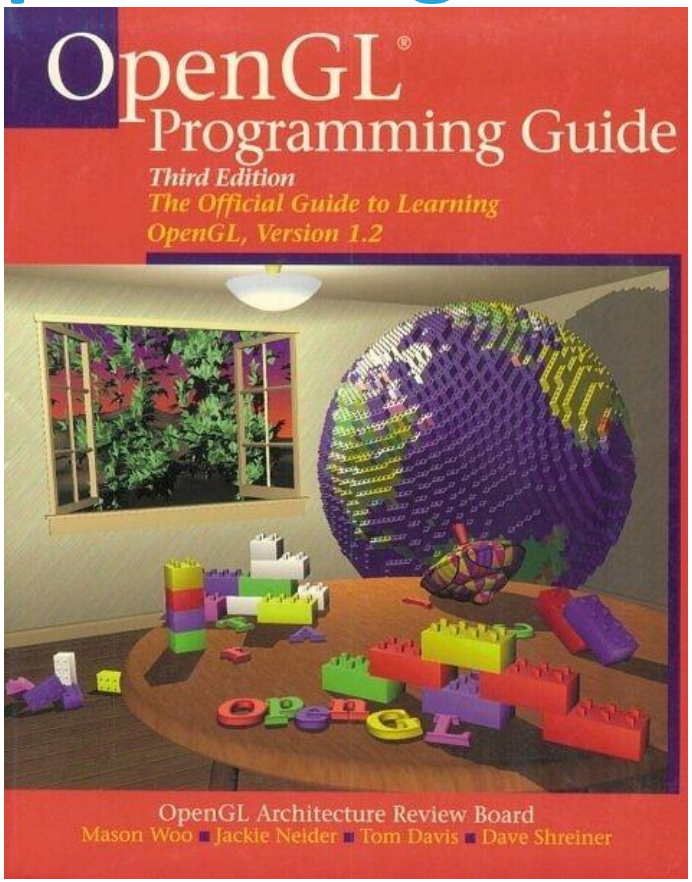
Real-Time Rendering

Graphics Programming



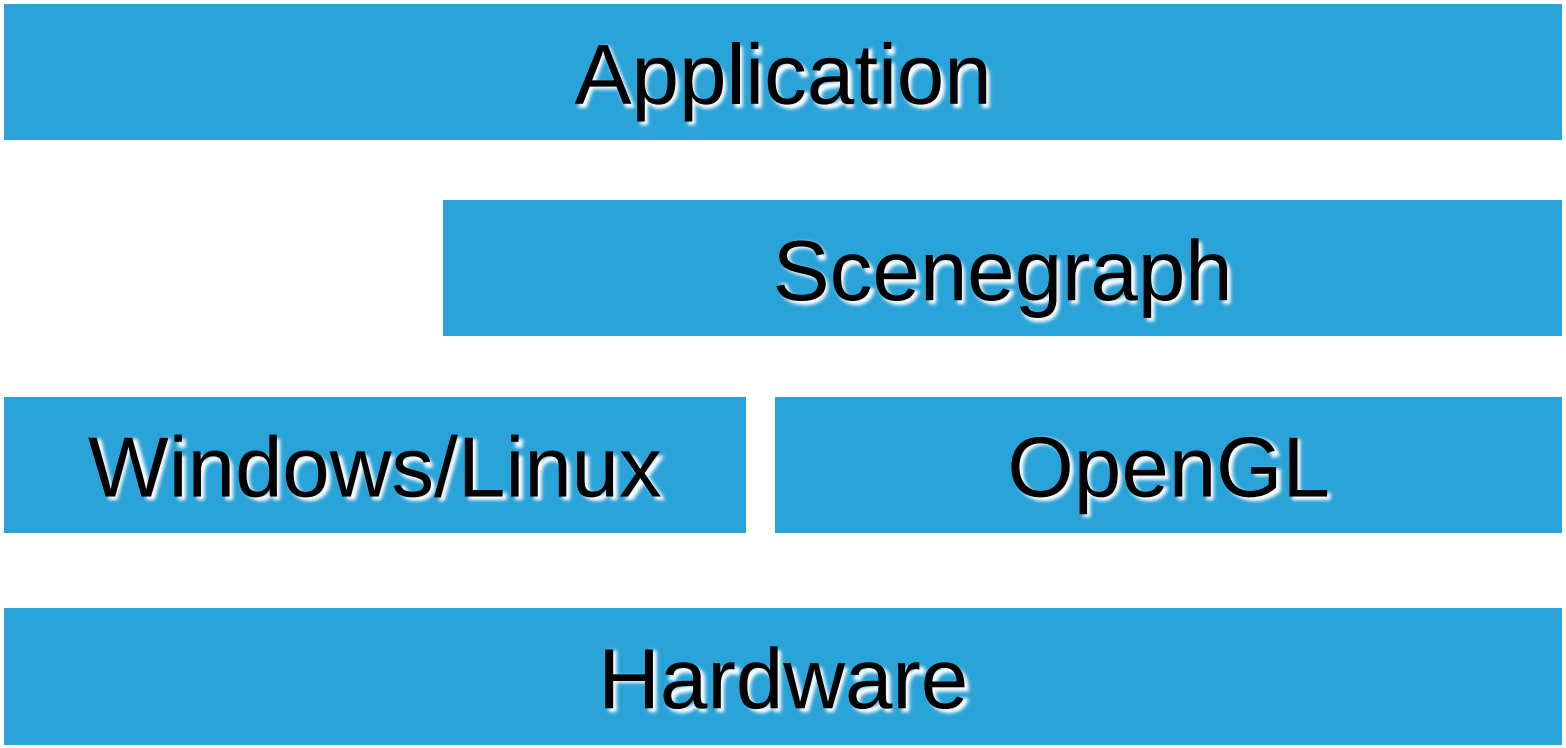
Graphics Libraries (APIs)

Give access to graphics hardware…

- Declarative (What, not How)
  - Describe the scene (e.g., scene graphs)
  - SGI Open Inventor, SGI Performer, Renderman, OpenSceneGraph…
- Imperative (How, not What)
  - Sequence of drawing commands
  - OpenGL, DirectX (Direct3D), Postscript
  - More direct control

