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…

Application

Scenegraph

Windows/Linux  OpenGL

Hardware
Graphics Libraries (APIs)

- Using an immediate-mode API...

Application

GLUT

Windows/Linux

OpenGL

Hardware
Immediate vs. Retained

- Immediate (OpenGL)
  - Total control over rendering
    - traversal and data structures and formats
  - Many subroutine calls
  - Driver cannot optimize

- Retained (scene graph, display lists)
  - Driver can optimize execution of display lists
  - API can optimize traversal of scene graph
    - for different platforms!
  - Traversal is difficult to change
The OpenGL Graphics System

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

- All primitives made up of vertices…
Shading:
From wire frame to texture mapped…

- Wire frame (hidden line) flat shading
- Gouraud Textured (Combination)
### Short History of OpenGL

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1982</td>
<td>Silicon Graphics (SGI) incorporated</td>
</tr>
<tr>
<td>1983</td>
<td>IRIS GL on IRIS 1000 terminal</td>
</tr>
<tr>
<td></td>
<td>(the predecessor to OpenGL)</td>
</tr>
<tr>
<td>1991</td>
<td>OpenGL ARB created</td>
</tr>
<tr>
<td>1992</td>
<td>OpenGL 1.0 (June 30)</td>
</tr>
<tr>
<td>1995</td>
<td>OpenGL 1.1</td>
</tr>
<tr>
<td>1996</td>
<td>OpenGL specification made public</td>
</tr>
<tr>
<td>1998</td>
<td>OpenGL 1.2</td>
</tr>
<tr>
<td>2000</td>
<td>OpenGL goes open source</td>
</tr>
<tr>
<td>2001</td>
<td>OpenGL 1.3</td>
</tr>
<tr>
<td>2002</td>
<td>OpenGL 1.4</td>
</tr>
<tr>
<td>2003</td>
<td>OpenGL 1.5</td>
</tr>
<tr>
<td>2004</td>
<td>OpenGL 2.0 (Shaders)</td>
</tr>
<tr>
<td>2008</td>
<td>OpenGL 3.0 (Depreciation model)</td>
</tr>
<tr>
<td>2008</td>
<td>OpenGL 3.0 (Depreciation model)</td>
</tr>
<tr>
<td>2009</td>
<td>OpenGL 3.2 (Geometry shaders)</td>
</tr>
<tr>
<td>2010</td>
<td>OpenGL 4.0 (Tessellation)</td>
</tr>
</tbody>
</table>
Modern Graphics Pipeline

Application → Command → Geometry → Rasterization → Texture → Fragment → Display
OpenGL Programming Model

- OpenGL is a state machine
  - All command change state
  - Only glVertex causes action
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 9)
  - 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 9)
  - Missing hardware features emulated in software
  - Silent error recovery
- Clean interface (unlike DirectX 9)
  - State machine
  - Most states are orthogonal (i.e., don’t influence each other, no side effects!)
- Extensibility (unlike DirectX 9)
  - 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: NVPerfKit)
  - Mesh tools (already included in DirectX)

- Mediocre driver support
Open GL Extensions

- SGI maintains central registry
- Carefully documented
  - Takes into account previous extensions
  - New Open GL 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
EXT_stencil_wrap

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 similar 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. They 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
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, DBCR, INVERT, INCR_WRAP_EXT, or DECR_WRAP_EXT.

New State

(table 6.15, page 205)

<table>
<thead>
<tr>
<th>Get Value</th>
<th>Type</th>
<th>Get Command</th>
<th>Initial Value</th>
<th>Sec</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>STENCIL_PAIL</td>
<td>Z8</td>
<td>GetIntegerv</td>
<td>KEEP</td>
<td>4.1.4</td>
<td>stencil-buffer</td>
</tr>
<tr>
<td>STENCIL_PASS_DEPTH_PAIL</td>
<td>Z8</td>
<td>GetIntegerv</td>
<td>KEEP</td>
<td>4.1.4</td>
<td>stencil-buffer</td>
</tr>
<tr>
<td>STENCIL_PASS_DEPTH_PASS</td>
<td>Z8</td>
<td>GetIntegerv</td>
<td>KEEP</td>
<td>4.1.4</td>
<td>stencil-buffer</td>
</tr>
</tbody>
</table>

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**: use glew or glee

```cpp
#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
- sRGB framebuffers
- Texture arrays
- Transform feedback
- Extensions: geometry shaders, instancing, ...
- Depreciation mechanism!
OpenGL 3.2

- Geometry shaders
OpenGL 4.0/3.3

- Tessellation
- Timer queries
- Double precision floating point
- Etc.

- OpenGL 3.3: for compatibility with older hardware
OpenGL ES

- For embedded systems
- Reduced instruction set
- Developers love it 😊
- OpenGL 4.1 is backwards compatible with OpenGL ES!
OpenGL Architecture

- Symmetric geometry and image paths
- Operations independent ("orthogonal")
Typical OpenGL Block Diagram
They match!

Application
- Command
- Geometry
- Rasterization
- Texture
- Fragment
- Display