High Performance Graphics and Text Rendering on the GPU

Barbara Geller & Ansel Sermersheim

MeetingC++ November 2019

Introduction

- Prologue
- What are Graphics
- Why Graphics Matter to (Almost) Everyone
- Using Graphics in a C++ Application
- Show me the Demo
- Graphics Terminology
- Program/Data Flow
- Rendering Text on the GPU
- Useful Links
- Roadmap

Who is CopperSpice

- Maintainers and Co-Founders
  - CopperSpice
    - cross platform C++ libraries (linux, os x, windows)
  - CsString
    - support for UTF-8 and UTF-16, extensible to other encodings
  - CsSignal
    - thread aware signal / slot library
  - libGuarded
    - library for managing access to data shared between threads
  - DoxyPress
    - documentation generator for C++ and various other languages

Who is CopperSpice

- Credentials
  - every library and application is open source
  - developed using cutting edge C++ technology
  - source code hosted on github
  - prebuilt binaries available on our download site
  - all documentation is generated by DoxyPress

  - youtube channel with over 40 videos
  - speakers at multiple CppCon and CppNow conferences
  - speakers at emBO++ 2019
  - numerous presentations in the US and Europe

Graphics

- What are Graphics
  - graphics are responsible for displaying images and text in a way which is effective and meaningful to the consumer
  - classified into distinct categories
    - raster graphics
      - bitmap image rendered by addressing discrete pixels
      - typically processed on the CPU
      - resolution dependent, does not scale without loss of quality
    - vector graphics
      - an image defined by vertex coordinates
      - drawing commands are based on complicated mathematics
      - should be processed on the GPU
      - scaled to a larger size the image quality is not compromised

Graphics

- For More Information . . .
  - timeline of graphics changes from 1989 through 2018
  - what is driving the graphics industry, changes in GPU design, overview of API specifications, introduction to Vulkan

  - Evolution of Graphics Technology
    - https://www.youtube.com/watch?v=u5SNd9sKn94

  - GPU, Pipeline, and the Vector Graphics API
    - https://www.youtube.com/watch?v=CrKxMrLczis

  - Rendering 3D Graphics
    - https://www.youtube.com/watch?v=MXz2t0gvRxI

Graphics

- Image Quality
  - VGA
    - 307,000 pixels
    - 640 x 480
  - 1080p, Full HD
    - 2.07 million pixels
    - 1920 x 1080
  - 4k, UHD
    - 8.2 million pixels
    - 3840 x 2160
  - 8k, 8k UHD
    - 33.1 million pixels
    - 7680 x 4320

Graphics

- Processors
  - CPU
    - average around 4-8 cores
    - high end around 10-20 cores
    - typically 1 or 2 threads run on each core
  - GPU
    - average around 1000 cores
    - high end around 3000 cores
    - many are hyperthreaded which means there are typically 5-60 threads running on each core

Graphics

- Using Graphics in a C++ Application
  - gaming industry has the largest influence over changes in graphics
  - new GPU designs require new programming tools
  - graphics are more than just explosions, robots, and lightsabers
  - desktop programs are transitioning to better quality graphics

  - graphics for the GUI developer
    - high dpi support for free
    - responsive user interface
    - smooth scrolling of text
    - clean colors and edges
    - scalable charts, graphs, animations

Graphics

- Demo Dependencies
  - C++17
  - CMake version 3.8 or newer
  - SDL2 library for window management
    - handles user input like keyboard and mouse events
    - open source answer to DirectX
    - cross platform
    - lots of games leverage this library
    - similar to GLUT, GLEW, or GLFW
  - <insert demo here>

Graphics

- Structure of our Demo Program
  - demo.cpp
    - setup, main event loop (60 lines)
  - demo.h
    - two structures, function declarations (60 lines)
  - data.cpp
    - transform calculations (200 lines)
  - resources
    - mesh files, font image, shaders
  - links with SDL2 and one other library

