OpenGL with Qt 5

Qt Developer Days, Berlin 2012

Presented by Sean Harmer

Produced by Klarälvdalens Datakonsult AB

Material based on Qt 5.0, created on November 9, 2012

- Overview of OpenGL Support in Qt 5
- QtQuick 2 and OpenGL
- The Future

- Overview of OpenGL Support in Qt 5
- QtQuick 2 and OpenGL
- The Future

What is OpenGL?

- Family of APIs for 2D and 3D rendering
- Very flexible
- Massive computational power
- Industry standard
- Cross-platform
- Actively developed – http://www.khronos.org/
- Extensible

What is OpenGL used/good for?

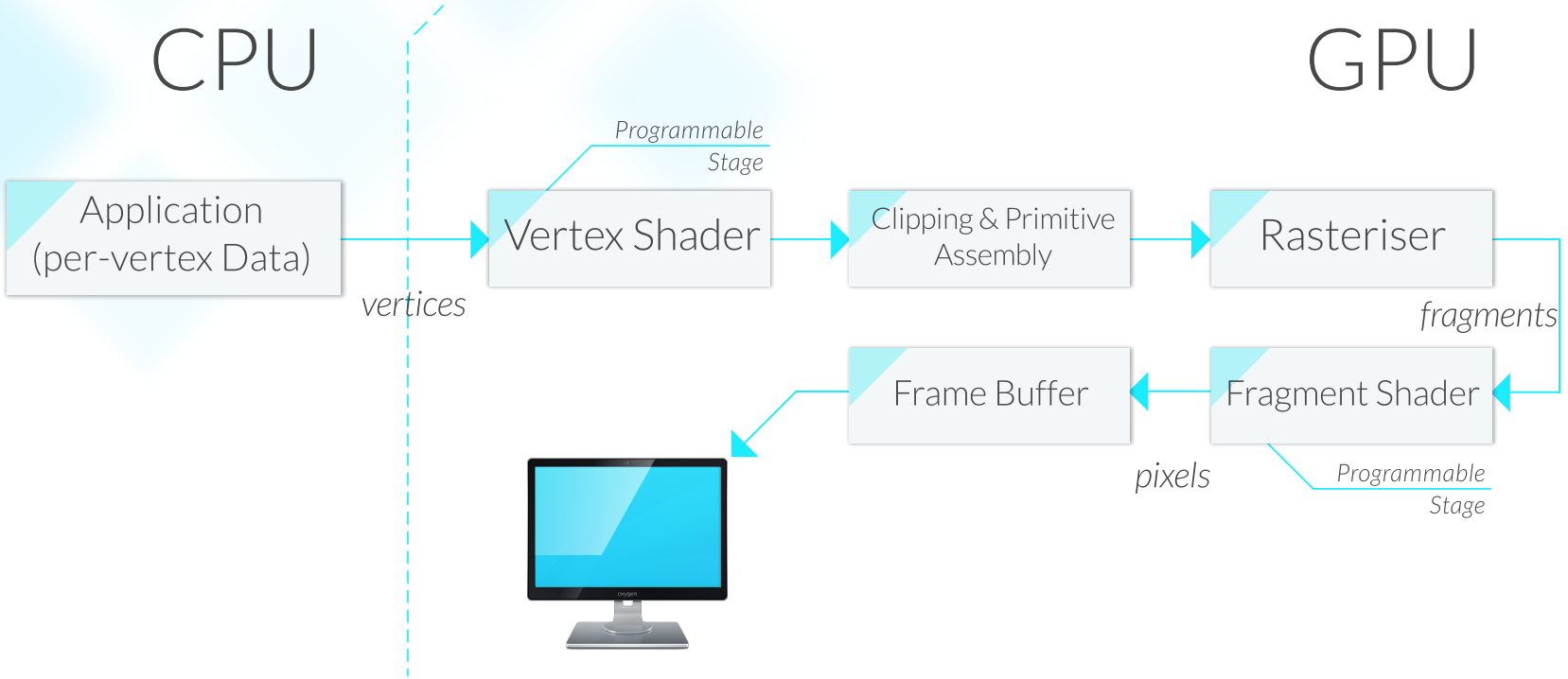
- Computer Aided Design
- Content creation
- Data visualisation
- Games
- Image processing
- Simulation

Hello Triangle Example

- Application setup
- Window creation
- Context creation
- Build an OpenGL pipeline
- Prepare geometry
- Feed data

