Performance OpenGL

Platform Independent Techniques

Dave Shreiner

shreiner@sgi.com

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

What You'll See Today …

- An in-depth look at the OpenGL pipeline from a performance perspective
- Techniques for determining where OpenGL application performance bottlenecks are
- A bunch of simple, good habits for OpenGL applications

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Performance Tuning Assumptions

- You’re trying to tune an interactive OpenGL application
- There’s an established metric for estimating the application’s performance
  - Consistent frames/second
  - Number of pixels or primitives to be rendered per frame
- You can change the application’s source code

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

3

Errors – The Silent Performance Killers

Asynchronous Error Reporting

- OpenGL doesn’t tell you when something goes wrong
  - Calls will silently mark an error and return
  - Need to use **`glGetError()`** to determine if something went wrong

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

4

Checking for Errors

Check Early and Often in Application Development

- Only first error* is retained
  - Additional errors are discarded until error flag is cleared by calling **`glGetError()`**
- Erroneous OpenGL function skipped

Checking a single command

Simple Macro

```
#define CHECK_OPENGL_ERROR( cmd ) \
     cmd; \
     { GLenum  error; \
       while ( (error = glGetError()) != GL_NO_ERROR) { \
         printf( "[%s:%d] '%s' failed with error %s\n", \
                __FILE__, __LINE__, #cmd, \
                gluErrorString(error) ); \
     }
```

- Some limitations on where the macro can be used
  - can't use inside of **`glBegin()`** / **`glEnd()`** pair

Checking More Thoroughly

Modified `gl.h` checks almost every situation

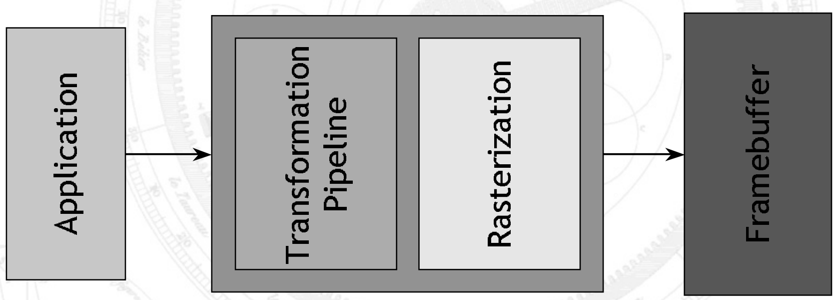
```
#define glBegin( mode ) \
  if ( __glDebug_InBegin ) { \
    printf( "[%s:%d] glBegin( %s ) called between” \
            “glBegin()/glEnd() pair\n", \
          __FILE__, __LINE__, #mode ); \
  } else { \
    __glDebug_InBegin = GL_TRUE; \
    glBegin( mode ); \
  }
```

- Script for re-writing **`gl.h`** available from web site

OpenGL

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

The OpenGL Pipeline
(The Macroscopic View)



OpenGL

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Performance Bottlenecks

Bottlenecks are the performance limiting part of the application

- *Application* bottleneck
  - Application may not pass data fast enough to the OpenGL pipeline
- *Transform-limited* bottleneck
  - OpenGL may not be able to process vertex transformations fast enough

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

9

Performance Bottlenecks (cont.)

- *Fill-limited* bottleneck
  - OpenGL may not be able to rasterize primitives fast enough

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

10

There Will Always Be A Bottleneck

Some portion of the application will always be the limiting factor to performance

- If the application performs to expectations, then the bottleneck isn't a problem
- Otherwise, need to be able to identify which part of the application is the bottleneck
- We'll work backwards through the OpenGL pipeline in resolving bottlenecks

OpenGL

11

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Fill-limited Bottlenecks

System cannot fill all the pixels required in the allotted time

- Easiest bottleneck to test
- Reduce number of pixels application must fill
  - Make the viewport smaller

OpenGL

12

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Reducing Fill-limited Bottlenecks

