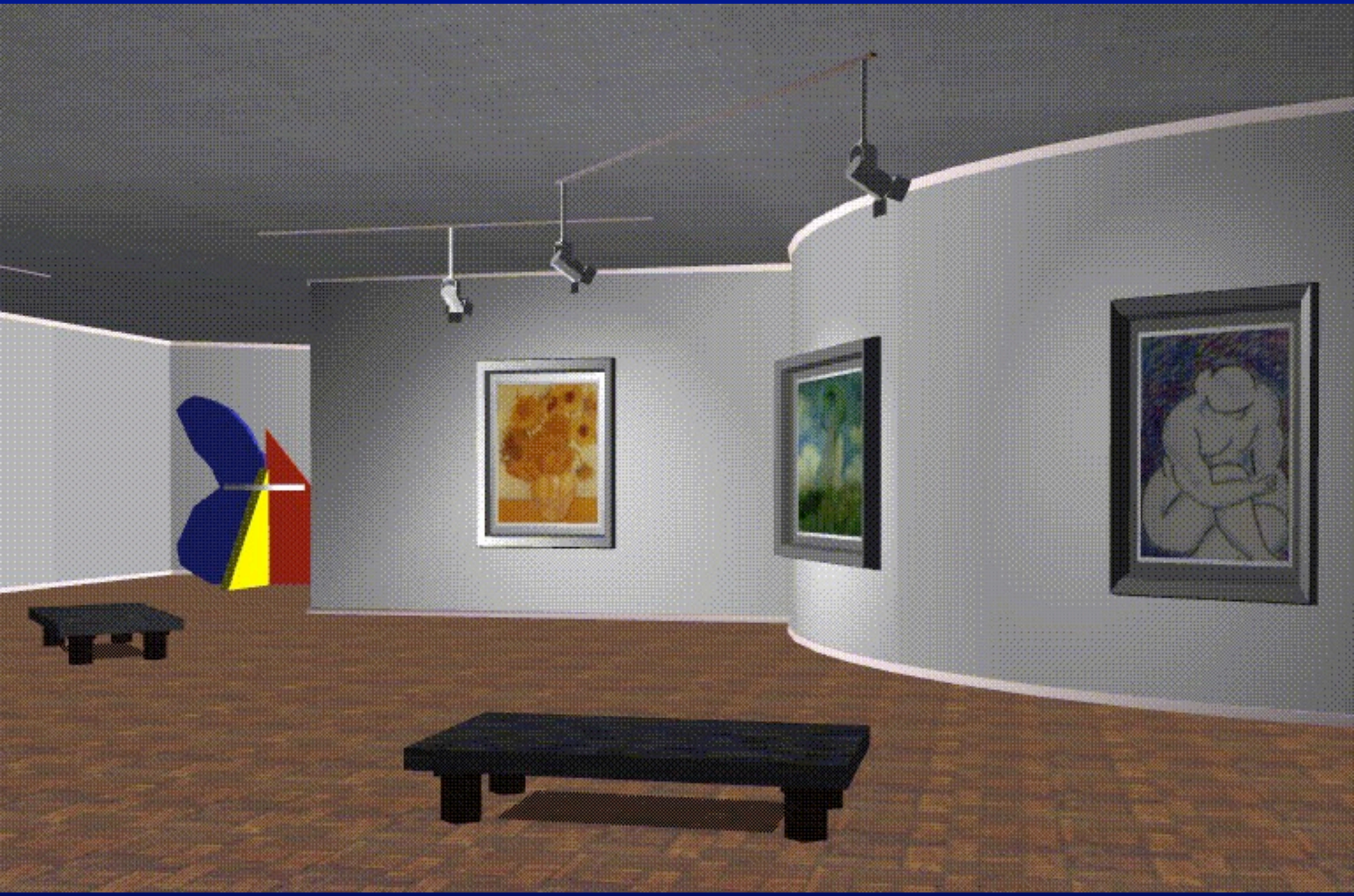


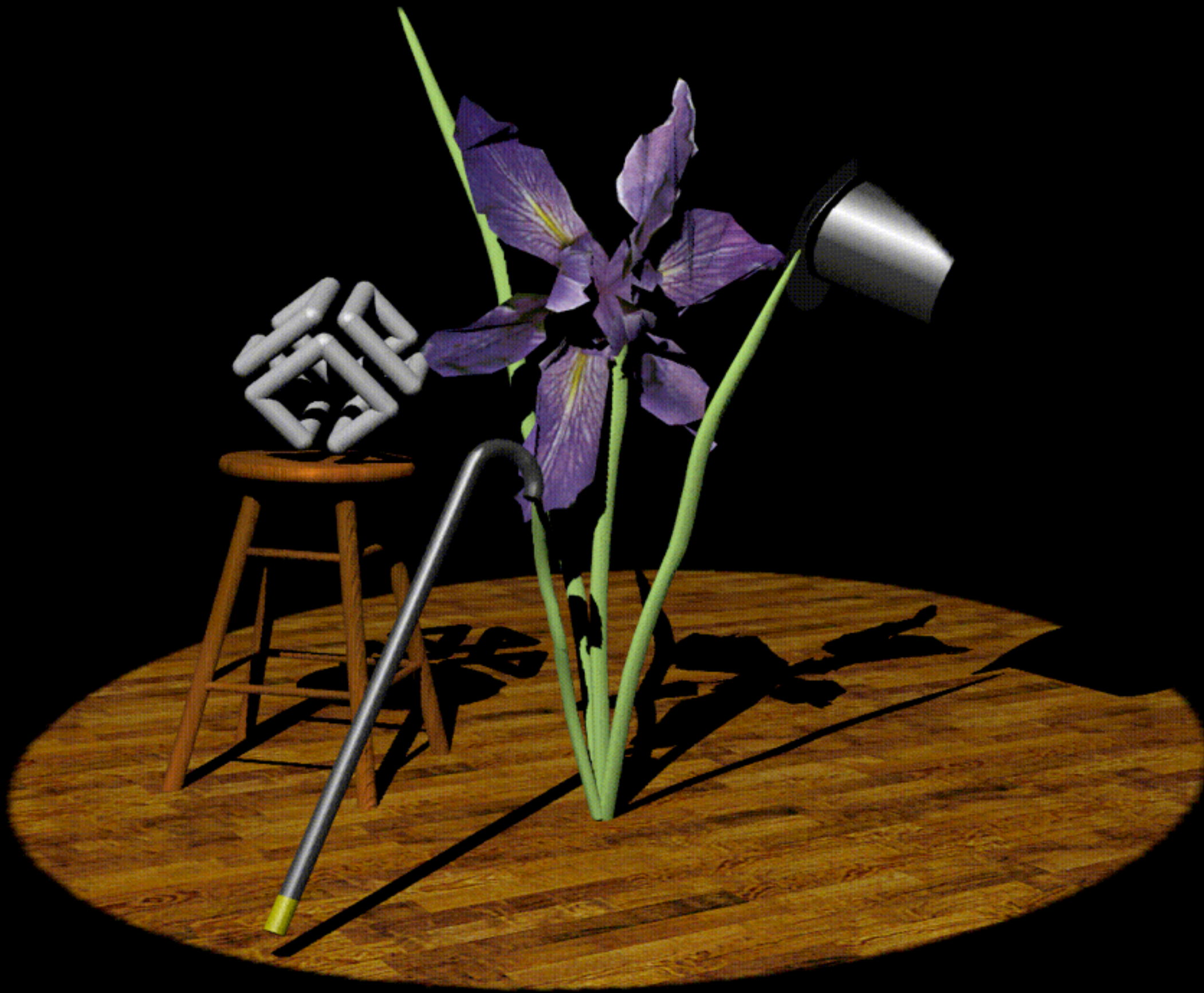


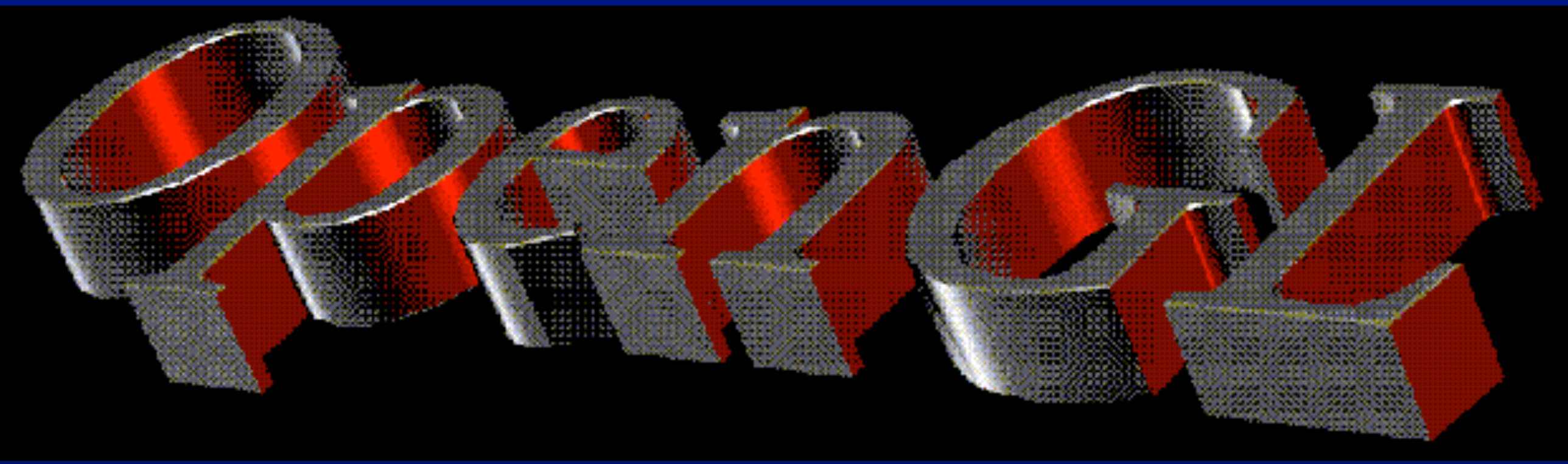

OPOL


OpenGL


OpenGL



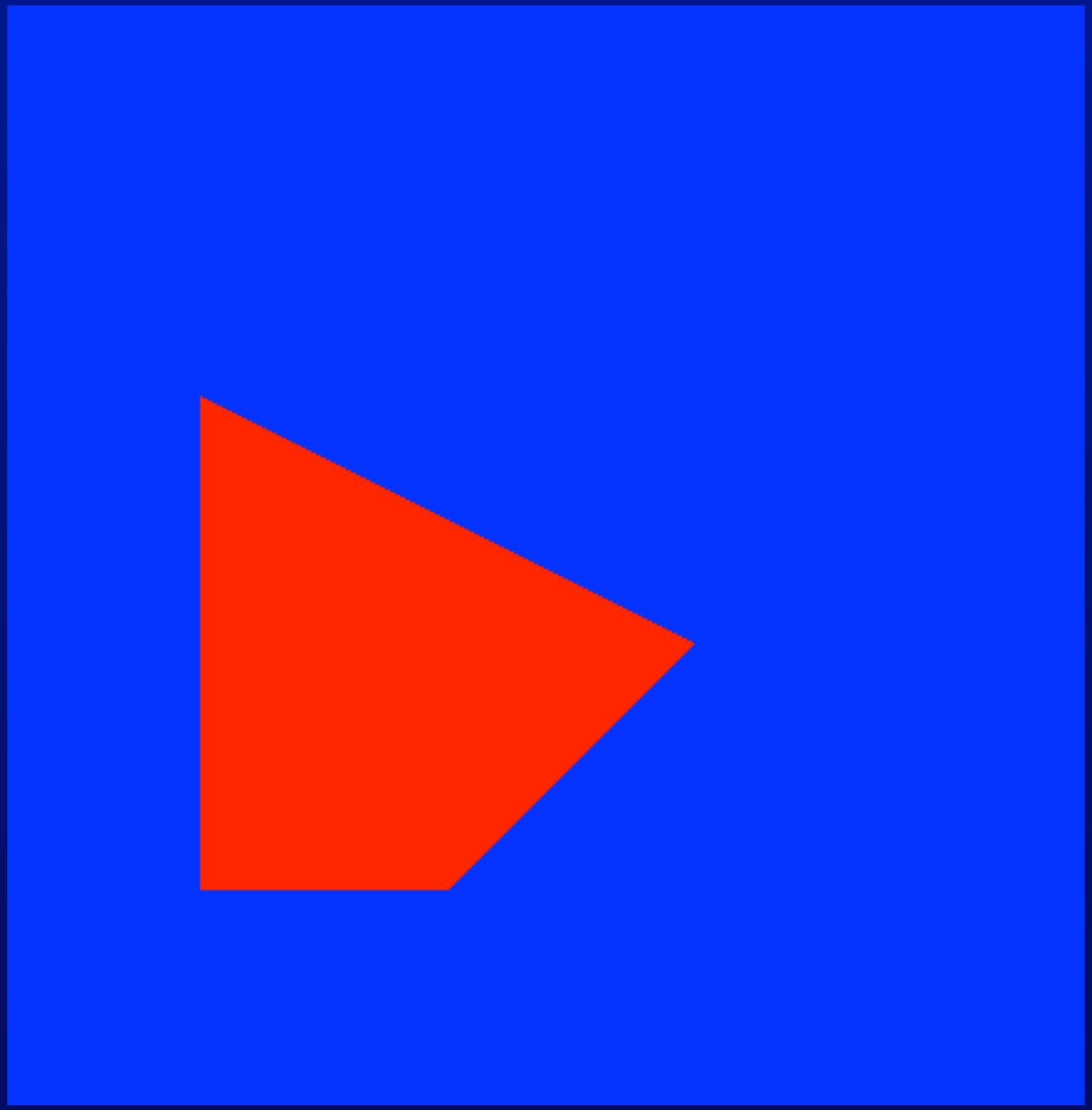




Structure of an OpenGL program

- Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects.
- Arrange the objects in three-dimensional space and select the camera position (view point).
- Calculate the colour of all the objects.
- Convert the mathematical description of objects to pixels on the screen (rasterisation).

```
#include <whateverYouNeed.h>
main() {
    OpenAWindowPlease();
    glClearColor(0.0, 0.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
        glVertex2i(100, 100);
        glVertex2i(100, 300);
        glVertex2i(300, 200);
        glVertex2i(200, 100);
    glEnd();
    glFlush();
    KeepTheWindowOnTheScreenForAWhile();
}
```



OpenGL Command Syntax

- Commands use the prefix `gl` and initial capital letters for each word making up the command name (`glClearColor()`).
- Constants begin with `GL_`, use all capital letters, and use underscores to separate words (`GL_COLOR_BUFFER_BIT`).

Parameter Definition (1)

- Most functions come in several different flavours depending on type (and possibly number) of parameters.
- Consider `glColor*()`. We can use three values (RGB) to represent a colour, or we can include transparency (α-channel).
- Similarly we can specify the values as bytes, integers, floats, doubles, etc.