Graphics

- What is behind the Demo
  - CsPaint library
    - open source BSD license
    - encapsulates the Vulkan API exposing a higher level API
    - works on any GPU which supports Vulkan 1.1
    - uses vulkan.hpp which is the C++ interface
    - approximately 40 source files

    - GLM
      - used for matrix math / linear algebra
      - provides buffer containers like the class glm::vec3
      - bundled with CsPaint
      - MIT license

Graphics

- CsPaint
  - platform independent
    - windows and linux have native drivers
    - android native drivers available since version 7
    - MoltenVK is a Vulkan wrapper on OS X and iOS
  - using the CsPaint library allows you to render graphics without having to focus on the tedious and repetitive sections
  - instead you are free to concentrate on writing your shaders and modeling 3D images or simply displaying text

Graphics

- Why Vulkan
  - OpenGL is unable to send multiple commands at once
    - GPU sits idle too much of the time
    - CPU spends too much time waiting for the GPU

  - Khronos Group took OpenGL and brought it closer to the metal
  - burden is now on the developer, fewer defaults supplied
  - there is no built in memory management, must be implemented and controlled by the developer
  - fixed function pipeline does not exist, developers must supply their own shader implementations
  - same API for Linux, Windows, and Android
  - well supported translation layer for OS X and iOS

Graphics

- Demo Source Code (set up, demo.cpp)

```
auto window              = init_window();
auto [context, surface] = init_vulkan(window);

auto device              = surface->graphicsDevice();
auto vertexShader        = device->createShader(vertexShaderData);
auto fragmentShader      = device->createShader(fragmentShaderData);
auto textFragmentShader = device->createShader(textFragmentShaderData);

auto renderPass          = device->createRenderPass();
auto commandpool         = device->graphicsCommandPool();

init_render(device, surface, commandpool, vertexShader, fragmentShader,
            textFragmentShader, renderPass);
```

- Demo Source Code (event loop, demo.cpp)

```
while (run) {
  SDL_Event event;

  while (SDL_PollEvent(&event)) {
    if (event.type == SDL_QUIT) {
      run = 0;

    } else if (event.type == SDL_KEYDOWN) {
      // process key events like arrow keys to rotate the copper pot

    }
  }

  draw_frame(device, surface, renderPass,
       transform_matrix(glm::vec3(x_rotation, y_rotation, 0.0)), zoom_factor);
}
```

Terminology

- Vulkan Graphics Terminology
  - instance, context, surface
  - memory heap
  - vertex buffer, uniform buffer
  - shaders (vertex, tessellation, geometry, fragment)
  - pipeline
  - command buffer
  - textures
  - frame buffer, depth buffer, render pass
  - swapchain
  - queues (graphics, present, transfer, compute)
  - fences, semaphores
  - face culling, winding
  - push constants, scissor, extent, viewport

Terminology

- Graphics Definitions
  - instance
    - connection between your application and the Vulkan library
    - your application will usually only require a single instance
    - Vulkan calls typically need to receive the instance
  - context
    - not used in Vulkan, an OpenGL context is similar to an instance
  - surface
    - term for the window region where images are rendered
    - native window support is not handled directly in the API, implemented in platform extensions

Terminology

- Graphics Definitions
  - memory heap
    - provides storage for buffers which will be accessed by the GPU
    - every buffer must be created and managed by the user
    - memory is allocated from a memory heap
    - the buffer is then bound to the allocated memory
    - any memory which is visible to the GPU will be in the list of all memory heaps
    - there are usually multiple heaps available
    - certain heaps can only be used for specific types of buffers

Terminology

- Buffers
  - vertex buffer
    - coordinates of all vertices which describes the geometry for the image being rendered
    - triangles are the most widely used shape
    - there can be as few as three vertices, typically thousands for a given image
  - uniform buffer
    - contains data which is applied to every vertex
    - for example, transformations can be used to shift the geometry and render the image somewhere other than the center

Graphics

- What is a Shader
  - a program written in a specialized language, designed to be run on the GPU

