Advances in OpenGL for Mac OS X Lion

Session 420

Matt Collins
GPU Software

These are confidential sessions—please refrain from streaming, blogging, or taking pictures

OpenGL

High-performance rendering API

- Direct access to vastly powerful GPUs
- Broadly used for 3D in games, visualization, and entertainment
- Foundation of visual technologies in Mac OS X

Introduction

You're wondering where we're at?

- What's new in Mac OS X Lion?
- The OpenGL core profile
- New rendering features
- New support features

OpenGL in Lion

- Streamlined OpenGL API
- More modern GPU features
- More closely aligned with the hardware
- Similar to OpenGL ES 2.0

Spoiler Alert!

Legacy and core

- New features require the Core Profile

Legacy Profile	Core Profile
Fixed Function or Programmable	Fully Programmable
Binary Compatibility	Efficient and modern
Default	Opt-in

New Core Profile Features

Rendering and more

- GLSL 1.50
- Uniform buffer objects
- Texture buffer objects
- Shader instancing
- SNorm textures
- Multisample textures
- Timer query

New Core Profile Features

Other rendering features

- Float and integer rendering
    - RG, multisample
- Texture arrays
- Conditional rendering
- Transform feedback

Other New Features

- IOSurface
  - Sharing between processes
- Automatic Graphics Switching
  - Run your app on integrated graphics

Using the Core Profile

OpenGL Core Profile

Concept

- New features and cleaner API
- Aligned to the hardware
- Cleaner expression of GPU power
- A similar jump as OpenGL ES 1.1 to 2.0
  - Very similar to OpenGL ES 2.0

OpenGL Core Profile

Why change?

- Lots of cruft in the API
  - Designed in 1992!
  - Abstracts CPU and GPU work
  - Want to express GPU capability only!
- The GPU is hungry for work
  - Need to keep it fed

OpenGL Core Profile

Development topics

- Getting started
- Differences from legacy profile
- Supplying data to the GPU
- Shader development
- API changes
  - Flushing
  - Syncing
- Tips

Getting Started

Requesting an OpenGL 3.2 Core Profile Context

```
NSOpenGLPixelFormatAttribute attr[] =
{
   NSOpenGLPFAOpenGLProfile,  NSOpenGLProfileVersion3_2Core,
   NSOpenGLColorSize, 24,
   NSOpenGLAlphaSize, 8,
   NSOpenGLPFAAccelerated,
   0
};

NSOpenGLPixelFormat* pix = [NSOpenGLPixelFormat initWithAttributes:attr];
NSOpenGLContext* ctx = [NSOpenGLContext initWithFormat:pix
                                           shareContext:nil];
```

Getting Started

Differences from Legacy Profile

- Programmable shaders vs. fixed function
  - Everything is a programmable shader
  - The GPU works this way!
- Batched draws vs. immediate mode
  - Hook up the hose
  - Much more efficient

Using the Core Profile

Supplying data to the GPU

- Vertex Buffers for everything
  - Send data to GPU once and reuse
- Supply shader via Generic Vertex Attributes
  - `glVertexAttribArray(…)`
- Vertex Array Objects (VAO)
  - Help you manage buffers and enables

Using the Core Profile

Supplying data to the GPU

```
#define ATTRIB_POS 0
//Create vertex array
glCreateVertexArray(1, &vao);
//bind
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
//replaces glVertex/Normal/ColorPointer
glVertexAttribPointer(ATTRIB_POS, 3, GL_FLOAT, GL_FALSE, 0, NULL);
//make sure you enable!
glEnableVertexAttribArray(ATTRIB_POS);
```

Using the Core Profile

Supplying data to the GPU

- 16 Vertex Attributes
  - Query limit to be sure
- Enables and pointers are saved
  - Bind once!
- Array Buffer is not saved
  - `GL_ARRAY_BUFFER` only affects `glMap`, `glBufferData`, etc.