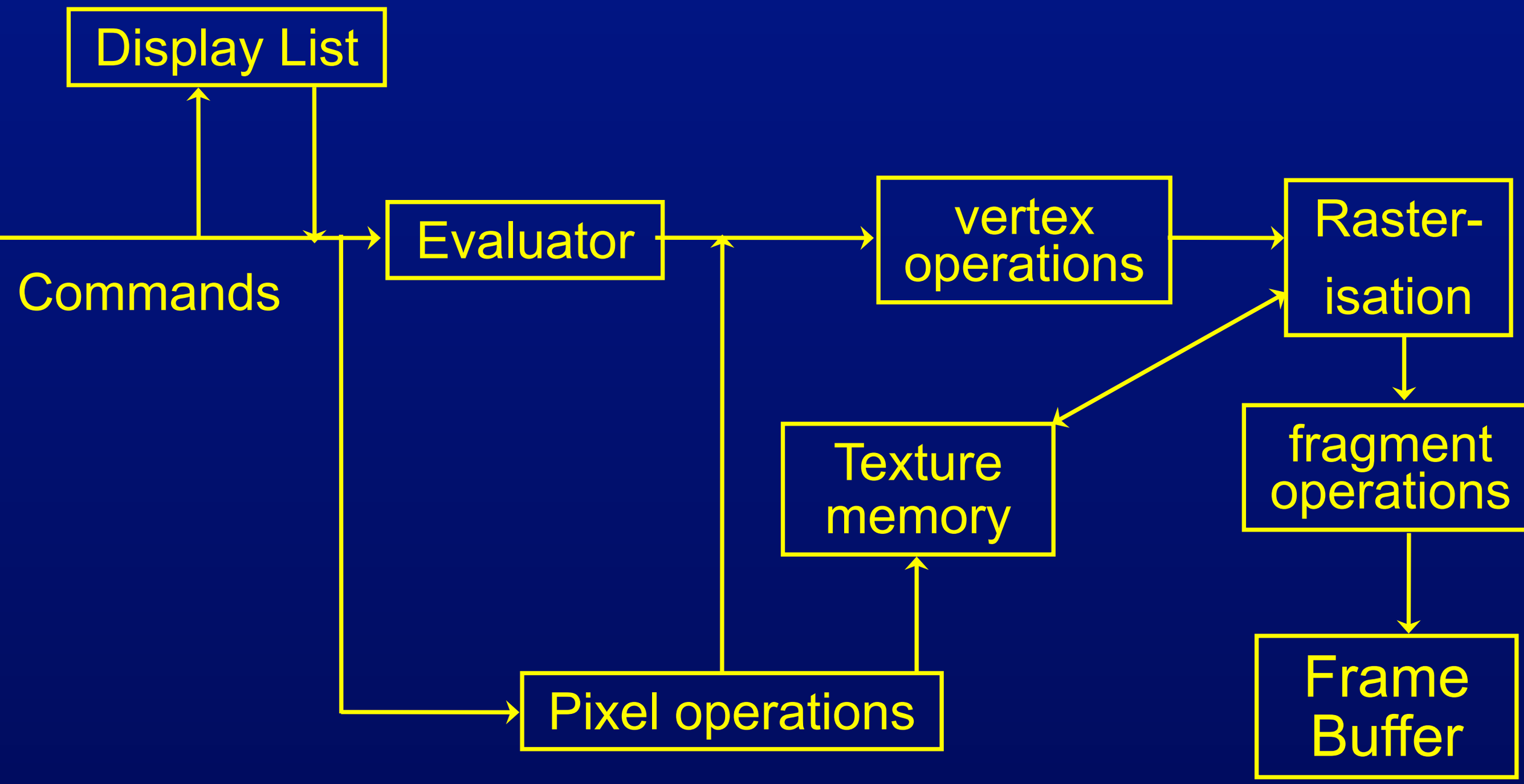
Parameter Definition (2)

- Thus we can set the colour without or with transparency using `glColor3()` or `glColor4()`.
- Similarly we can set the values of the components using any one of `glColor3i()`, `glColor3f()`, `glColor3d()`, etc.

OpenGL as a state machine

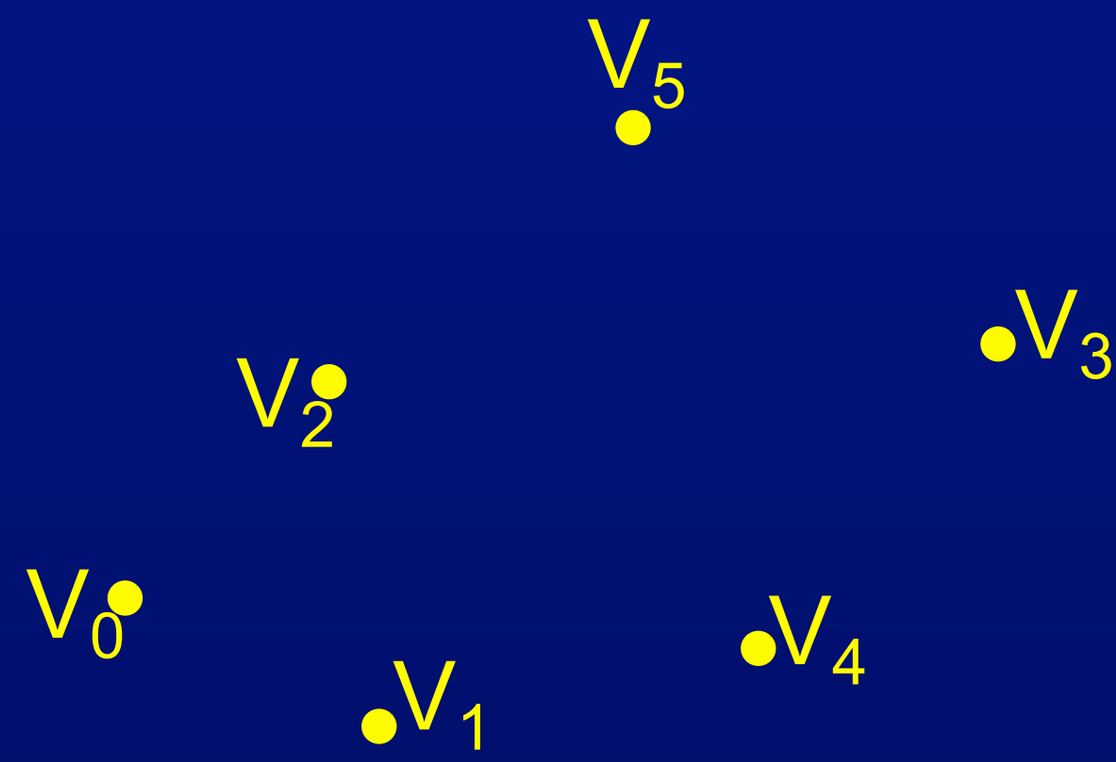
- OpenGL maintains a variety of states (or modes) that start off with default values and that remain set until changed.
- Modes include background and fore-ground colours, viewing and projection transformations, polygon drawing modes, and so on.

Basic OpenGL Operation



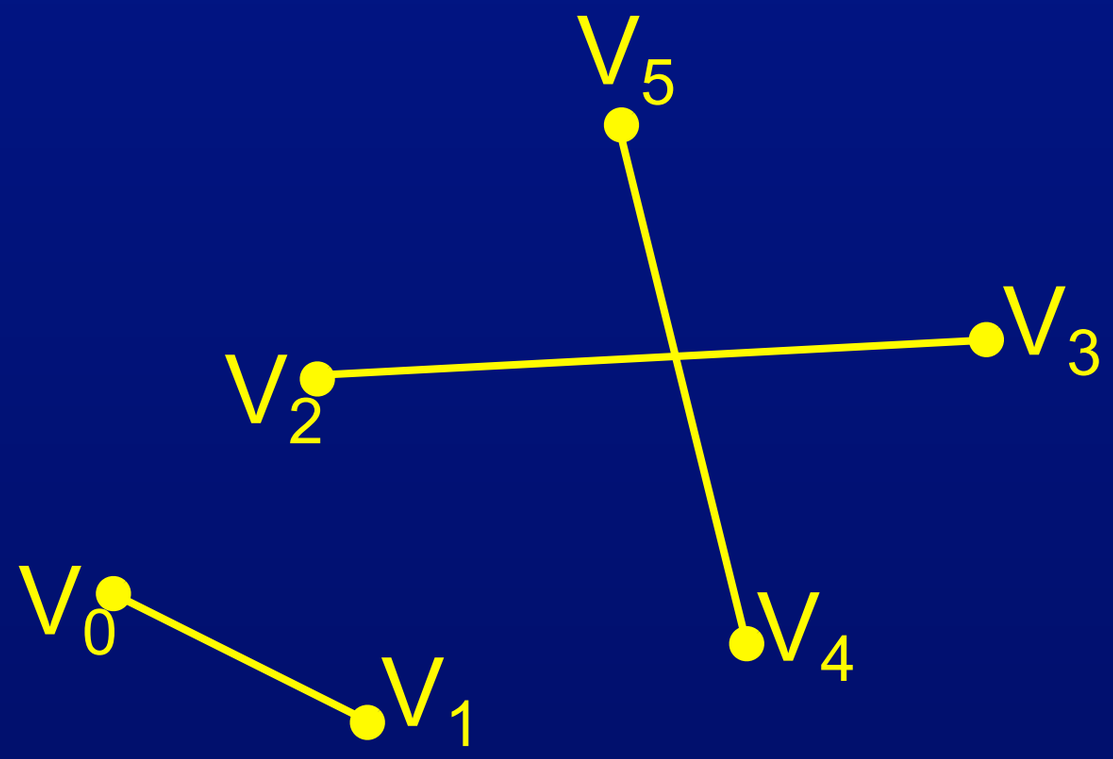
Primitives (1)