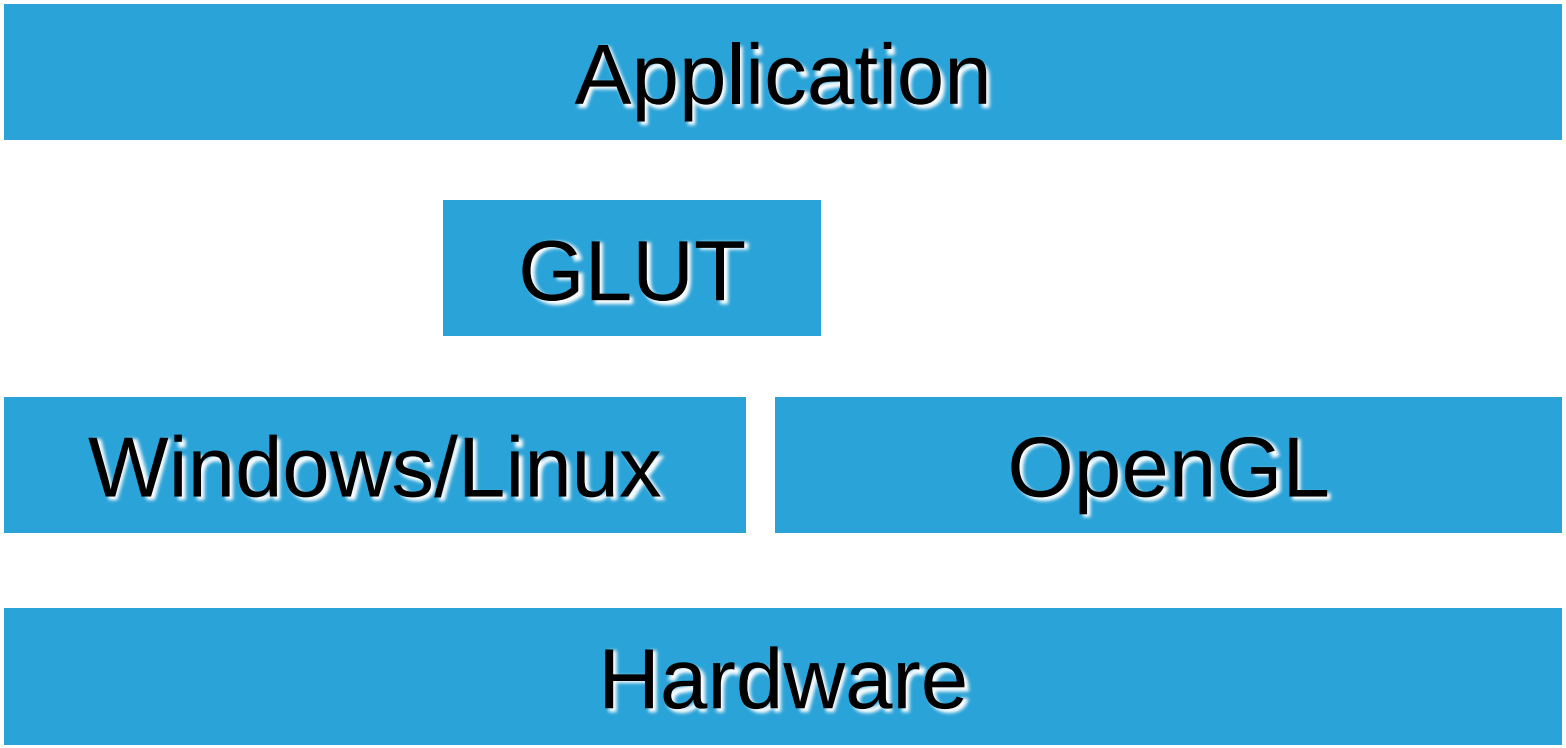
Graphics Libraries (APIs)

- Using a scene graph API…



Graphics Libraries (APIs)

- Using an immediate-mode API…



The OpenGL Graphics System

- Web site: www.opengl.org
- OpenGL trademark owned by SGI
  - More than 70 licensees
- OpenGL was controlled by the “ARB”
  - Architecture Review Board
  - Compaq, IBM, Intel, Microsoft, SGI, Evans & Sutherland, HP, Sun, NVidia, ATI, Apple
  - Meeting notes on the Web
  - follow ARB decisions, discussions, …