Using the Core Profile

Supplying data to the GPU

- glUniformMatrix vs glMatrixMode
  - `glUniformMatrix4fv`
    - Count is matrices, not components or vectors
- Better programmer control
  - `glMultMatrix` was CPU-side!
  - Easier to associate transforms with objects
- A single matrix mul is faster than 10,000 (once per vtx)
  - Projection * Modelview

What Does a Shader Look Like?

Setting the (Vertex) stage

```
#version 150
in vec4 in_position;
in vec3 in_normal;
in vec3 in_color;
uniform mat4 ModelView;
uniform mat4 ModelViewProj;
out vec3 normal;
out vec3 color;
out vec4 pos;
void main() {
normal = normalize((ModelView*vec4(in_normal, 0.0)).xyz);
pos = ModelView*in_position;
color = in_color;
gl_Position = ModelViewProj * in_position;
}
```

What Does a Shader Look Like?

Vertex shader

```
#version 150
```

GLSL Version

```
in vec4 position;
in vec3 normal;
in vec3 color;
```

- Shader inputs
- Hooked up to Vertex Attributes
  - `glBindAttribLocation`
  - `glGetAttribLocation`
- `in` replaces `attribute`

```
out vec3 normal;
out vec3 color;
out vec4 pos;
```

- Shader Outputs
- `out` replaces `varying`

What Does a Shader Look Like?

Setting the (Fragment) stage

```
#version 150
in vec3 normal;
in vec3 color;
in vec4 pos;
uniform vec3 lightPos;
out vec4 finalColor;
void main()
{
  vec3 L = normalize(lightPos - pos.xyz);
  float attenuation = dot(L, normal);
  finalColor = attenuation*vec4(color, 1.0);
}
```

What Does a Shader Look Like?

Fragment shader

```
#version 150
```

GLSL Version

```
in vec3 normal;
in vec3 color;
in vec4 pos;
```

- Shader inputs
  - From Vertex Stage Output
- `in` replaces `varying`

```
out vec3 finalColor;
```

- Shader Outputs
  - `glBindFragDataLocation`
  - `glGetFragDataLocation`
- `out` replaces `gl_FragData`

Legacy to Core

API differences

Legacy Profile	Core Profile
glVertexPointer(...) glNormalPointer(...) glColorPointer(...)	glVertexAttribPointer(...)
glEnableClientState(...)	glEnableVertexAttribArray(...)
glBegin(...) … glEnd()	glDrawArrays(...) glDrawElements(...)
glGetString(GL_EXTENSIONS)	glGetStringi(GL_EXTENSIONS, <index>)

Legacy to Core

API differences

Legacy Profile	Core Profile
gl_ModelVIew	uniform mat4 ModelView (user defined)
gl_Normal	in vec3 normal (user defined)
Vertex Shader	
attribute	in
varying	out
Fragment Shader	
varying	in
gl_FragColor	out <variable>

Legacy to Core

Mapping and flushing

- `glMapBufferRange`
  - `GL_MAP_INVALIDATE_BUFFER`
  - `GL_MAP_INVALIDATE_RANGE`
    - Marks buffer/range as invalid
  - `GL_MAP_UNSYNCHRONIZED_BIT`
    - Does not sync on previous usage
  - `GL_MAP_FLUSH_EXPLICIT_BIT`
    - `glFlushMappedBufferRange`

Legacy to Core

En Garde! (fencing)

- Looking for synchronization primitives?
  - glFenceSync
    - Inserts a fence into command stream
  - `glClientWaitSync`
    - CPU waits for fence
  - `glWaitSync`
    - GPU waits for fence
    - Queue commands that depend on earlier operations
  - Flush appropriately!