- Points



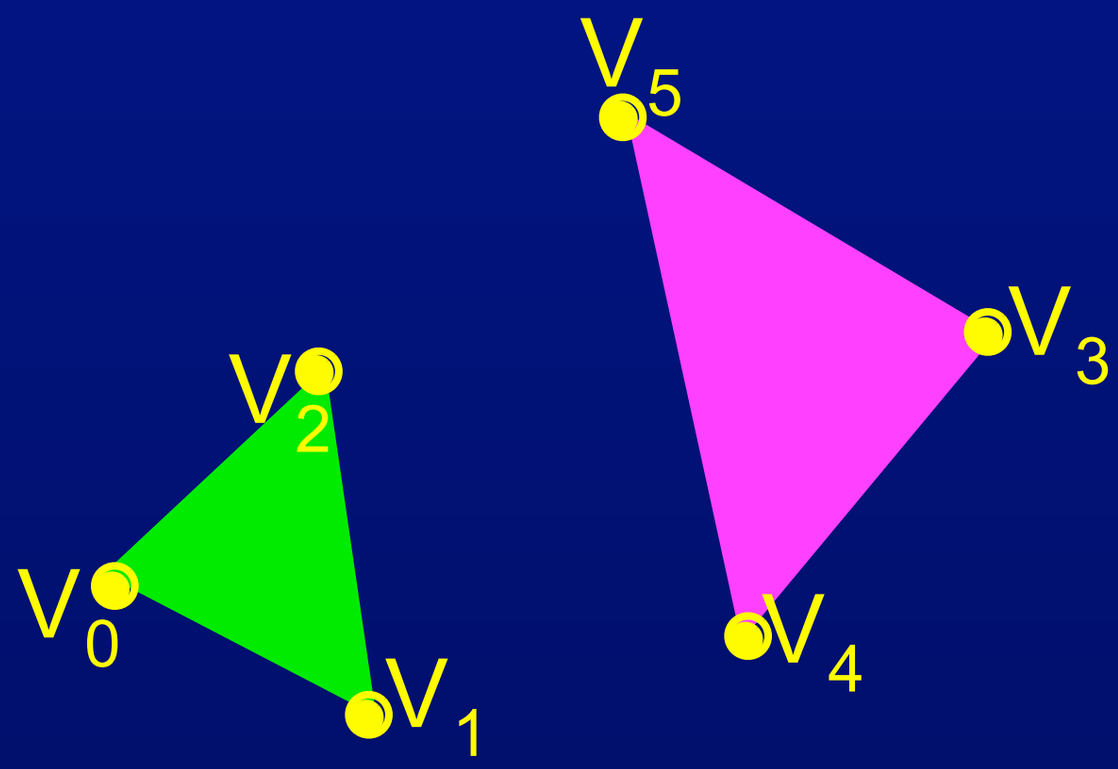
Primitives (1)

- Points
- Lines



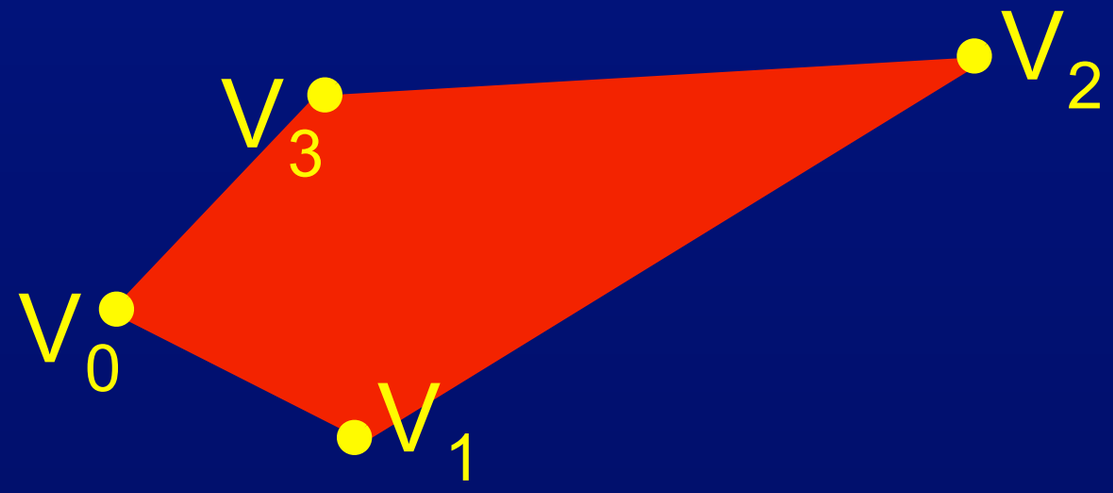
Primitives (1)

- Points
- Lines
- Triangles



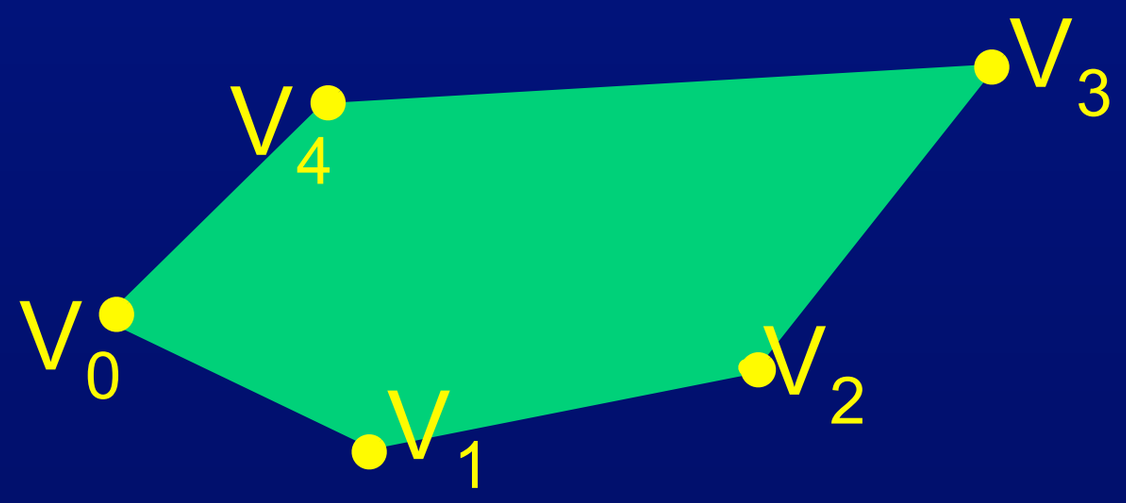
Primitives (1)