Demo opengl/shader-fundamentals/ex_basic_usage

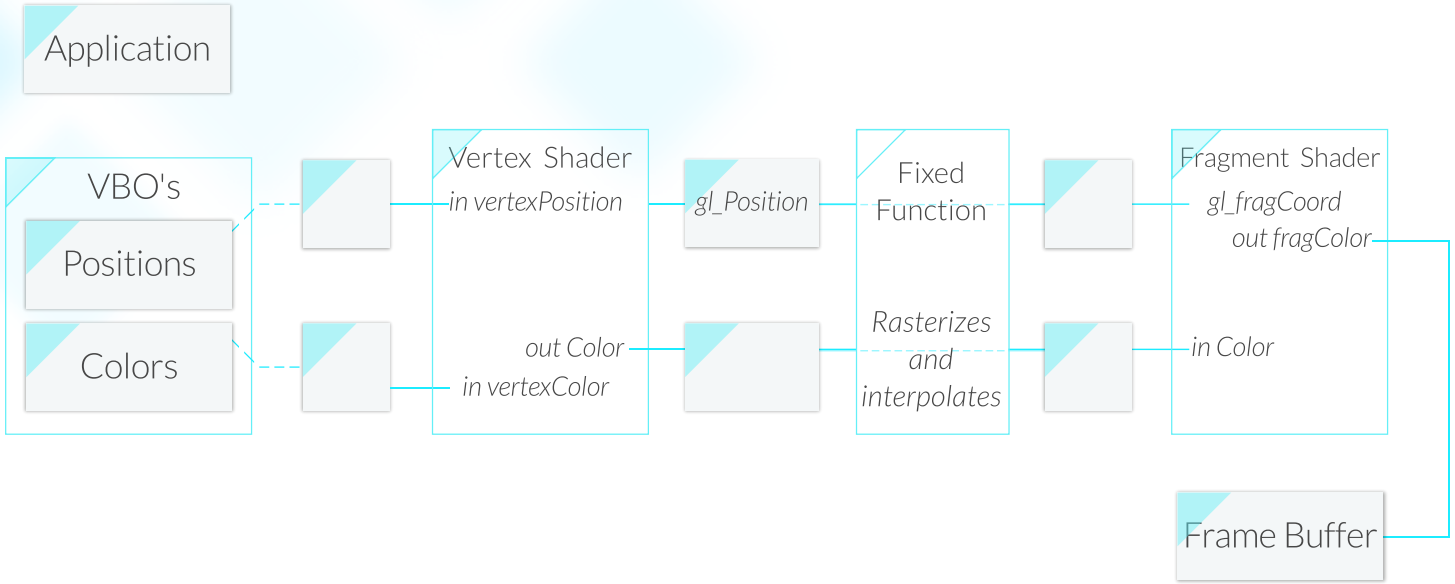
Simplified OpenGL Pipeline



Application Tasks

- Create window and OpenGL context
- Create per-vertex data
- Configure OpenGL state
  - Create shader programs
  - Create Vertex Buffer Objects (VBOs)
  - ...
  - Create the pipeline and configure it
- Kickoff the drawing!
- Update AI, physics, application state, make coffee...

Shader Interfaces cont'd.



Varying Degrees of Change

- Shaders can have constant variables:

```
const float pi = 3.14159;
const vec2 resolution = vec2( 1024.0, 768.0 );
```

- VBOs can hold per-vertex attributes:

```
m_colors.create();
m_colors.setUsagePattern( QOpenGLBuffer::StaticDraw );
m_colors.bind();
m_colors.allocate( colorData, 3 * 3 * sizeof( float ) );
```

What about in between these extremes?

Demo opengl/shader-fundamentals/ex_shader_uniform

`QOpenGL*` vs `QGL*`

Rule of thumb: Prefer `QOpenGL*`

OpenGL with Qt 5 Summary

- OpenGL windows and contexts
- Pipeline is programmable via shaders
- Shaders are compiled and linked
- Use Vertex Buffer Objects (VBOs) for speed
- Enable VBOs to match Vertex Shader interface
- Draw with `glDrawArrays()` or `glDrawElements()`
- Shader programs are configurable via `uniform` variables

- Overview of OpenGL Support in Qt 5
- QtQuick 2 and OpenGL
- The Future

- QtQuick 2 built on top of OpenGL renderer
- QDeclarativeView (QQ1) → QQuickView (QQ2)
- Much in common...
- Can we integrate it?
- Several approaches
  - OpenGL underlay
  - OpenGL overlay
  - Framebuffer Objects
  - Custom QtQuick2 Items
  - `ShaderEffect` item

Demo opengl/qtquick2-integration/ex_basic_integration

Demo opengl/qtquick2-integration/ex_updates

QtQuick 2/OpenGL Integration

- Don't clear!
- Receive notification
- Borrow the QtQuick2 OpenGL context
- Set state and draw stuff
- Reset state
- Return context in condition we found it
  - No Vertex Buffer Objects bound
  - No Index Buffer bound
  - No Shader Program bound
  - Restore texture unit configuration
  - Don't forget sampler objects!

Using a Custom Context

- QtQuick 2 built on OpenGL 2.x (or OpenGL ES 2)
- Must be compatible with QtQuick 2's needs
- Compatibility Profile
- `QSurfaceFormat` and `QWindow::setFormat()`
- Same rules apply
- Now we can take advantage of new features!
- Export objects/types to QML

Demo opengl/qtquick2-integration/ex_custom_context

Demo opengl/qtquick2-integration/ex_qq2ui

Demo opengl/qtquick2-integration/ex_qq2ui_uniforms

- Overview of OpenGL Support in Qt 5
- QtQuick 2 and OpenGL
- The Future

Qt 5.1 and Beyond...

- Version functions and extensions
- Vertex Array Objects (VAOs)
- Transform feedback objects
- Texture/Sampler objects
- Geometry, Tessellation, and Compute Shader support
- Atomic counters, fences
- Query objects - timing, occlusion
- Debugging/profiling support
- ...