Khronos Group

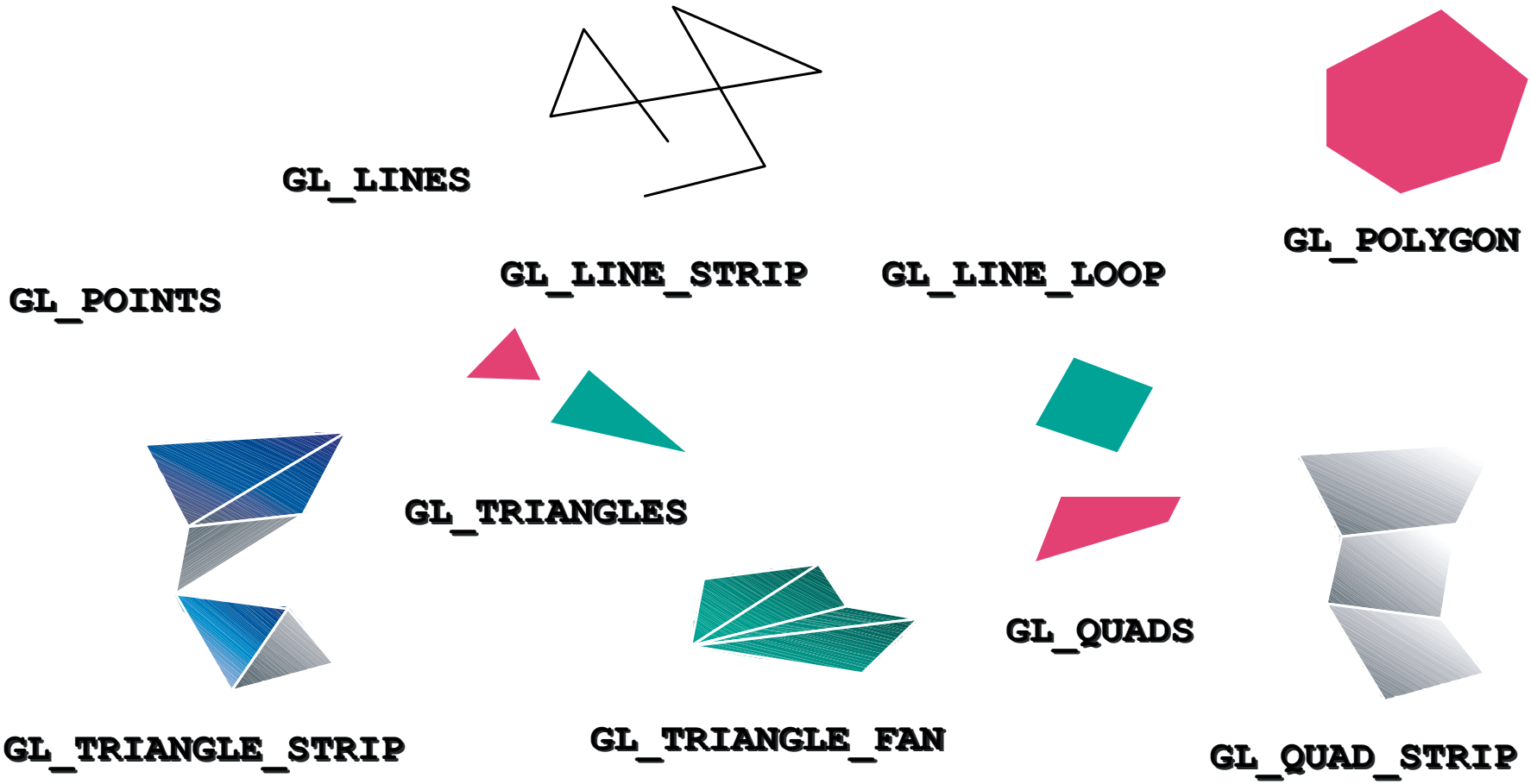
- Foundation: 2000
- Supersedes ARB
- ~100 member companies
- Many APIs
  - OpenGL (since 2006)
  - OpenGL ES
  - OpenVG
  - OpenCL
  - WebGL
  - Collada
  - Vulkan
  - …

Short History of OpenGL

1982	Silicon Graphics (SGI) incorporated
1983	IRIS GL on IRIS 1000 terminal (the predecessor to OpenGL)
1991	OpenGL ARB created
1992	OpenGL 1.0 (June 30)
1995	OpenGL 1.1
1996	OpenGL specification made public
1998	OpenGL 1.2
2000	OpenGL goes open source
2001	OpenGL 1.3
2002	OpenGL 1.4
2003	OpenGL 1.5
2004	OpenGL 2.0 (Shaders)
2008	OpenGL 3.0 (Depreciation model)
2008	OpenGL 3.0 (Depreciation model)
2009	OpenGL 3.2 (Geometry shaders)
2010	OpenGL 4.0 (Tesselation)