- Points
- Lines
- Triangles
- Quads



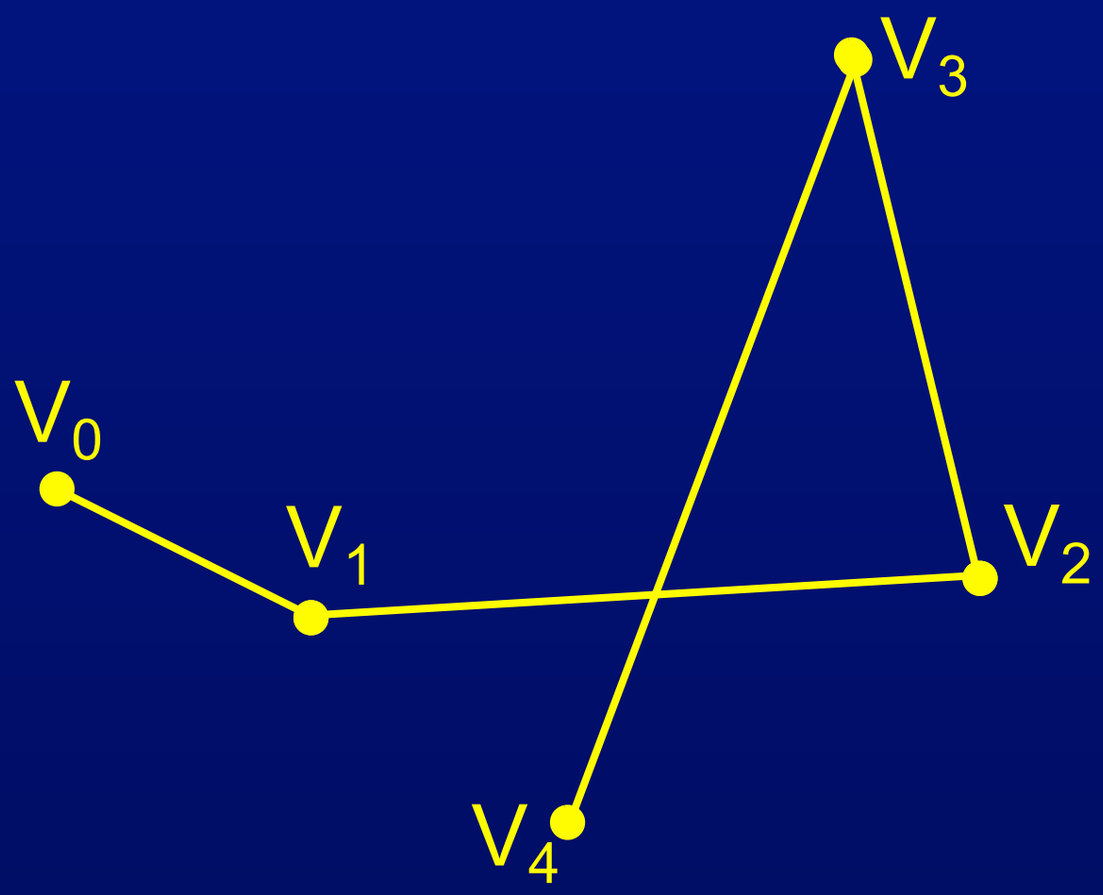
Primitives (1)

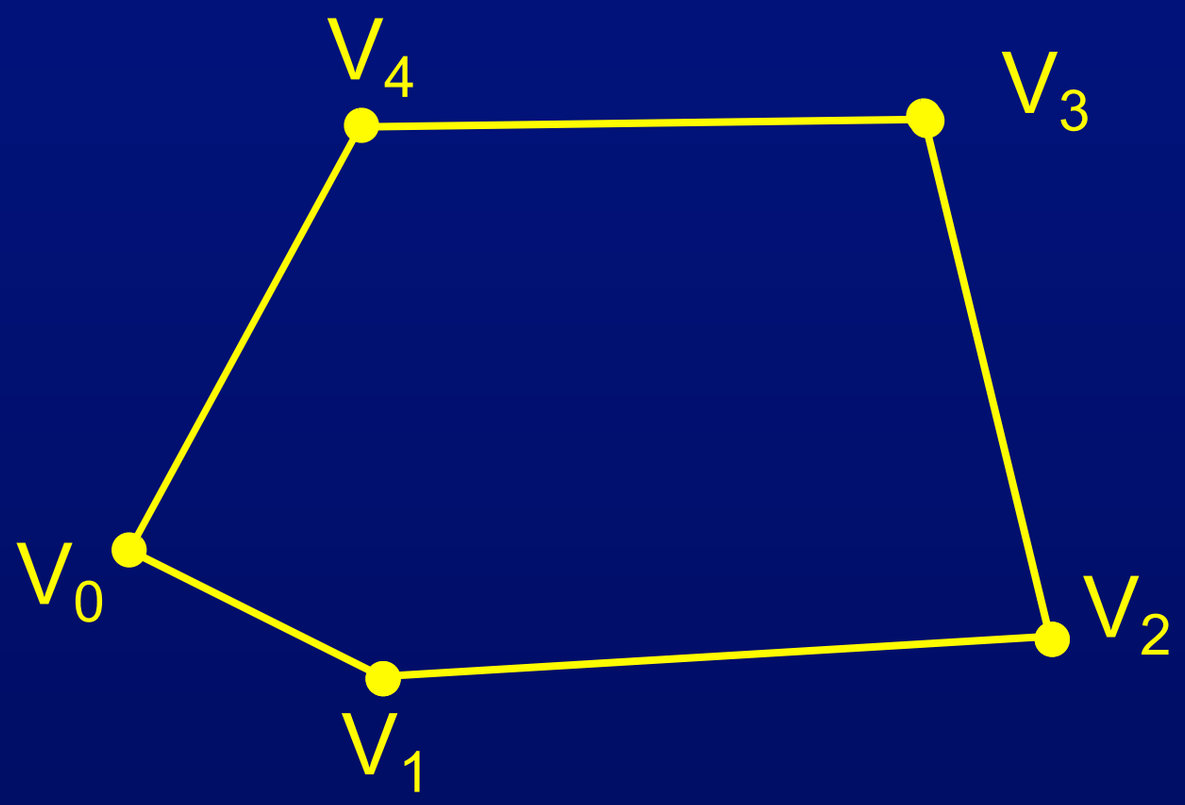
- Points
- Lines
- Triangles
- Quads
- Polygons.



Primitives (2)

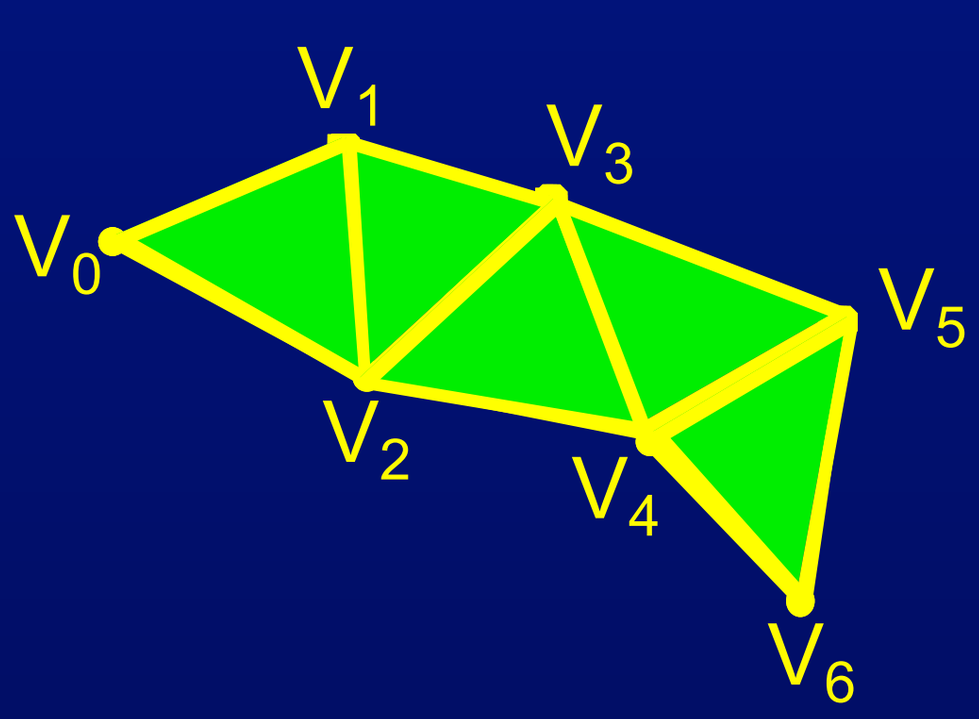
- Line strips and line loops.