Legacy to Core

Buffers and syncs

```
//insert a fence to test when the draw is finished
glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE,0);

//map a bound vbo
glMapBufferRange(...,
GL_MAP_FLUSH_EXPLICIT_BIT | GL_MAP_UNSYNCHRONIZED_BIT | GL_MAP_WRITE_BIT);

glClientWaitSync(bufferFlushed, GL_SYNC_FLUSH_COMMANDS_BIT, 0)
```

```
//update buffer
```

```
//flush the updated buffer to the gpu
glFlushMappedBufferRange(...);
//delete the fence
glDeleteSync(bufferFlushed);
```

Legacy to Core

Offscreen rendering

- FBO instead of PBuffer
  - More efficient
  - Much cleaner API
  - FBO error checking is more strict

ES 2.0 to Core Profile

API differences

ES 2.0	Core Profile
Vertex Shader	
attribute	in
varying	out
Fragment Shader	
varying	in
gl_FragColor	out <variable>

ES 2.0 to Core Profile

Cocoa differences

ES 2.0	Core Profile
EAGLLayer	NSOpenGLView
EAGLContext	NSOpenGLContext
presentRenderbuffer	Draw to FBO 0 + flushBuffer

- `NSOpenGLView` is a standalone object
  - Don’t need to grab the GL Layer
- Must use `[NSOpenGLContext flushBuffer]`

Core Profile

Summary

- Use the Core Profile!
- Modern features
- Better API
  - VAO to group buffers
  - Generic Vertex Attributes to supply data
  - GLSL 1.5 for your shaders

Core Profile

Tips

- Querying extensions
  - `GLubyte* glGetStringi(GL_EXTENSIONS, <GLuint index>)`
  - Indexed from 0 to GL_NUM_EXTENSIONS
  - Individual extension strings vs. one long one

```
glGetIntegerv(GL_NUM_EXTENSIONS, &numExts);
for(i = 0; i < numExts; i++) {
  const GLubyte *extString = glGetStringi(GL_EXTENSIONS, i);
}
```

Core Profile

Tips

- Don't forget to include `gl3.h`!
- Calling deprecated API will error
  - `GL_INVALID_OPERATION`
- There is no VAO 0
  - Must create your own
- Must have drawable attached to FBO 0

New Rendering Features

Uniform Buffer Objects

Mass data upload

- Allows app to upload and store uniform data
- Uses Buffer Objects for storage
- Can be faster than calls to `glUniform`

Uniform Buffer Objects

Shader layout

```
layout(std140) uniform UBO
{
     mat4x4 MV[INSTANCES_PER_BLOCK];
} block;

in vec4 inPos;

void main()
{
  mat4x4 MV = block.MV[0];
  mat4x4 MVP = P*MV;
  gl_Position = MVP * vec4(inPos, 1.0);
}
```

Uniform Buffer Objects

Hooking it all up

```
//picked by us
#define BLOCKBINDING 0

GLuint UBOBlockIndex;
//Get the index, similar to a uniform location
UBOBlockIndex = glGetUniformBlockIndex(prg, "UBO");
//Set this index to a binding index
glUniformBlockBinding(prg, UBOBlockIndex, BLOCKBINDING);
//bind our buffer to our chosen binding index
glBindBufferRange(GL_UNIFORM_BUFFER,BLOCKBINDING,uboID,offset,size);
```

Uniform Buffer Objects

Hooking it all up

- Get UBO block index (think attribute location)

```
glGetUniformBlockIndex(prg, "UBO")
```

- Set block binding index

```
glUniformBlockBinding(prg, UBOBlockIndex, BLOCKBINDING)
```

- Bind buffer object

```
glBindBufferRange(GL_UNIFORM_BUFFER, BLOCKBINDING, buffer, ...)
glBindBufferBase(GL_UNIFORM_BUFFER, BLOCKBINDING, buffer)
```

Uniform Buffer Objects
Recap

- Remember
  - Check size limits!
  - Can’t modify UBO that’s being used to draw
  - Orphan buffers
    - `glBufferData(GL_UNIFORM_BUFFER,…,NULL)`
    - Or double-buffer
  - Split frequently updated uniforms into separate UBO
  - Don’t update more than you have to!

Texture Buffer Objects

Texture-backed storage

- New texture target
  `GL_TEXTURE_BUFFER`
- New sampler type
  `samplerBuffer`
- Takes advantage of texture caching
- Fast uploads
- A big 1D texture!
- Check texture size limits!