A Short OpenGL Freshup

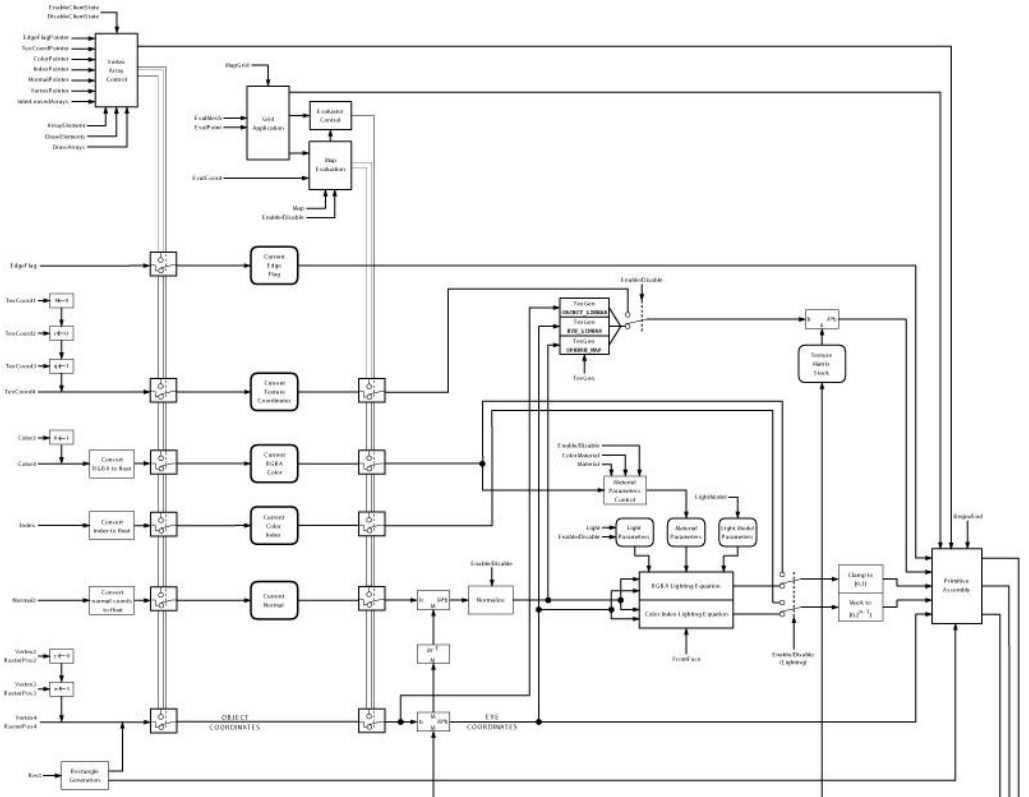
- All primitives made up of vertices…

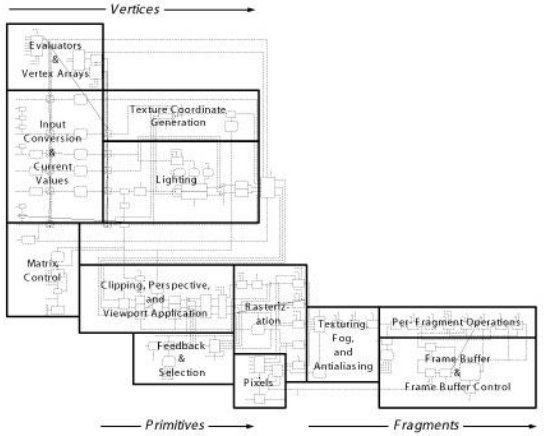


OpenGL Programming Model

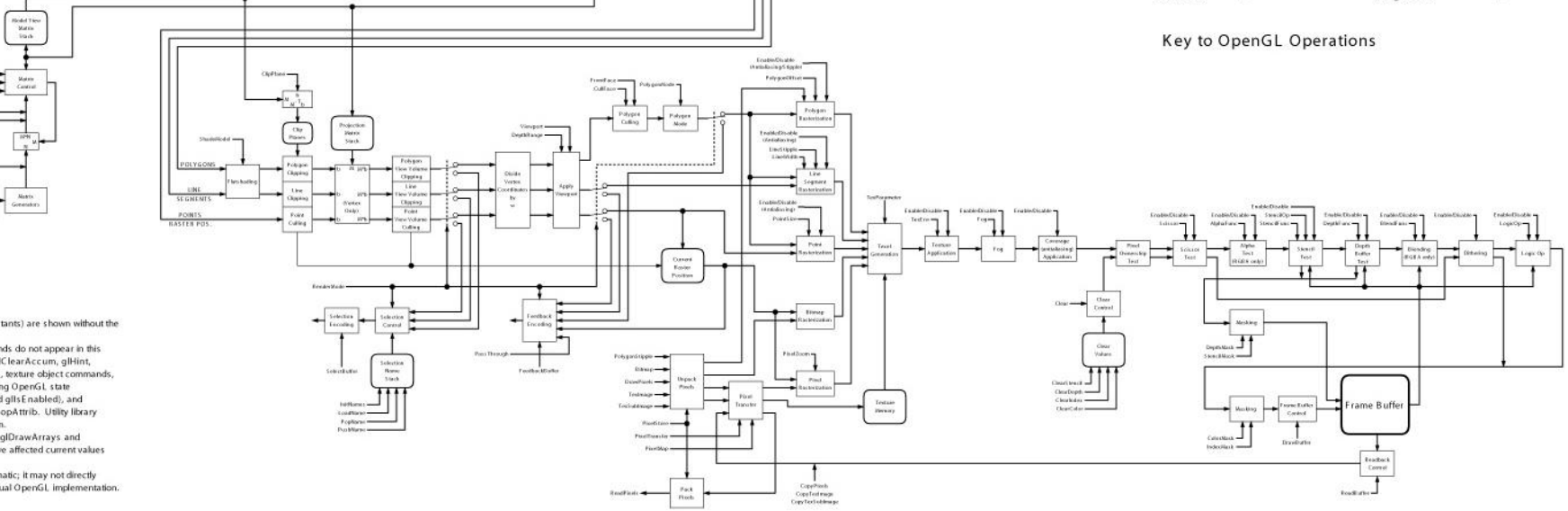
- OpenGL is a state machine
  - All commands change state
  - Fixed function: only glVertex causes action
    - This is still the “model”
  - Superseded by new “macro” commands (glDrawBuffers, glDrawElements, …)

The OpenGL® Machine





Key to OpenGL Operations



Notes:

1. Commands (and constants) are shown without the gl (or GL_) prefix.
2. The following commands do not appear in this diagram: glAccum, glClearAccum, glHint, display list commands, texture object commands, commands for obtaining OpenGL state (glGet commands and glIsEnabled), and glPushAttrib and glPopAttrib. Utility library routines are not shown.
3. After their execution, glDrawArrays and glDrawElements leave affected current values indeterminate.
4. This diagram is schematic; it may not directly correspond to any actual OpenGL implementation.

OpenGL Design Goals

- Platform independent (unlike DirectX)
  - Window-system dependent code separate (GLX, WGL)
  - Implementations on Windows, Linux, MacOS, Be, OS/2, Unix, …
  - Language independent (bindings for C, Java, Fortran, …)
- Consistency (unlike DirectX)
  - Tightly written specification
  - Conformance tests and required verification
  - Not too tight: not pixel exact
  - Invariance across passes (for correct multipass)

OpenGL Design Goals

- Complete implementations (unlike DirectX)
  - Missing hardware features emulated in software
  - Silent error recovery
- Clean interface (unlike DirectX)
  - State machine
  - Most states are orthogonal (i.e., don’t influence each other, no side effects!)
- Extensibility (unlike DirectX)
  - Favors innovation
  - New HW features first available on OpenGL!