The Easy Fixes

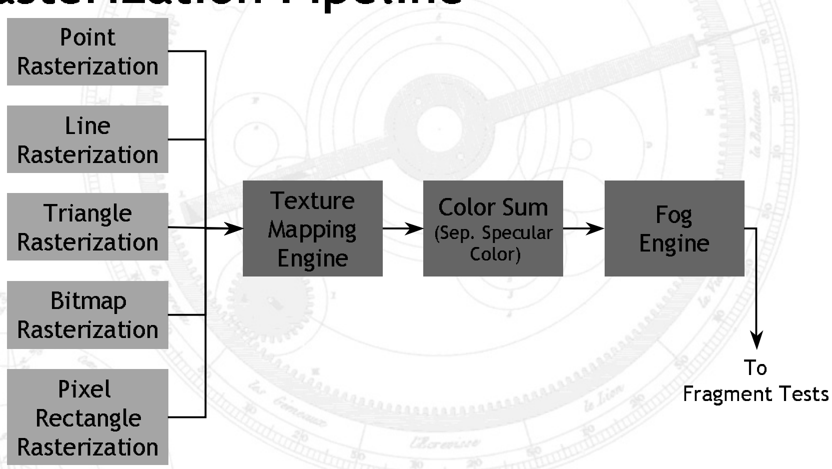
- Make the viewport smaller
  - This may not be an acceptable solution, but it's easy
- Reduce the frame-rate

OpenGL.

13

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

A Closer Look at OpenGL's Rasterization Pipeline

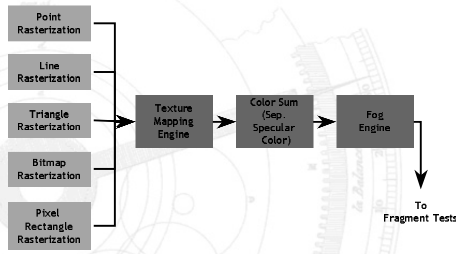


OpenGL.

14

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Reducing Fill-limited Bottlenecks (cont.)



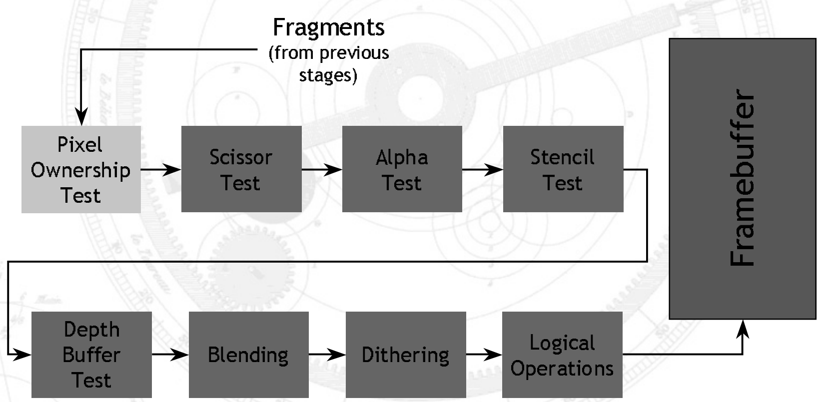
Rasterization Pipeline

- Cull back facing polygons
  - Does require all primitives have same facediness
- Use a simpler texture filter
  - Particularly on objects that occupy small screen area
    - far from the viewer
- Use per-vertex fog, as compared to per-pixel

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

15

A Closer Look at OpenGL's Rasterization Pipeline (cont.)

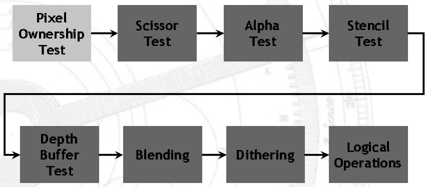


OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

16

Reducing Fill-limited Bottlenecks (cont.)

Fragment Pipeline



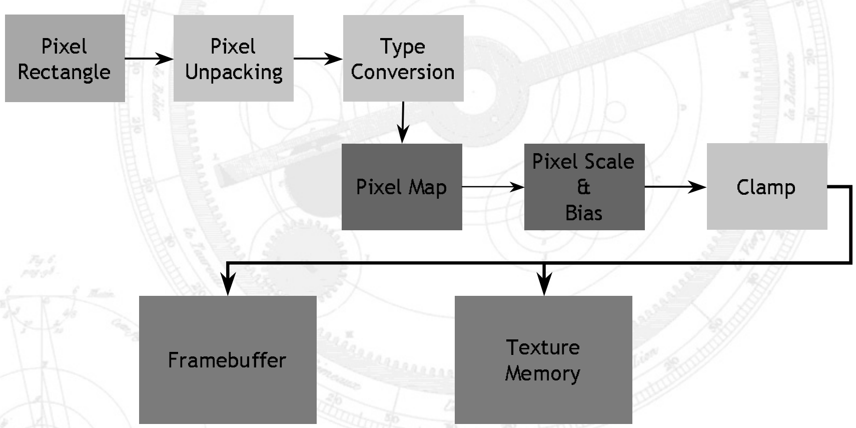
- Do less work per pixel
  - Disable dithering
  - Depth-sort primitives to reduce depth testing
  - Use alpha test to reject transparent fragments
    - saves doing a pixel read-back from the framebuffer in the blending phase