  - OpenGL shaders are written in GLSL (OpenGL Shading Language)
  - Direct3D shaders are written in HLSL
  - Vulkan shaders are typically written in GLSL or HLSL and must be compiled to the SPIR-V binary format

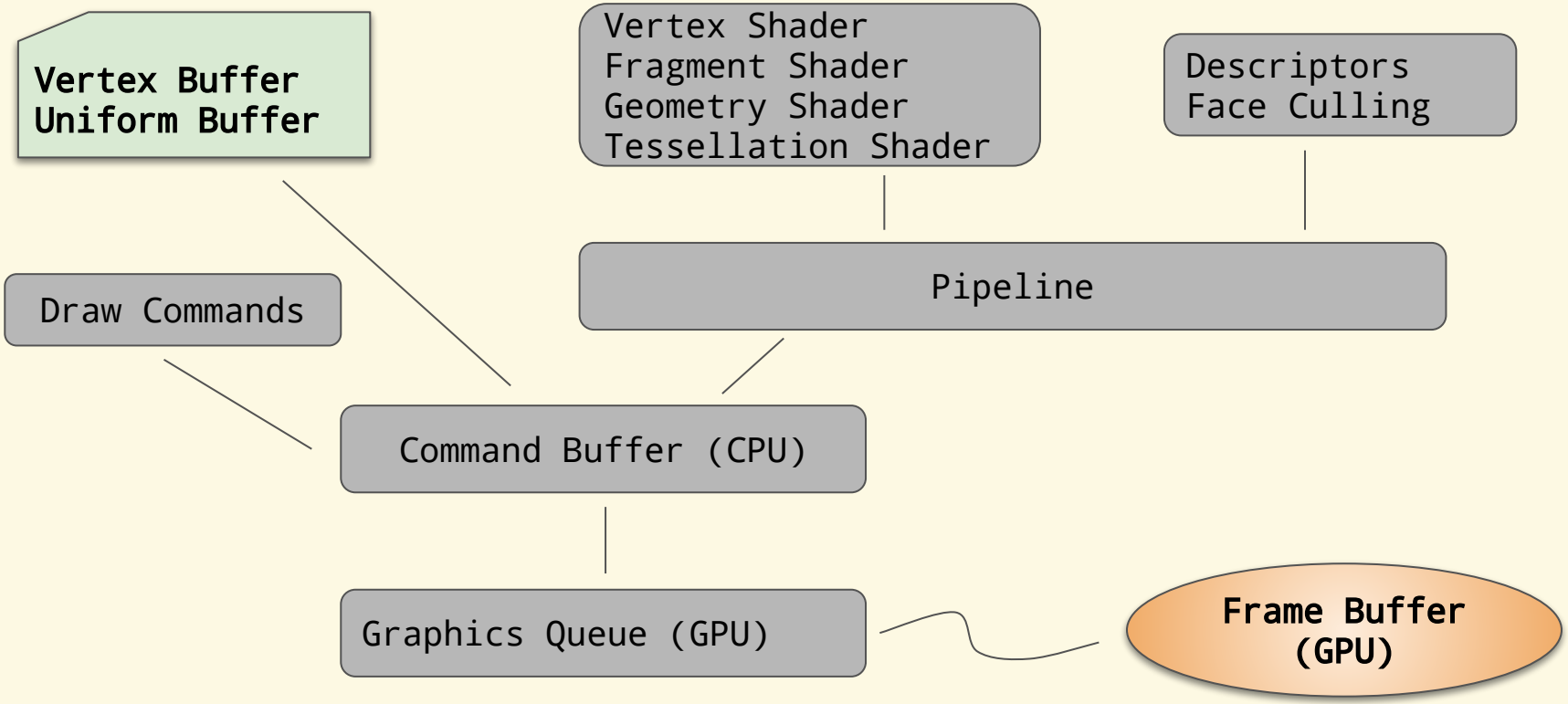
  - various categories of shaders
  - each shader is responsible for a different aspect of rendering