Texture Buffer Objects

TBO creation and setup

```
//create texture as normal
glGenTextures(1, &tex);
glBindTexture(GL_TEXTURE_BUFFER, tex);
//TBOs bind previously existing buffer objects
//as textures
glTexBuffer(GL_TEXTURE_BUFFER, GL_RGBA32F, vbo);
```

Texture Buffer Objects

Shader layout

```
//declare sampler type as TBO
uniform samplerBuffer TBO;
in vec3 inPos;\n"
void main() {
  //fetch data via texelFetch
  mat4x4 MV = mat4x4(texelFetch(TBO,0),
     texelFetch(TBO, 1),
     texelFetch(TBO, 2),
     texelFetch(TBO, 3));
  mat4x4 MVP = P*MV;\n"
  gl_Position = MVP * vec4(inPos, 1.0);
}
```

Instancing

Shader Instancing

- Last year was Divisor Instancing
- This year is Shader Instancing
- New built-in `instanceID`
- Can index uniform or texture data
  - Skinning matrix array

Instancing

Shader Instancing

```
#version 150
uniform mat4x4 P;
uniform vec3 lightPos;
uniform vec4 amb, dif;
layout(std140) uniform UBO {
  mat4x4 MV[20000];
} block;
in vec3 inPos;
in vec3 inNrm;
out vec3 color;
```

Instancing

```
void main() {
  mat4x4 MV = block.MV[gl_InstanceID];
  mat4x4 MVP = P*MV;
  gl_Position = MVP * vec4(inPos, 1.0);
  vec3 nrm = normalize(mat3x3(MV) * inNrm);
  float lit = max(dot(nrm, lightPos), 0.0);
  color = amb.rgb + dif.rgb * lit + spc;
}
```