OpenGL

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

17

A Closer Look at OpenGL's Pixel Pipeline



OpenGL

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

18

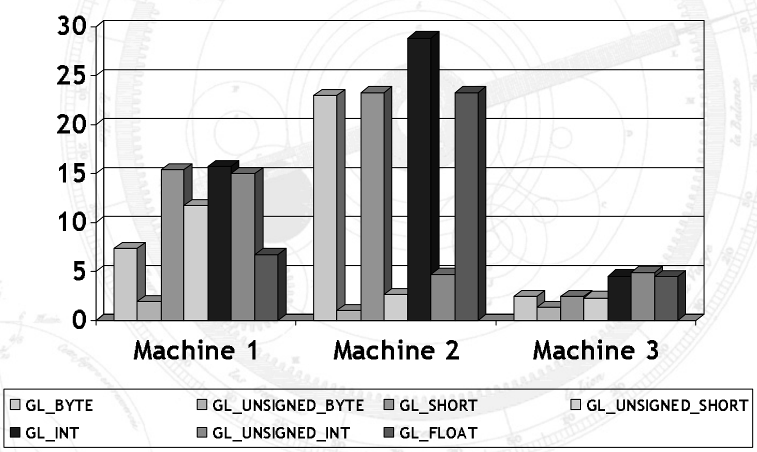
Working with Pixel Rectangles

Texture downloads and Blts

- OpenGL supports many formats for storing pixel data
  - Signed and unsigned types, floating point
- Type conversions from storage type to framebuffer / texture memory format occur automatically

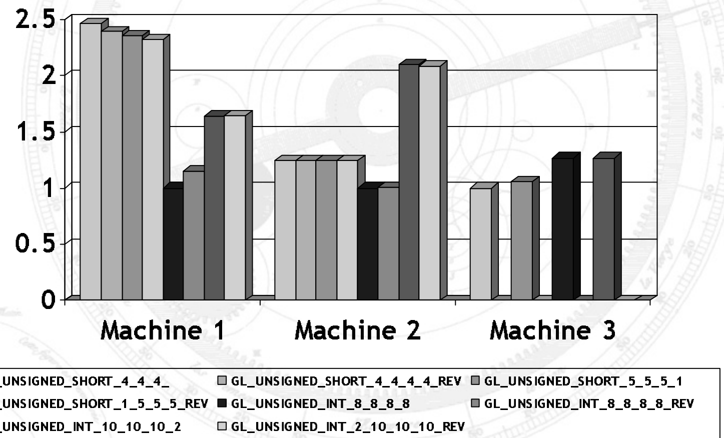
19

Pixel Data Conversions



20

Pixel Data Conversions (cont.)



OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

21

Pixel Data Conversions (cont.)

Observations

- Signed data types probably aren't optimized
  - OpenGL clamps colors to [0, 1]
- Match pixel format to window's pixel format for blts
  - Usually involves using *packed pixel formats*
  - No significant difference for rendering speed for texture's internal format

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

22

Texture-mapping Considerations

Use Texture Objects

- Allows OpenGL to do texture memory management
  - Loads texture into texture memory when appropriate
  - Only convert data once
- Provides queries for checking if a texture is resident
  - Load all textures, and verify they all fit simultaneously

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

23

Texture-mapping Considerations (cont.)

Texture Objects (cont.)

- Assign priorities to textures
  - Provides hints to texture-memory manager on which textures are most important
- Can be shared between OpenGL contexts
  - Allows one thread to load textures; other thread to render using them
- Requires OpenGL 1.1

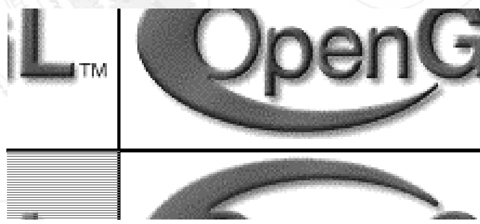
OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

24

Texture-mapping Considerations (cont.)