Terminology

- Graphics Definitions
  - winding direction
    - must be specified as part of the pipeline
    - conventionally clockwise winding is used
    - the direction is used to determine whether a given triangle is facing the camera (front facing) or facing away (back facing)

  - back face culling
    - is the process of discarding triangles which are back facing since these are not normally visible to the viewer
    - there is no default so it must be set on or off in the pipeline
    - triangles are "culled" early in the rendering pipeline
    - increases efficiency by reducing the number of fragments which are processed

Program / Data Flow



When the frame buffer is ready, a previously configured semaphore is triggered which presents the output

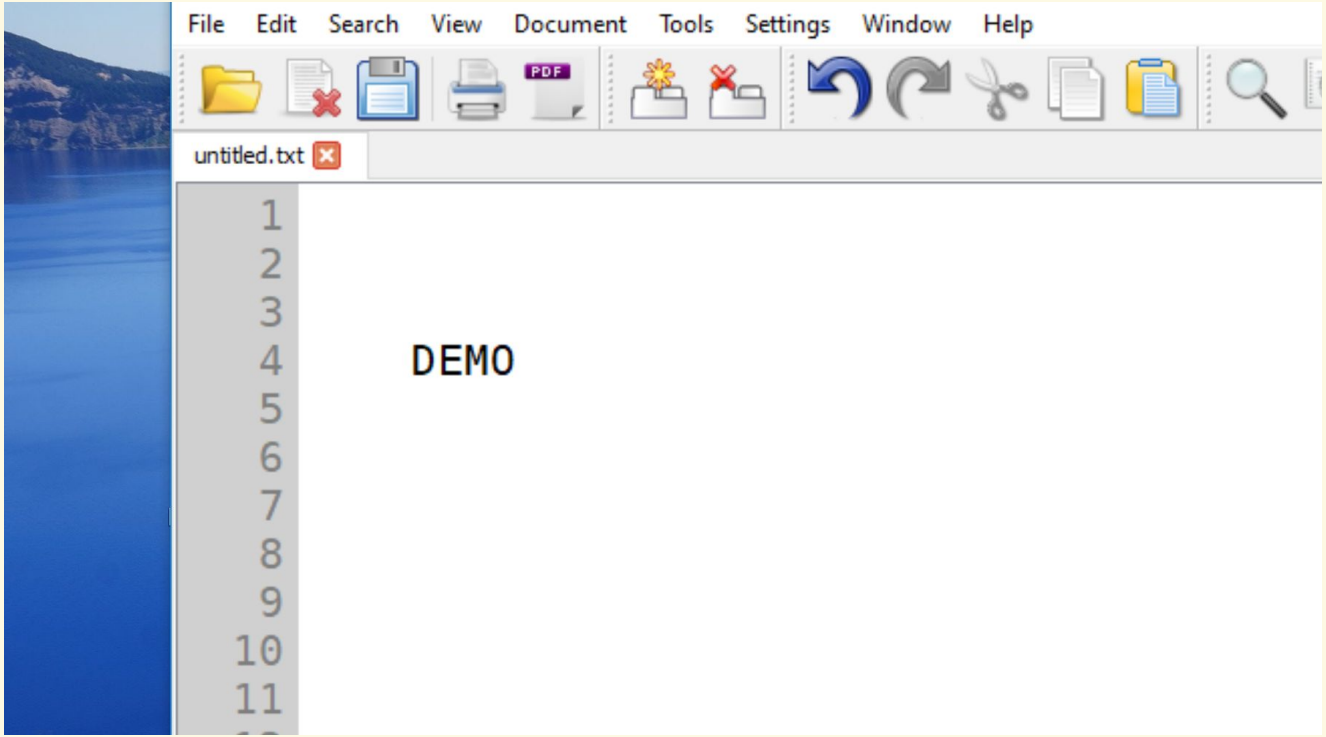
Rendering Text on the GPU

- Putting the Graphics back in GUI
  - intent of a GUI is to present a graphical interface
  - graphics have traditionally been rendered on the CPU
  - most GUI libraries still use software rendering

  - why not use the GPU for all types of graphics
  - text is graphics too

  - CsPaint can render text as graphics on the GPU
  - more efficient
  - infinitely scalable
  - cleaner edges
  - textured, reflective, shadowing . . .