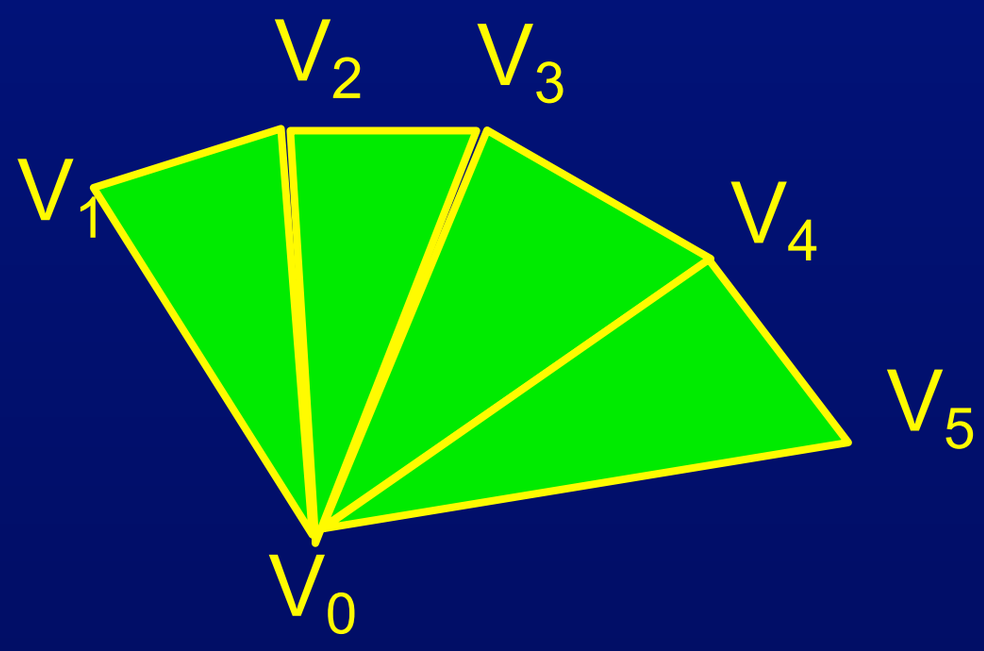




Primitives (2)

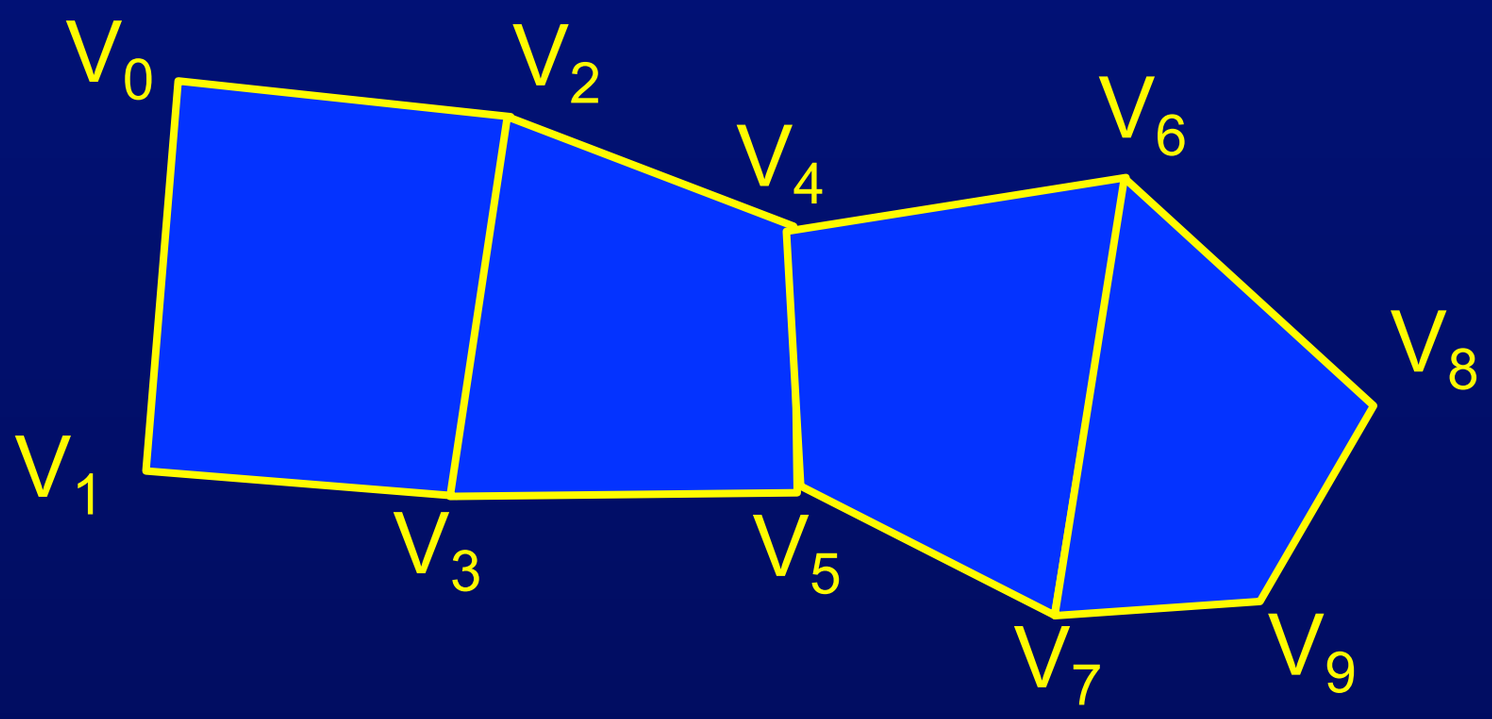
- Line strips and line loops.
- Triangle strips and fans.





Primitives (2)

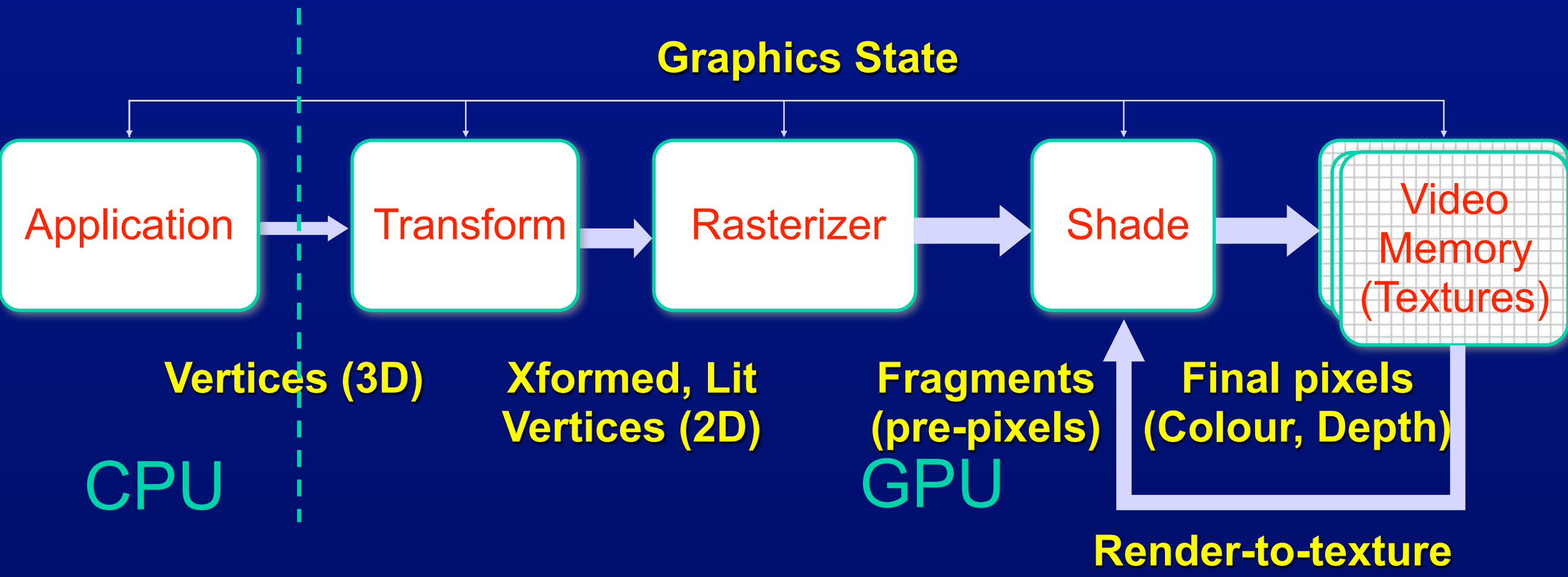
- Line strips and line loops.
- Triangle strips and fans.
- Quad strips.