Sub-loading Textures

- Only update a portion of a texture
  - Reduces bandwidth for downloading textures
  - Usually requires modifying texture-coordinate matrix



Texture-mapping Considerations (cont.)

Know what sizes your textures need to be

- What sizes of mipmaps will you need?
- OpenGL 1.2 introduces texture *level-of-detail*
  - Ability to have fine grained control over mipmap stack
    - Only load a subset of mipmaps
    - Control which mipmaps are used

What If Those Options Aren't Viable?

- Use more or faster hardware
- Utilize the "extra time" in other parts of the application
  - Transform pipeline
    - tessellate objects for smoother appearance
    - use better lighting
  - Application
    - more accurate simulation
    - better physics

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

27

Transform-limited Bottlenecks

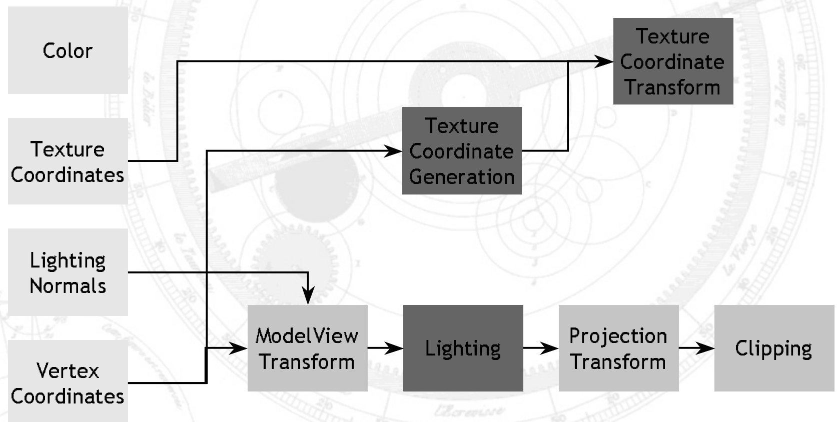
System cannot process all the vertices required in the allotted time

- If application doesn't speed up in fill-limited test, it's most likely transform-limited
- Additional tests include
  - Disable lighting
  - Disable texture coordinate generation

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

28

A Closer Look at OpenGL's Transformation Pipeline



OpenGL

29

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Reducing Transform-limited Bottlenecks

Do less work per-vertex

- Tune lighting
- Use "typed" OpenGL matrices
- Use explicit texture coordinates
- Simulate features in texturing
  - lighting

OpenGL

30

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

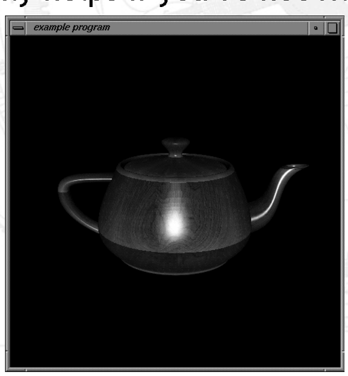
Lighting Considerations

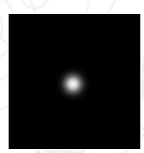
- Use infinite (directional) lights
  - Less computation compared to local (point) lights
  - Don’t use **`GL_LIGHTMODEL_LOCAL_VIEWER`**
- Use fewer lights
  - Not all lights may be hardware accelerated

31

Lighting Considerations (cont.)

- Use a texture-based lighting scheme
  - Only helps if you’re not fill-limited





32

Reducing Transform-limited Bottlenecks (cont.)

Matrix Adjustments

- Use “typed” OpenGL matrix calls

“Typed”	“Untyped”

`glRotate*()`
`glScale*()`
`glTranslate*()`
`glLoadIdentity()`
	`glLoadMatrix*()`
	`glMultMatrix*()`

- Some implementations track matrix type to reduce matrix-vector multiplication operations

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

33

Application-limited Bottlenecks

When OpenGL does all you ask, and your application still runs too slow

- System may not be able to transfer data to OpenGL fast enough
- Test by modifying application so that no rendering is performed, but all data is still transferred to OpenGL

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

34

Application-limited Bottlenecks (cont.)

