Intro to Computer Graphics: Just enough OpenGL

Updated: September 13, 2019
Slides By: Philmo Gu
Graphics Processing Unit (GPU)

• GPU VS CPU
  • CPU: Running instructions over a few cores with lots of cache memory
  • GPU: Sharing instructions over many simple cores
• Can accelerate some software by 100x

“GPUs are optimized for taking huge batches of data and performing the same operation over and over very quickly, unlike PC microprocessors, which tend to skip all over the place."
– Nathan Brookwood, Insight 64 principal analyst

Graphics API

• Graphics API is a specification that describes the behaviour of rasterization-based rendering system
  • Implemented by hardware manufacturer
  • Standard API allows applications to run on different kinds of graphical hardware

• Drivers translate OpenGL API commands into GPU commands
  • Defect in graphics is often due to the hardware, not the library

• Examples:
  • OpenGL, Vulkan (low-level), CUDA (Nvidia, low-level), DirectX (Microsoft)

Source: https://developer.samsung.com/tech-insights/vulkan/what-is-a-graphics-api
Source: https://www.khronos.org/opengl/wiki/FAQ
OpenGL – “Open Graphics Library”

• Rendering Pipeline: Sequence of steps taken to render objects

• Immediate mode vs Core-profile
  • Immediate mode: Pipeline is fixed; developers are limited to existing functions;
    • inefficient, but easy to learn
  • Core-profile: Pipeline is modular; development is flexible by editing parts of pipeline
    • more flexible and efficient, but more difficult to learn

Source: https://learnopengl.com/Getting-started/OpenGL
Source: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview
OpenGL – Open Graphics Library

• OpenGL is a state machine
  • *State*: information used by the rendering system (e.g. vertex position, colour)
  • *Objects*: container for OpenGL’s states (e.g. vertex array object)
  • *OpenGL Context*: object that holds all of OpenGL

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_BUFFER</td>
<td>Buffer Object</td>
</tr>
<tr>
<td>GL_SHADER</td>
<td>Shader object</td>
</tr>
<tr>
<td>GL_PROGRAM</td>
<td>Program Object</td>
</tr>
<tr>
<td>GL_VERTEX_ARRAY</td>
<td>Vertex Array Object</td>
</tr>
<tr>
<td>GL_QUERY</td>
<td>Query Object</td>
</tr>
<tr>
<td>GL_PROGRAM_PIPELINE</td>
<td>Program Pipeline Object</td>
</tr>
<tr>
<td>GL_TRANSFORM_FEEDBACK</td>
<td>Transform Feedback Object</td>
</tr>
<tr>
<td>GL_SAMPLER</td>
<td>Sampler Object</td>
</tr>
<tr>
<td>GL_TEXTURE</td>
<td>Texture Object</td>
</tr>
<tr>
<td>GL_RENDERBUFFER</td>
<td>Renderbuffer Object</td>
</tr>
<tr>
<td>GL_FRAMEBUFFER</td>
<td>Framebuffer Object</td>
</tr>
</tbody>
</table>

Source: https://www.khronos.org/opengl/wiki/OpenGL_Object
Source: https://www.khronos.org/opengl/wiki/OpenGL_Context
OpenGL Loading Library

• Library that loads pointers to OpenGL functions
  • required to access functions from OpenGL versions above 1.1
  • abstracts away the difference between the loading mechanisms on different platforms

• We’ll be using “glad” (Multi-Language GL/GLES/EGL/GLX/WGL Loader-Generator)

Source: https://www.khronos.org/opengl/wiki/OpenGL_Loading_Library
GLFW – “Graphics Library FrameWork”

• Utility library to create and manage windows, OpenGL context, and input controls (e.g. mouse & keyboard input)

Source: https://www.glfw.org/
Putting it together

GPU → Driver → OpenGL → OpenGL Loading Library → Your Program → Interface (e.g. GLFW)

Rendering

Events from input controls