Buffers

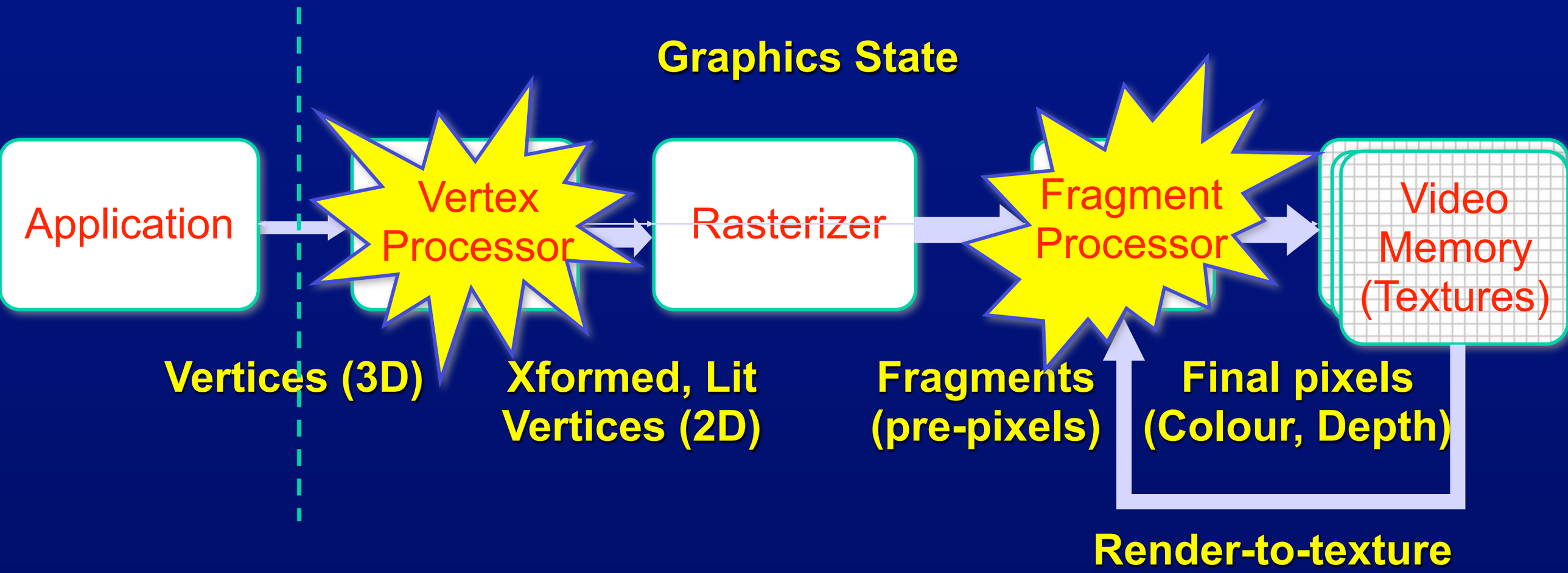
- Front and Back Buffers.
- Depth Buffer (Z-Buffer).
- Stencil Buffer.
  - Used to restrict the drawing to certain portions of the screen. Useful for constructive solid geometry.
- Accumulation Buffer.
  - Creating motion blur, and depth-of-field effects.
- Alpha Buffer.

GPU Fundamentals: The Graphics Pipeline



- A simplified graphics pipeline
  - Note that pipe widths vary
  - Many caches, FIFOs, and so on not shown

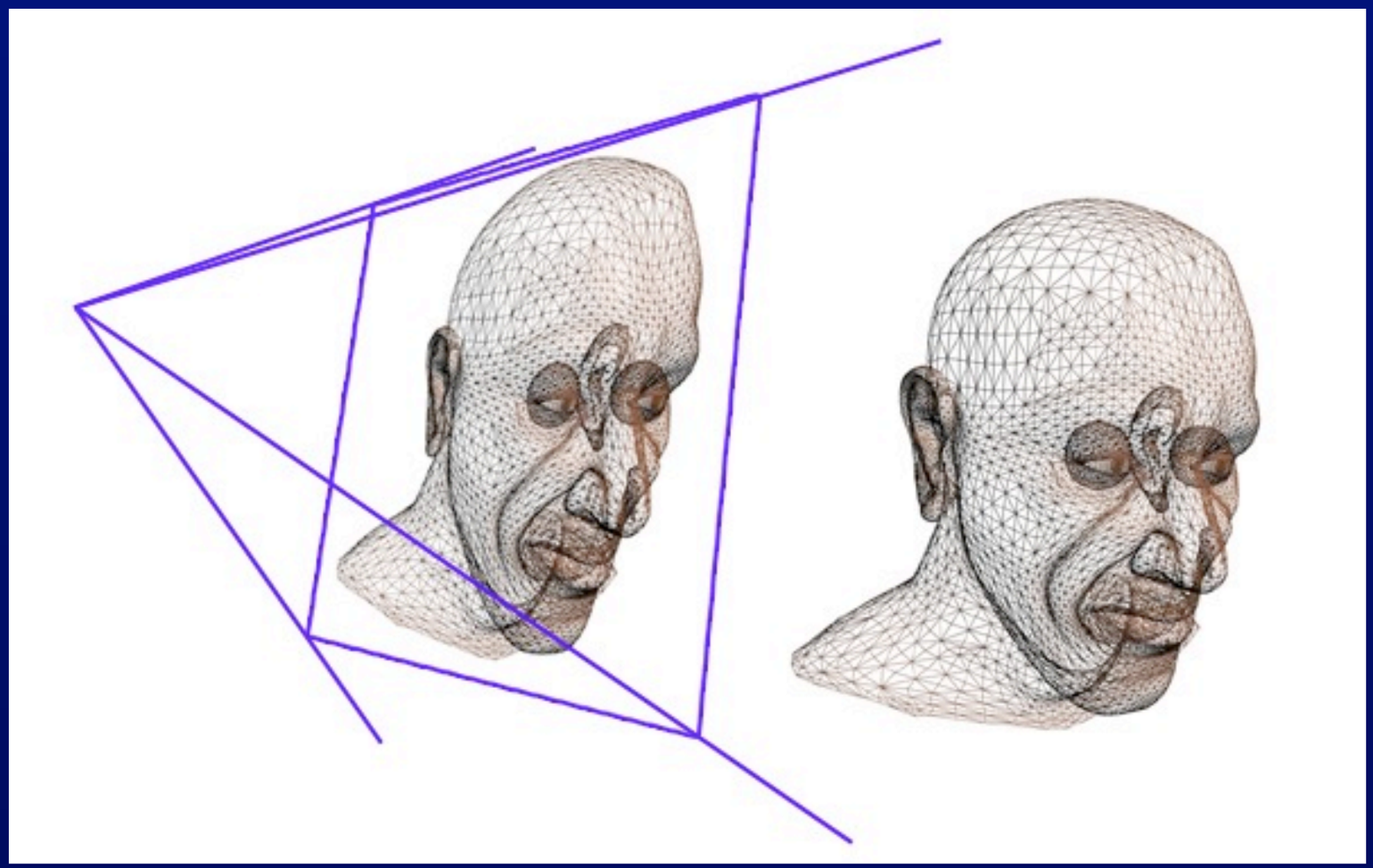
GPU Fundamentals: The *Modern* Graphics Pipeline



- Programmable vertex processor!
- Programmable pixel processor!