- Rendering in OpenGL is triggered when vertices are sent to the pipe
- Send *all* data to pipe, just not necessarily in its original form
  - Replace all **glVertex*()** calls with **glNormal*()** calls
    - **glNormal*()** only sets the current vertex's normal values, but transfers the same amount of data

OpenGL

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

35

Reducing Application-limited Bottlenecks

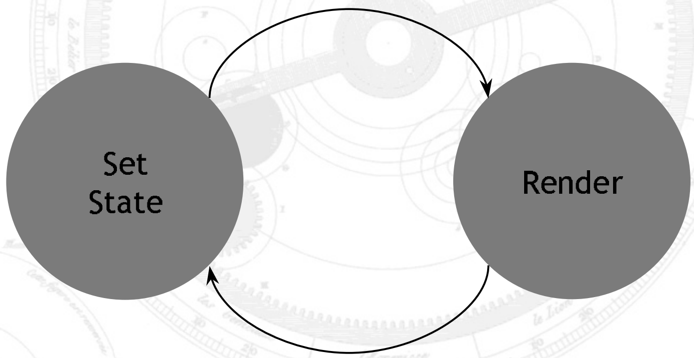
- No amount of OpenGL transform or rasterization tuning will help the problem
- Revisit application design decisions
  - Data structures
  - Traversal methods
  - Storage formats
- Use an application profiling tool (e.g. **pixie** & **prof**, **gprof**, or other similar tools)

OpenGL

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

36

The Novice OpenGL Programmer's View of the World



37

What Happens When You Set OpenGL State

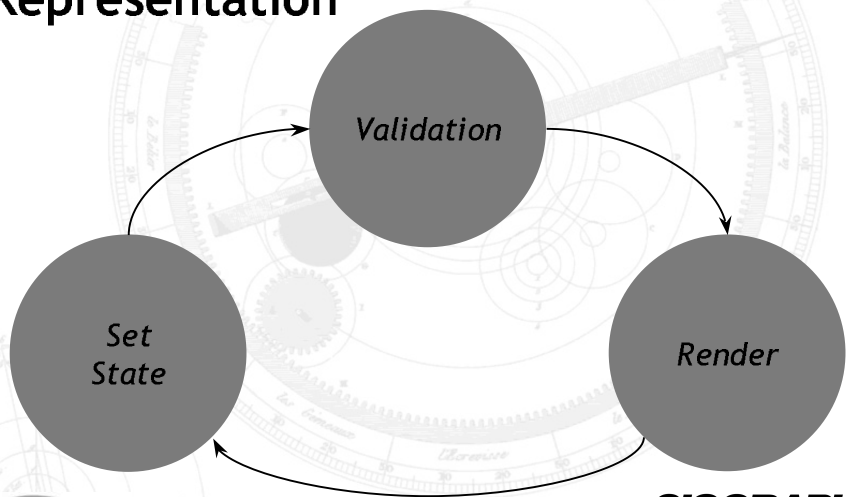
- The amount of work varies by operation

Turning on or off a feature (`glEnable()`)	**Set the feature's enable flag**
Set a "typed" set of data (`glMaterialfv()`)	**Set values in OpenGL's context**
Transfer "untyped" data (`glTexImage2D()`)	**Transfer and convert data from host format into internal representation**

- But all request a validation at next rendering operation

38

A (Somewhat) More Accurate Representation



39

Validation

OpenGL's synchronization process

- *Validation* occurs in the transition from state setting to rendering

```
glMaterial( GL_FRONT, GL_DIFFUSE, blue );
glEnable( GL_LIGHT0 );
glBegin( GL_TRIANGLES );
```

- Not all state changes trigger a validation
  - Vertex data (e.g. color, normal, texture coordinates)
  - Changing rendering primitive

40

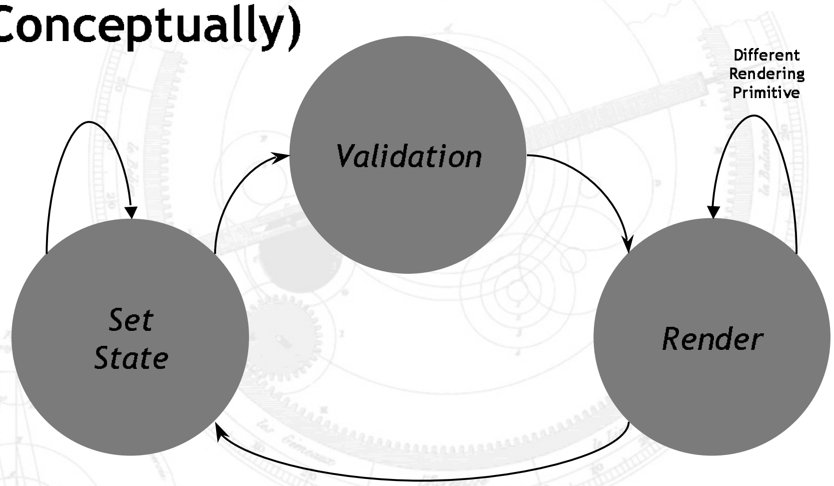
What Happens in a Validation

- Changing state may do more than just set values in the OpenGL context
  - May require reconfiguring the OpenGL pipeline
    - selecting a different rasterization routine
    - enabling the lighting machine
  - Internal caches may be recomputed
    - vertex / viewpoint independent data