More Goals

- High quality
- Intuitive usability (beauty counts)
- Good documentation (Programming Guide)
- Long life…

OpenGL Problems

- Extensibility
  - Different extensions for different GPUs
  - Hell for production code (games)
- Design by committee
  - Unified extension interfaces take long time
  - Very slow to adopt non-GPU specific features (e.g., offscreen buffers)
- Non-existent toolset
  - Shading debuggers (but: gDebugger)
  - Performance tools (but: NVIDIA Parallel NSight)
  - Mesh tools (already included in DirectX)
- Mediocre driver support

OpenGL Extensions

- Khronos maintains central registry
- Carefully documented
  - Takes into account previous extensions
  - New OpenGL version could be implemented by applying all extensions
- A bit difficult to read
  - Read overview, then “Additions to…”
- Very stable process
  - Extensions are refined and improved…

OpenGL Extension Categories

- Proprietary: suffixed with vendor
  - e.g., SGIS_texture_lod, NV_fragment_program
- EXT suffix
  - Implemented by at least 2 vendors (usually NV,AMD)
  - e.g. EXT_blend_func_separate
- ARB suffix
  - Specification controlled by ARB
  - ARB_multitexture
- 1.x: no suffix
  - Required feature for version 1.x

Name

EXT_stencil_wrap

Name Strings

GL_EXT_stencil_wrap

Version

Date: 4/4/2002 Version 1.2

Number

176

Dependencies

None

Overview

Various algorithms use the stencil buffer to "count" the number of surfaces that a ray passes through. As the ray passes into an object, the stencil buffer is incremented. As the ray passes out of an object, the stencil buffer is decremented.

GL requires that the stencil increment operation clamps to its maximum value. For algorithms that depend on the difference between the sum of the increments and the sum of the decrements, clamping causes an erroneous result.

This extension provides an enable for both maximum and minimum wrapping of stencil values. Instead, the stencil value wraps in both directions.

surfaces that a ray passes through. As the ray passes into an object, the stencil buffer is incremented. As the ray passes out of an object, the stencil buffer is decremented.

GL requires that the stencil increment operation clamps to its maximum value. For algorithms that depend on the difference between the sum of the increments and the sum of the decrements, clamping causes an erroneous result.

This extension provides an enable for both maximum and minimum wrapping of stencil values. Instead, the stencil value wraps in both directions.

Two additional stencil operations are specified. These new operations are similiar to the existing INCR and DECR operations, but they wrap their result instead of saturating it. This functionality matches the new stencil operations introduced by DirectX 6.

New Procedures and Functions

None

New Tokens

Accepted by the <sfail>, <dpfail>, and <dppass> parameter of StencilOp:

INCR_WRAP_EXT	0x8507
DECR_WRAP_EXT	0x8508

Additions to Chapter 2 of the GL Specification (OpenGL Operation)

None

Additions to Chapter 3 of the GL Specification (Rasterization)

None

Additions to Chapter 4 of the GL Specification (Per-Fragment Operations and the Framebuffer)

Section 4.1.4 "Stencil Test" (page 144), change the 3rd paragraph to read:

"... The symbolic constants are KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP_EXT, and DECR_WRAP_EXT. The correspond to keeping the current value, setting it to zero, replacing it with the reference value, incrementing it with saturation, decrementing it with saturation, bitwise inverting it, incrementing it without saturation, and decrementing it without saturation. For purposes of incrementing and decrementing, the stencil bits are considered as an unsigned integer. Incrementing or decrementing with saturation will clamp values at 0 and the maximum representable value. Incrementing or decrementing without saturation will wrap such that incrementing the maximum representable value results in 0 and decrementing 0 results in the maximum representable value. ..."