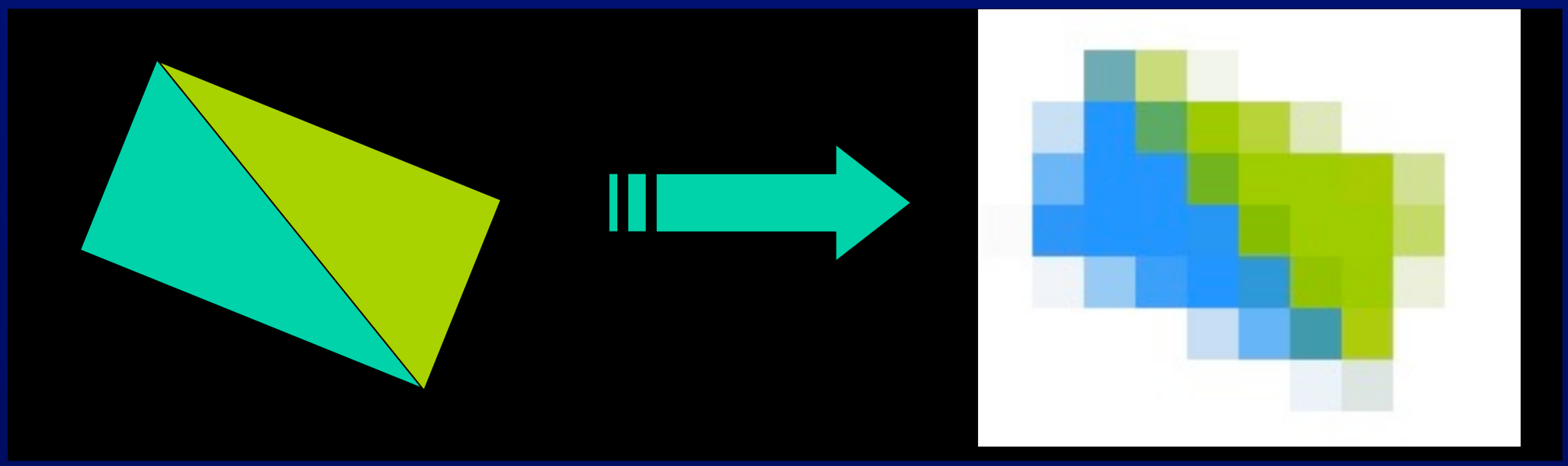
GPU Pipeline: Transform

- Vertex Processor (multiple operate in parallel)
  - –Transform from “world space” to “image space”
  - –Compute per-vertex lighting



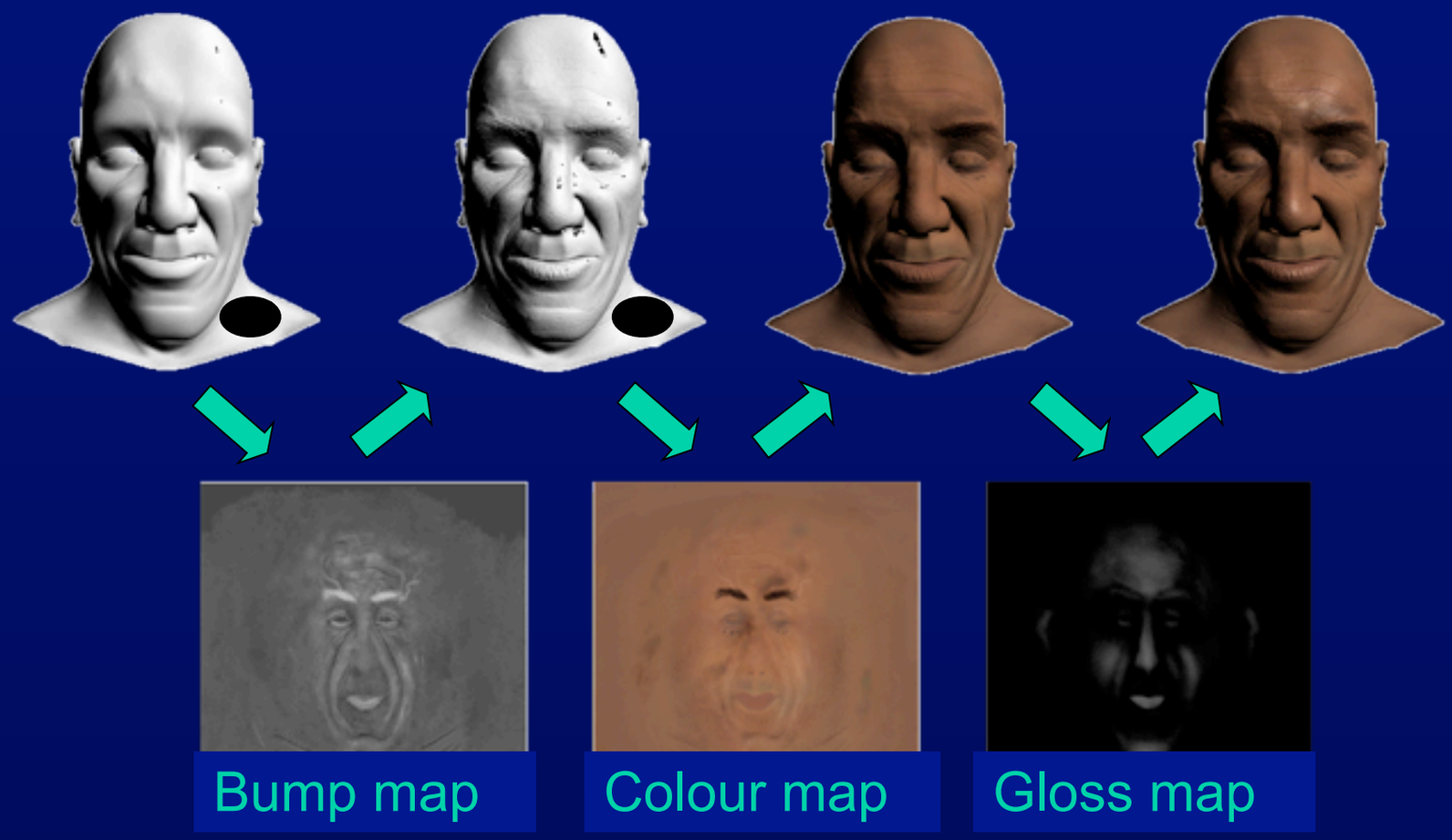
GPU Pipeline: Rasterizer

- Converts geometric representation (vertex) to image representation ((image) fragment).
- Fragment = Pixel + associated data — colour, depth, stencil, etc.
- Interpolate per-vertex quantities across pixels



GPU Pipeline: Shade

- Fragment Processors (multiple in parallel)
  - –Compute a colour for each pixel
  - –Optionally read colours from textures (images)



Less shady: CUDA, OpenCL, ...

- NVIDIA's CUDA and Khronos OpenCL
  - –Compute Unified Device Architecture
  - –Open Computing Language
- C-like languages for GPU computing
- Single instruction multiple data SIMD
- NVIDA CUDA 128 processor (2007)
- Full access to GPU programmability
- Ray tracing in hardware?

Resources

- http://www.cs.wpi.edu/~matt/courses/cs563/talks/OpenGL_Presentation/OpenGL_Presentation.html
- http://www.calsoftlabs.com:80/whitepapers/open-gl.html
- http://www.opengl.org/documentation/red_book/
- http://www.opengl.org/documentation/blue_book/
- http://www.sgi.com/products/software/opengl/
- http://fly.cc.fer.hr/~unreal/theredbook/chapter01.html