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

41

The Way it Really Is (Conceptually)



OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

42

Why Be Concerned About Validations?

Validations can rob performance from an application

- "Redundant" state and primitive changes
- Validation is a two-step process
  - Determine what data needs to be updated
  - Select appropriate rendering routines based on enabled features

OpenGL

43

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

How Can Validations Be Minimized?

Be Lazy

- Change state as little as possible
- Try to group primitives by type
- Beware of "under the covers" state changes
  - **`GL_COLOR_MATERIAL`**
    - may force an update to the lighting cache ever call to **`glColor*()`**

OpenGL

44

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

How Can Validations Be Minimized? (cont.)

Beware of `glPushAttrib()` / `glPopAttrib()`

- Very convenient for writing libraries
- Saves lots of state when called
  - All elements of an *attribute groups* are copied for later
- Almost guaranteed to do a validation when calling `glPopAttrib()`

State Sorting

Simple technique … Big payoff

- Arrange rendering sequence to minimize state changes
- Group primitives based on their state attributes
- Organize rendering based on the expense of the operation

State Sorting (cont.)

Most Expensive

- Texture Download
- Modifying Lighting Parameters
- Matrix Operations
- Vertex Data

Least Expensive

A Comment on Encapsulation

An Extremely Handy Design Mechanism, however …

- Encapsulation may affect performance
  - Tendency to want to complete all operations for an object before continuing to next object
    - limits state sorting potential
    - may cause unnecessary validations

A Comment on Encapsulation (cont.)

- Using a "visitor" type pattern can reduce state changes and validations
  - Usually a two-pass operation
    - ① Traverse objects, building a list of rendering primitives by state and type
    - ② Render by processing lists
  - Popular method employed by many scene-graph packages

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

49

Case Study: Rendering A Cube

More than one way to render a cube

- Render 100000 cubes

Render six separate quads

Render two quads, and one quad-strip

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

50

Case Study: Method 1

Once for each cube …

```
glColor3fv( color );
for ( i = 0; i < NUM_CUBE_FACES; ++i ) {
  glBegin( GL_QUADS );
  glVertex3fv( cube[cubeFace[i][0]] );
  glVertex3fv( cube[cubeFace[i][1]] );
  glVertex3fv( cube[cubeFace[i][2]] );
  glVertex3fv( cube[cubeFace[i][3]] );
  glEnd();
}
```

OpenGL. 51 SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Case Study: Method 2

Once for each cube …

```
glColor3fv( color );
glBegin( GL_QUADS );
for ( i = 0; i < NUM_CUBE_FACES; ++i ) {
 glVertex3fv( cube[cubeFace[i][0]] );
 glVertex3fv( cube[cubeFace[i][1]] );
 glVertex3fv( cube[cubeFace[i][2]] );
 glVertex3fv( cube[cubeFace[i][3]] );
}
glEnd();
```

OpenGL. 52 SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Case Study: Method 3

```
glBegin( GL_QUADS );
for ( i = 0; i < numCubes; ++i ) {
  for ( i = 0; i < NUM_CUBE_FACES; ++i ) {
    glVertex3fv( cube[cubeFace[i][0]] );
    glVertex3fv( cube[cubeFace[i][1]] );
    glVertex3fv( cube[cubeFace[i][2]] );
    glVertex3fv( cube[cubeFace[i][3]] );
  }
}
glEnd();
```

OpenGL.