Multisample Textures

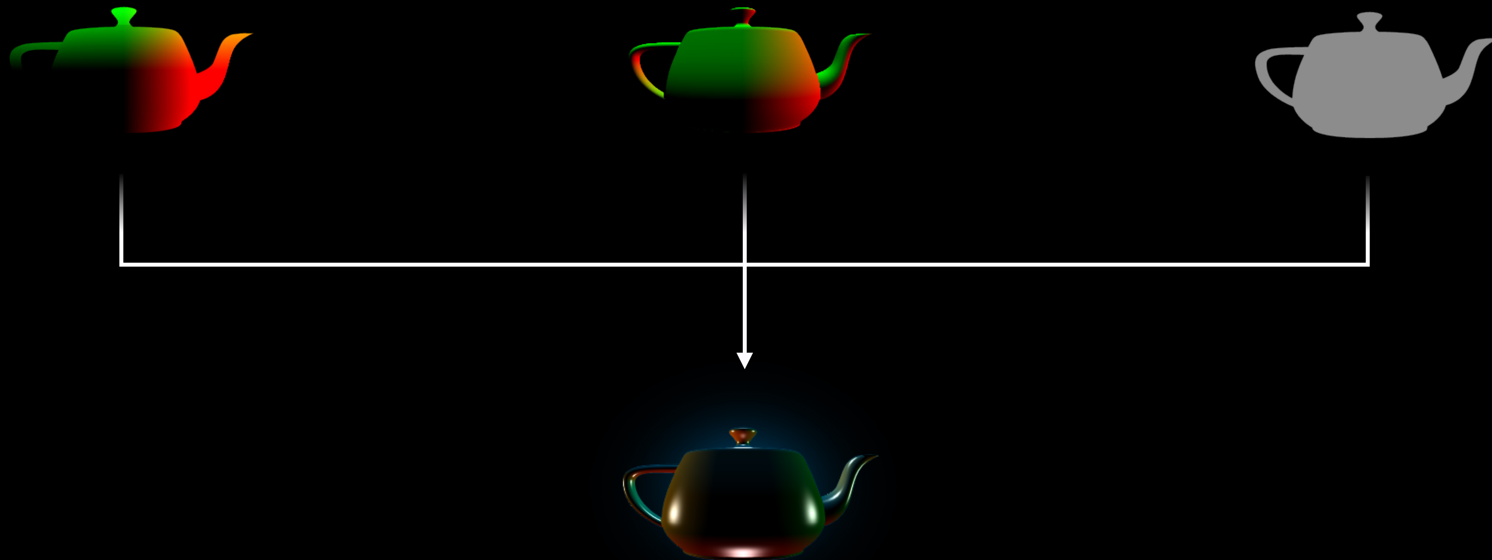
Multisample Rendertargets

- New texture targets
  - GL_TEXTURE2D_MULTISAMPLE
  - GL_TEXTURE2D_MULTISAMPLE_ARRAY
- Creating a multisample texture

```
GLboolean fixedSampleLocation = GL_TRUE;
GLsizei sampleCount = 4;
glBindTexture(GL_TEXTURE2D_MULTISAMPLE, textureID);
glTexImage2DMultisample(GL_TEXTURE2D_MULTISAMPLE, sampleCount,
                        GL_RGBA8, 800, 600, fixedSampleLocation);
```

Multisample Textures

Multisample Gbuffer



Multisample Textures

Resolving multisample textures

```
#version 150
sampler2DMS positionTexture, normalTexture;
uniform int sampleCount;
/* other uniform and input declarations go here */
void main() {
  //texelFetch uses integer texture coordinates
  ivec2 t = ivec2(in_texcoord * textureSize(positionTexture));
  for(int currentSample = 0; currentSample < sampleCount; currentSample++){
     vec3 position = texelFetch(positionTexture, t, currentSample);
     vec3 normal = texelFetch(normalTexture, t, currentSample);
     /* accumulate calculated color as usual, etc here */
  }
}
```

Tips
Legacy

- `texture2DLod` is an extension in Legacy
  - Use pragma require in shader
- TBO and PBO are separate things
  - Texture Buffer Object
    - Texture-backed storage
  - Pixel Buffer Object
    - Asynchronous texture upload

Other New Features

Automatic Graphics Switching

Save that battery!

- What is automatic switching?
- Pixel Format must track multiple GPUs
  - CGLFPAAllowOfflineRenderer
- Supporting Integrated GPUs
  - Add Info.plist attribute

```
NSSupportsAutomaticGraphicsSwitching = YES
```

- See session 310 (OpenGL Techniques for SnowLeopard) from WWDC 2009

Test on actual hardware!

Other Cool Stuff

OpenGL Profiler and more

- OpenCL
  - Check out the OpenCL talks
- IOSurface
  - Share surfaces between contexts and apps
- OpenGL Profiler
  - Now with Remote Profiling!

Fullscreen Modes

The Full Monty

- Want fullscreen?
  - Create a covering window
  - All the benefits, none of the hassle
- Absolutely must switch modes?
  - See QA on `CGLSetFullscreenWithOptions`
- `CGDisplayBaseAddress() returns NULL`
  - Don’t use it!
- Interested in video capture?
  - See the AV Foundation talks

OpenGL on Lion

Wrap-up

- Lots of new features in Lion
- Take advantage of the Core Profile
  - Create Core Profile context
  - Use VAO
  - Use Generic Vertex Attributes
  - Use GLSL 1.5
- Try out the new features

Demo

More Information

Allan Shaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Documentation
OpenGL Programming Guide
https://developer.apple.com/library/prerelease/mac/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/

More Information (Cont.)

Technical Q&As
How to capture screen activity to a movie
http://developer.apple.com/library/prerelease/mac/#qa/qa1740/

How to take an image snapshot
http://developer.apple.com/library/prerelease/mac/#qa/qa1741/

Using the integrated GPU
http://developer.apple.com/library/prerelease/mac/#qa/qa1734/

Technical Note
Supporting multiple GPUs
http://developer.apple.com/library/mac/#technotes/tn2229/

Related Sessions

Best Practices for Open GL ES Apps in iOS	Mission Wednesday 4:30PM
Introducing AV Foundation Capture for Lion	Pacific Heights Wednesday 3:15PM

Labs

OpenGL for Mac OS X Lab	Graphics, Media & Games Lab B Thursday 2:00PM

WWDC2011