CPU Text vs GPU Rendering



This text is a 16 point monospace font.
Generated screenshot was taken from a Windows GUI application.

CPU Text vs GPU Rendering

Increased the previous image size by 1400 percent.

CPU Text vs GPU Rendering

In the “Graphics Demo” moved the camera closer to the text.
Generated a screenshot with no alterations.

Rendering Text on the GPU

- Why is this Comparison Valid
  - scaling up text on the CPU does not require a significant amount of memory, however the quality is blurry and unacceptable
  - most programs scale up text by redrawing the font using a higher DPI, this can require a great deal of additional memory
  - rendering text on the GPU at a higher DPI does not require using a larger font and the quality of the text remains crisp
  - font scaling is done by calculations in the fragment shader

Rendering Text on the GPU

- Steps to Render Text
  - walk the string and create a “rectangle” for each letter
  - uses a process similar to applying a standard texture to a model

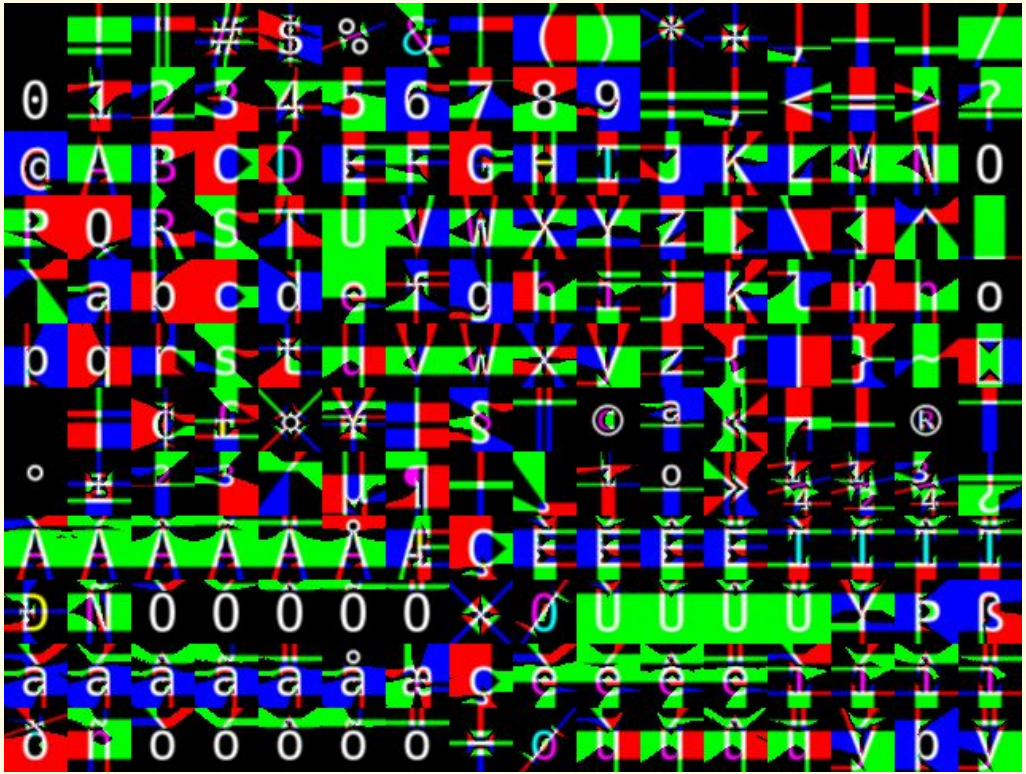
  - texture for text does not contain a color or pattern
  - contains a distorted font image generated from any normal font file
  - currently supports monospace fonts for Latin-1

  - fragment shader
    - computes a high resolution output using a multi channel (color) signed distance field algorithm