53

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Case Study: Method 4

Once for each cube …

```
glColor3fv( color );

glBegin( GL_QUADS );
glVertex3fv( cube[cubeFace[0][0]] );
glVertex3fv( cube[cubeFace[0][1]] );
glVertex3fv( cube[cubeFace[0][2]] );
glVertex3fv( cube[cubeFace[0][3]] );

glVertex3fv( cube[cubeFace[1][0]] );
glVertex3fv( cube[cubeFace[1][1]] );
glVertex3fv( cube[cubeFace[1][2]] );
glVertex3fv( cube[cubeFace[1][3]] );
glEnd();
```

```
glBegin( GL_QUAD_STRIP );
  for ( i = 2; i < NUM_CUBE_FACES; ++i ) {
    glVertex3fv( cube[cubeFace[i][0]] );
    glVertex3fv( cube[cubeFace[i][1]] );
  }
  glVertex3fv( cube[cubeFace[2][0]] );
  glVertex3fv( cube[cubeFace[2][1]] );
  glEnd();
```

OpenGL.

54

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Case Study: Method 5

```
glBegin( GL_QUADS );
for ( i = 0; i < numCubes; ++i ) {
  Cube& cube = cubes[i];
  glColor3fv( color[i] );

  glVertex3fv( cube[cubeFace[0][0]] );
  glVertex3fv( cube[cubeFace[0][1]] );
  glVertex3fv( cube[cubeFace[0][2]] );
  glVertex3fv( cube[cubeFace[0][3]] );

  glVertex3fv( cube[cubeFace[1][0]] );
  glVertex3fv( cube[cubeFace[1][1]] );
  glVertex3fv( cube[cubeFace[1][2]] );
  glVertex3fv( cube[cubeFace[1][3]] );
}
glEnd();
```

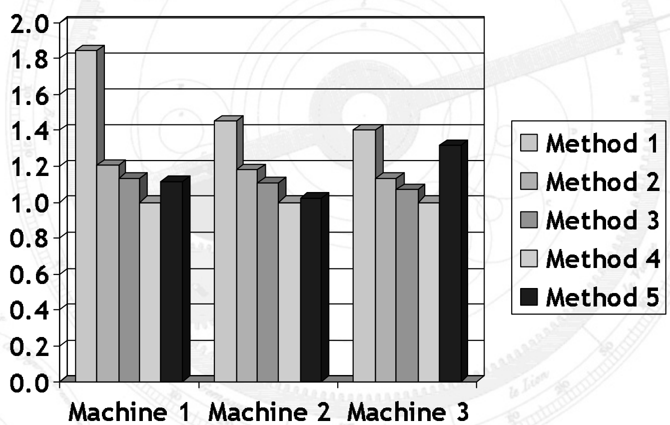
```
for ( i = 0; i < numCubes; ++i ) {
  Cube& cube = cubes[i];
  glColor3fv( color[i] );

  glBegin( GL_QUAD_STRIP );
  for ( i = 2; i < NUM_CUBE_FACES; ++i ) {
    glVertex3fv( cube[cubeFace[i][0]] );
    glVertex3fv( cube[cubeFace[i][1]] );
  }
  glVertex3fv( cube[cubeFace[2][0]] );
  glVertex3fv( cube[cubeFace[2][1]] );
  glEnd();
}
```

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

55

Case Study: Results



OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

56

Rendering Geometry

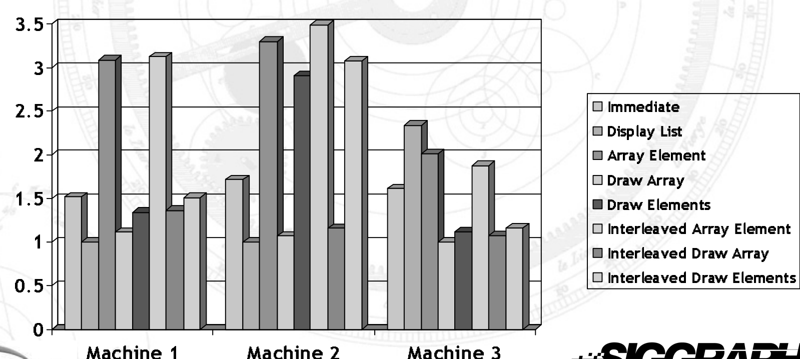
OpenGL has four ways to specify vertex-based geometry

- Immediate mode
- Display lists
- Vertex arrays
- Interleaved vertex arrays

57

Rendering Geometry (cont.)