Additions to Chapter 5 of the GL Specification (Special Functions)

None

Additions to Chapter 6 of the GL Specification (State and State Requests)

None

Additions to the GLX Specification

None

None

Additions to Chapter 6 of the GL Specification (State and State Requests)

None

Additions to the GLX Specification

None

GLX Protocol

None

Errors

INVALID_ENUM is generated by StencilOp if any of its parameters are not KEEP, ZERO, REPLACE, INCR, DECR, INVERT, INCR_WRAP_EXT, or DECR_WRAP_EXT.

New State

(table 6.15, page 205)

Get Value	Type	Get Command	Initial Value	Sec	Attribute
STENCIL_FAIL	Z8	GetIntegerv	KEEP	4.1.4	stencil-buffer
STENCIL_PASS_DEPTH_FAIL	Z8	GetIntegerv	KEEP	4.1.4	stencil-buffer
STENCIL_PASS_DEPTH_PASS	Z8	GetIntegerv	KEEP	4.1.4	stencil-buffer

NOTE: the only change is that Z6 type changes to Z8

New Implementation Dependent State

None

Using Extensions

- Get `glext.h` from www.opengl.org
- Check for extension availability
- Acquire function pointer(s) (only Win32)
- **Easier**: google “opengl loading library”

```
#include <GL/glut.h>
#include <GL/glext.h>

PFNGLDRAWRANGEELEMENTSEXTPROC glDrawRangeElementsEXT;

if (glutExtensionSupported( "GL_EXT_draw_range_elements" )
{
    glDrawRangeElementsEXT = (PFNGLDRAWRANGEELEMENTSEXTPROC)
        wglGetProcAddress( "glDrawRangeElementsEXT" );
}
```


OpenGL 2.0

- Main novelty: shading language GLSL
- Vertex and fragment shaders
  - Replace fixed functionality
- Shader: high-level language (C-like)
- OpenGL driver: compiler and linker for shaders
- Vertex-, texture coordinates etc.:
  abstract input values to shader function
- Arbitrary calculations possible
- Requires DX9 (GeforceFX/6) cards

OpenGL 3.0

- Not much new
- Vertex Array Objects (encapsulate VBO state)
- Framebuffer objects (offscreen rendering)
- sRGB framebuffers
- Texture arrays
- Transform feedback
- Conditional rendering
- Extensions: geometry shaders, instancing, ...
- Depreciation mechanism!

OpenGL 3.2

- Geometry shaders
- Synchronization primitives
- Core profile/compatibility profile

OpenGL 4.0/3.3

- Tessellation
- Timer queries
- Double precision floating point
- Etc.
- OpenGL 3.3: for compatibility with older hardware

OpenGL 4+

- OpenGL 4.1: minor stuff (OpenGL ES 2.0 compatibility)
- OpenGL 4.2: minor stuff (atomic counters, …)
- OpenGL 4.3: Compute Shaders, shader buffers, debugging, OpenGL ES 3.0 compatibility, texture views
- OpenGL 4.4: minor stuff (bindless textures, memory transfer optimizations, …)
- OpenGL 4.5: even more minor (OpenGL ES 3.1)

OpenGL ES

- For embedded systems
- Reduced instruction set
- Developers love it ☺
- OpenGL 4.3 is backwards compatible with OpenGL ES 3.0!

Vulkan

- Designated OpenGL successor
- Adapted from AMD Mantle
- Binary intermediate format for shaders (SPIR)
- Client-controlled command buffers
  - ```
    VK_CMD_BUFFER_BEGIN_INFO info = { ... };
    vkBeginCommandBuffer(cmdBuf, &info);
    vkCmdDoThisThing(cmdBuf, ...);
    vkCmdDoSomeOtherThing(cmdBuf, ...);
    vkEndCommandBuffer(cmdBuf);
    ```
- Render passes
  - ```
    VK_RENDER_PASS_CREATE_INFO info = { ... };
    VK_RENDER_PASS renderPass;
    vkCreateRenderPass(device, &info, &renderPass);
    ```