  - all the heavy lifting is part of CsPaint

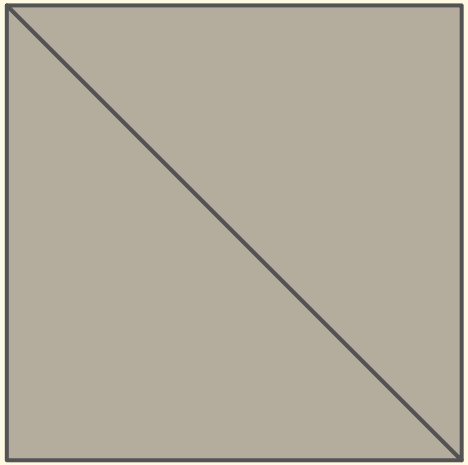
Rendering Text on the GPU

- Multi Channel Signed Distance Field Image (DejaVu Sans Mono)



Rendering Text on the GPU

- “rectangle” for one letter which is actually two triangles



Rendering Text on the GPU

- Fragment Shader to Render Text
  - written in GLSL (openGL shading language)
  - source code from demo/resources/text.frag

  - 5 inputs from the vertex shader
  - 1 input from the uniform buffer
  - 1 output containing the color, stored as part of the frame buffer

```
const float smoothing = 1.0/64.0;

float median(float r, float g, float b) {
   return max(min(r, g), min(max(r, g), b));
}
```

Rendering Text on the GPU

- Fragment Shader to Render Text
  - code to compute the actual outColor removed for readability
  - remainder of this source implements the MSDF sampling algorithm

```
void main() {
  vec3 fontSample = texture(fontSampler, texCoords).rgb;
  float sigDist   = median(fontSample.r, fontSample.g, fontSample.b);
  float opacity   = smoothstep(0.5-smoothing, 0.5 + smoothing, sigDist);

  if(opacity < 0.05) {
    discard;
  }

  outColor = vec4(outColor.rgb, opacity);
}
```

Useful Links

- Vulkan API
  - LunarG is the organization which provides the main Vulkan SDK
  - key element is the loader which is responsible for the bridge between your program and the vendor supplied graphics drivers
    - supports various layers
    - supports multiple GPUs
  - tools for compiling shaders to SPIR-V binary output
  - other development tools
    - https://www.lunarg.com/vulkan-sdk

  - informative overview of LunarG
    - https://www.youtube.com/watch?v=wWYRFwIHdJc

Useful Links

- Vulkan Tutorials
  - https://vulkan-tutorial.com/
  - https://vulkan.lunarg.com/doc/view/1.1.101.0/windows/tutorial/html/index.html
  - https://gpuopen.com/understanding-vulkan-objects/
  - https://developer.nvidia.com/transitioning-opengl-vulkan

3D Design

- Modeling
  - numerous open source and proprietary programs for modeling
  - selected Blender
    - open source 3D graphics modeling application
    - professional quality
    - version 2.8 released July 2019
    - several good youtube channels which use Blender
    - blender guru has a really good 9 part tutorial series
      - https://www.blenderguru.com/

Where is CsPaint Headed

- Roadmap
  - enhance the graphics demo
  - allow some of the defaulted parameters to be configured
  - add a slightly higher level API to CsPaint
  - provide sample fragment shaders
  - enhance 2D graphics support
  - create distance field textures on the fly for a given font
  - add text shaping using the Harfbuzz version 2 library
  - potentially link with CsString for Unicode support
  - enhance API documentation
  - incorporate user contributions

Integrating CopperSpice with CsPaint

- Roadmap
  - main application window will be a Vulkan surface
  - internal paint system redesigned to call CsPaint
  - all controls rendered using Vulkan
  - majority of controls will render on the GPU without modification
    - push button, radio button, check box, combo box
      table view, tree view, calendar, line edit, text edit
  - on Windows and Unix CopperSpice will use CsPaint directly
  - on OS X MoltenVK wrapper will be used to translate Vulkan
  - some level of fallback may be maintained for systems which do not support the Vulkan API

Bonus Slides

- Extra Information
  - CsPaint Documentation
    - https://www.copperspice.com/docs/cs_paint
  - repository for CsPaint
    - https://github.com/copperspice/cs_paint
  - Time permitting
    - three additional slides covering:
      - shaders, pipelines, frame buffer, depth buffer

Terminology

- Shader Categories
  - vertex shader
    - transforms a 3D position into 2D coordinates
    - executes once per vertex
  - tessellation shader
    - decompose shapes into smaller components, optional
  - geometry shader
    - alters the vertex shader output, optional
  - fragment shader
    - uses lighting and textures to calculate colors
    - executes at least once per pixel or fragment (partial pixel)

Terminology

- Graphics Definitions
  - pipeline
    - before a draw command is added to a command buffer you must create a pipeline object which sets a whole lot of options
    - shader handles, descriptor sets, push constants, depth buffer
    - winding direction, culling options, viewport
    - stencil, scissor, blending

  - command buffer
    - accumulates draw commands which are executed on the GPU at a later time
    - only way to synchronize these commands is by using a barrier
      - set semaphore, query semaphore
      - wait for pipeline stage (such as wait for a vertex shader to finish)

Terminology

- Graphics Definitions
  - frame buffer
    - no default buffer exists since displaying an image is optional
    - typically contains one image
    - not required if no images are displayed
  - depth buffer
    - a 3D mesh has perspective and to draw it realistically some triangles must appear in front of other triangles
    - a depth buffer is used to determine what part of the mesh is closer to the camera
    - misconfiguring this buffer will result in far away objects being rendered on top of closer objects

Presentations

- ❑ Why CopperSpice, Why DoxyPress
- ❑ Compile Time Counter
- ❑ Modern C++ Data Types (references)
- ❑ Modern C++ Data Types (value categories)
- ❑ Modern C++ Data Types (move semantics)
- ❑ CsString library (unicode)
- ❑ Multithreading in C++
- ❑ Multithreading using libGuarded
- ❑ Signals and Slots
- ❑ Build Systems
- ❑ Templates in the Real World
- ❑ Copyright Copyleft
- ❑ What's in a Container
- ❑ Modern C++ Threads
- ❑ C++ Undefined Behavior
- ❑ Regular Expressions
- ❑ Using DoxyPress
- ❑ Type Traits
- ❑ C++ Tapas (typedef, forward declarations)
- ❑ Lambdas in C++
- ❑ C++ Tapas (typename, virtual, pure virtual)
- ❑ Overload Resolution
- ❑ Futures & Promises
- ❑ Special Member Functions
- ❑ C++ in Review
- ❑ Thread Safety
- ❑ Constexpr Static Const
- ❑ When Your Codebase is Old Enough to Vote
- ❑ Sequencing, Linkage, Inheritance
- ❑ Evolution of Graphics Technology
- ❑ GPU, Pipeline, and the Vector Graphics API
- ❑ Rendering 3D Graphics
- ❑ Declarations and Type Conversions
- ❑ C++ ISO Standard
- ❑ Inline Namespaces
- ❑ Lambdas in Action
- ❑ Any Optional
- ❑ std::variant

Please subscribe to our YouTube Channel
https://www.youtube.com/copperspice

Libraries

- CopperSpice
  - libraries for developing GUI applications
- CsPaint Library
  - standalone C++ library for rendering graphics on the GPU
- CsSignal Library
  - standalone thread aware signal/slot library
- CsString Library
  - standalone unicode aware string library
- libGuarded
  - standalone multithreading library for shared data

Applications

- KitchenSink
  - contains 30 demos and links with almost every CopperSpice library
- Diamond
  - programmers editor which uses the CopperSpice libraries
- DoxyPress & DoxyPressApp
  - application for generating source code and API documentation

Where to find CopperSpice

- www.copperspice.com

- ansel@copperspice.com
- barbara@copperspice.com

- source, binaries, documentation files
  - download.copperspice.com

- source code repository
  - github.com/copperspice

- discussion
  - forum.copperspice.com