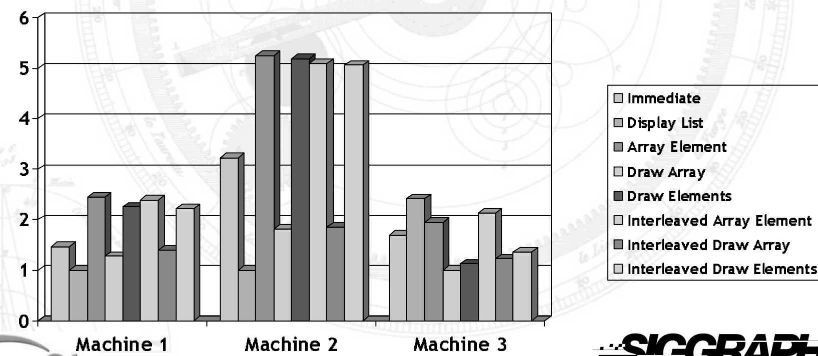
Not all ways are created equal



58

Rendering Geometry (cont.)

Add lighting and color material to the mix



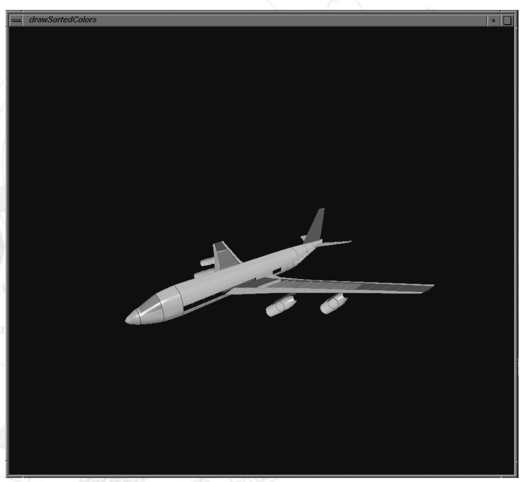
OpenGL.

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

59

Case Study: Application Description

- 1.02M Triangles
- 507K Vertices
- Vertex Arrays
  - Colors
  - Normals
  - Coordinates
- Color Material



OpenGL.

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

60

Case Study: What's the Problem?

Low frame rate

- On a machine capable of 13M polygons/second application was getting less than 1 frame/second

- Application wasn't fill limited

Case Study: The Rendering Loop

- Vertex Arrays

```
glVertexPointer( GL_VERTEX_POINTER );
glNormalPointer( GL_NORMAL_POINTER );
glColorPointer( GL_COLOR_POINTER );
```

- **`glDrawElements()`** – index based rendering
- Color Material

```
glColorMaterial( GL_FRONT,
                 GL_AMBIENT_AND_DIFFUSE );
```

Case Study: What To Notice

- Color Material changes two lighting material components per **glColor*()** call
- Not that many colors used in the model
  - 18 unique colors, to be exact
  - (3 * 1020472 – 18) = 3061398 "redundant" color calls per frame

Case Study: Conclusions

A little state sorting goes a long way

- Sort triangles based on color
- Rewriting the rendering loop slightly

```
for ( i = 0; i < numColors; ++i ) {
    glColor3fv( color[i] );
    glDrawElements( …, trisForColor[i] );
}
```

- Frame rate increased to six frames/second
  - 500% performance increase

Summary

Know the answer before you start

- Understand rendering requirements of your applications
  - Have a performance goal
- Utilize applicable benchmarks
  - Estimate what the hardware's capable of
- Organize rendering to minimize OpenGL validations and other work

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

65

Summary (cont.)

Pre-process data

- Convert images and textures into formats which don't require pixel conversions
- Pre-size textures
  - Simultaneously fit into texture memory
  - Mipmaps
- Determine what's the best format for sending data to the pipe

OpenGL. SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

66

Questions & Answers

Thanks for coming

- Updates to notes and slides will be available at

`http://www.shreiner.net/Performance.OpenGL`

- Feel free to email if you have questions

Dave Shreiner
shreiner@sgi.com

References

- *OpenGL Programming Guide*, 3rd Edition
  Woo, Mason et. al., Addison Wesley
- *OpenGL Reference Manual*, 3rd Edition
  OpenGL Architecture Review Board, Addison Wesley
- *OpenGL Specification*, Version 1.2.1
  OpenGL Architecture Review Board

For More Information

- SIGGRAPH 2001 Course # 12 - *Developing Efficient Graphics Software*
  - This afternoon @ 1:30 pm
- SIGGRAPH 2000 Course # 32 - *Advanced Graphics Programming Techniques Using OpenGL*

OpenGL

69

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Acknowledgements

A Big Thank You to …

- Peter Shaheen for a number of the benchmark programs
- David Shirley for Case Study application

OpenGL

70

SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES