DEVELOPER SUMMIT

March 10–13

WELCOME

ArcGIS Runtime SDKs: Core Display Architecture Performance Tips and Tricks

Christian Venegas and Ralf Gottschalk

WARNING!

The following presentation and demos have been rated as Extremely Nerdy and contains information that is not suitable for some viewers

EXTREMELY NERDY

EN

ESRI

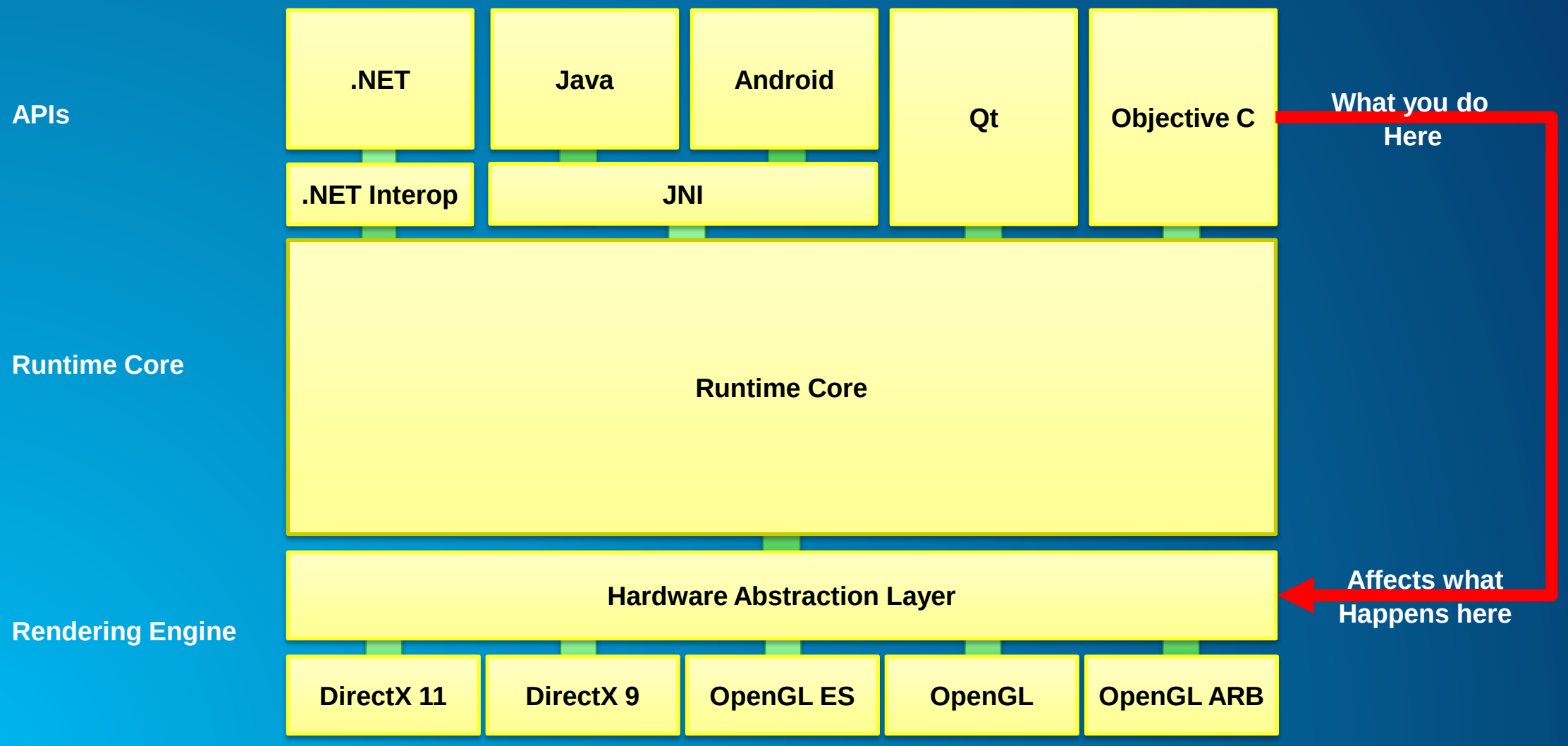
What We Will Cover Today

- Dive deep into the guts of the ArcGIS Runtime Rendering Engine
  - Use NVIDIA Nsight to see how rendering works
  - Once you understand the way the Runtime renders data you can make the right decisions for your application and target device
- General tips for overall performance gains with rendering in the Runtime
- Focus of this presentation is providing you with the knowledge to build high performance ArcGIS Runtime Apps

What is the ArcGIS Runtime?

- Set of APIs for building native applications on a wide range of platforms
- Built on a single C++ codebase compiled natively for each platform
  - No platform dependency (no COM)
  - Allows us to leverage the platform to maximize performance
  - Build functionality once and use it everywhere
- Rendering Engine
  - OpenGL, OpenGL ARB Assembly Language, OpenGL ES 2.0, DirectX 9, DirectX 11
  - Rendering selected for platform for optimal performance
  - Designed to work with both Desktop and Mobile Devices

Understanding the ArcGIS Runtime Architecture



Core Rendering Modes

- Static
  - Tiled Layers, Dynamic Map Service Layers, Feature and Graphics Layers (option)
  - Images are converted to textures
  - Textures are rendered on the GPU
- Dynamic
  - Feature and Graphics Layers (option), Grid Layer, Label Layer, GPS Layer
  - Symbols are converted to textures
  - Geometries are converted to triangles and texturized at render time

DEMO

ArcGIS Runtime Rendering Modes

Why different modes?

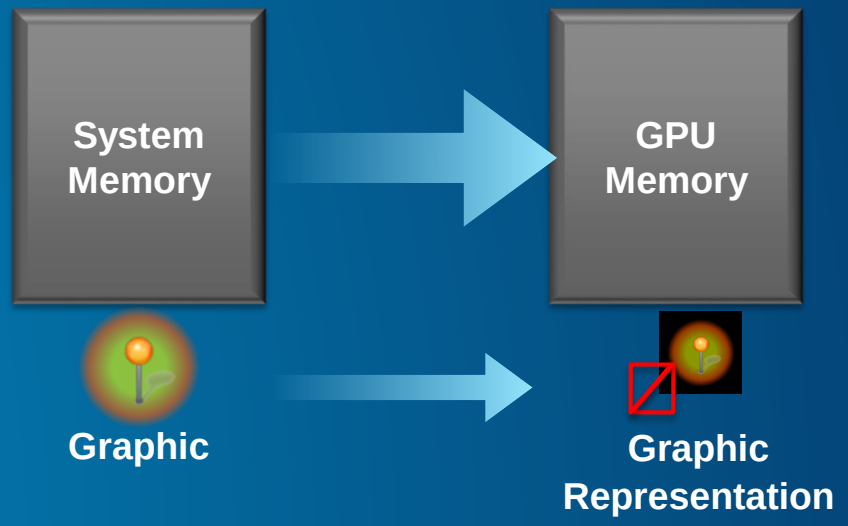
- We are building a GIS solution
- We don’t know the data ahead of time
- Must support a variety of use cases and workflows
- Performance is paramount
  - You choose the right mode for your use case and device

Graphics Processor Basics – What you should know

- Graphics card contains a dedicated processor (GPU) and memory which allows us to offload work from the CPU
  - Designed to perform complex rendering calculations
  - Produces less heat and uses less power than the CPU
- Execution occurs asynchronously until a state change is necessary
  - Runtime example: drawing points to drawing lines
  - When a state change occurs the GPU flushes the existing pipe
  - We batch as many things as we can to minimize this and keep the UI interactive
- Runtime renders by creating textures and triangles
  - We generate textures in many different ways depending on the layer

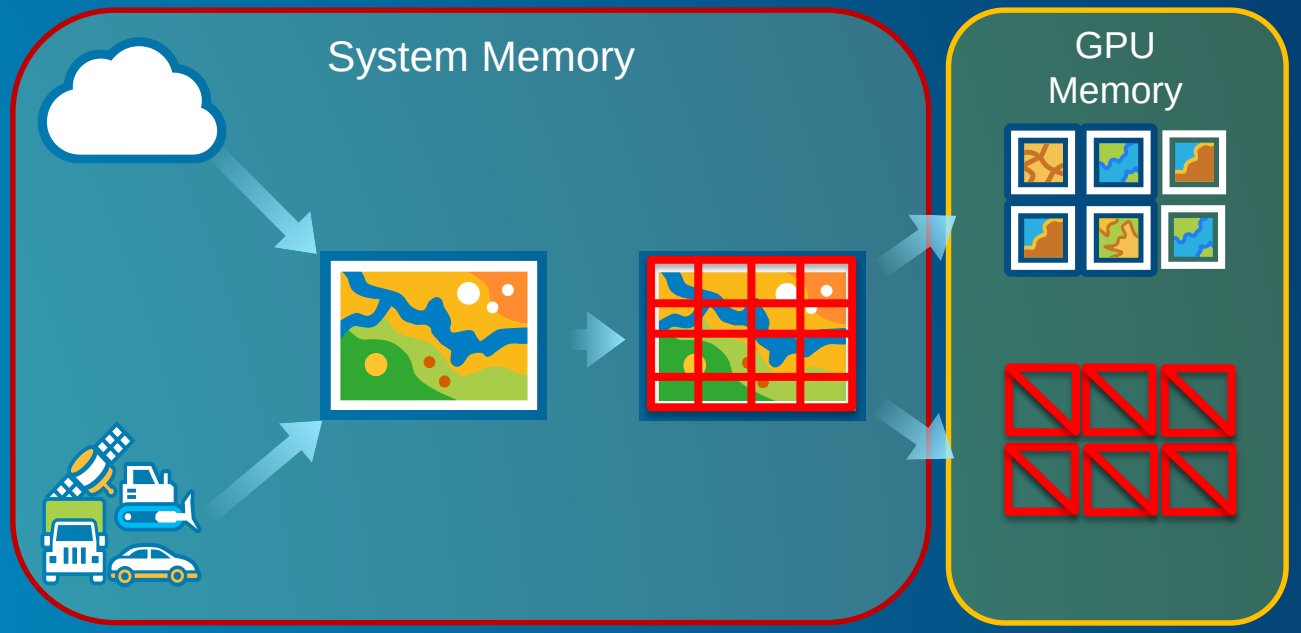
Graphics Processor Basics – What you should know

- Runtime data is stored in both system memory and graphics card memory
- Switching from system memory to GPU memory has a performance cost
  - Knowing when and how this occurs can help you build a high performance app
- Once data is in GPU memory rendering is fairly cheap
- Rendering Modes directly impact
  - Where data is stored
  - When and how it gets transferred
    - Runtime optimizes this as much as possible



Static Mode Rendering Explained

- Collections of Graphics on System Memory (Geometry and Symbol)
  - Graphics rendered as paths onto an image
- Or receive images from another source (Cloud or a TPK)
- Image is broken into blocks for partial updates
- Blocks are converted into textures for the GPU to render
- Triangles are created based on textures for proper placement



DEMO

Deep Dive into the Static Rendering Mode

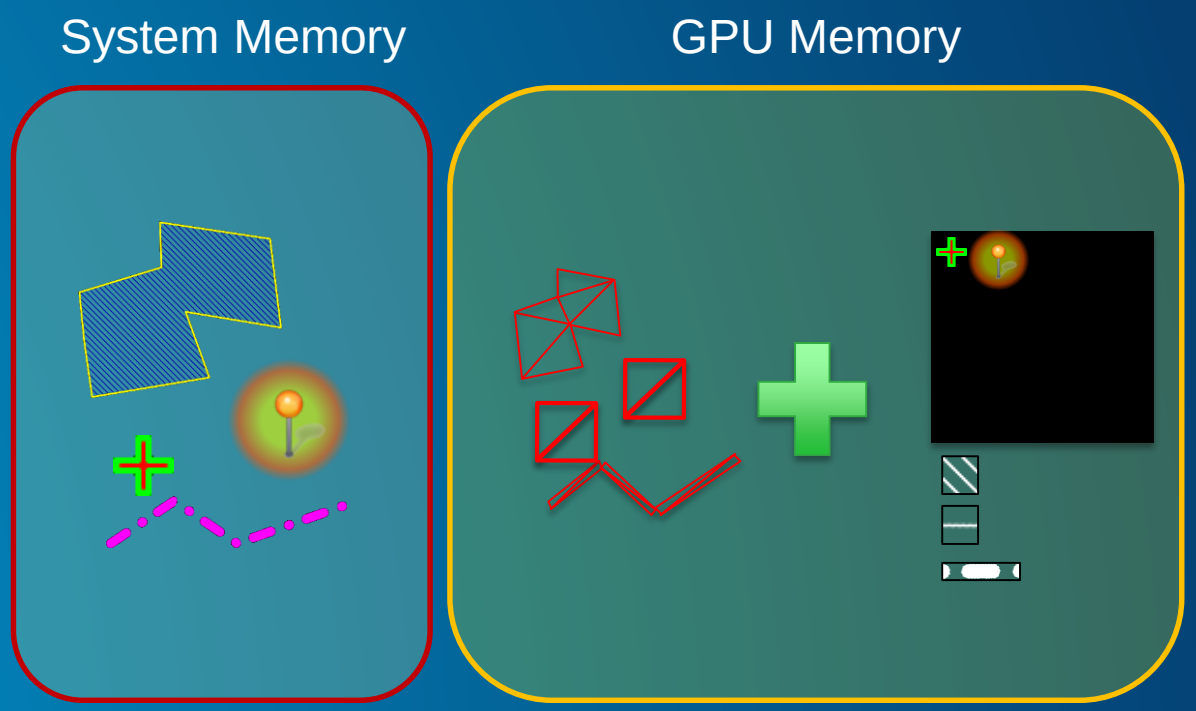
Static Mode Summary

- Designed for cartographic quality
  - Whole graphic is rendered as a path
- Scales up well
  - Only regenerate textures when we need to (passive rendering)
  - Increasing or decreasing the number of graphics within a layer does not affect number of textures on the GPU

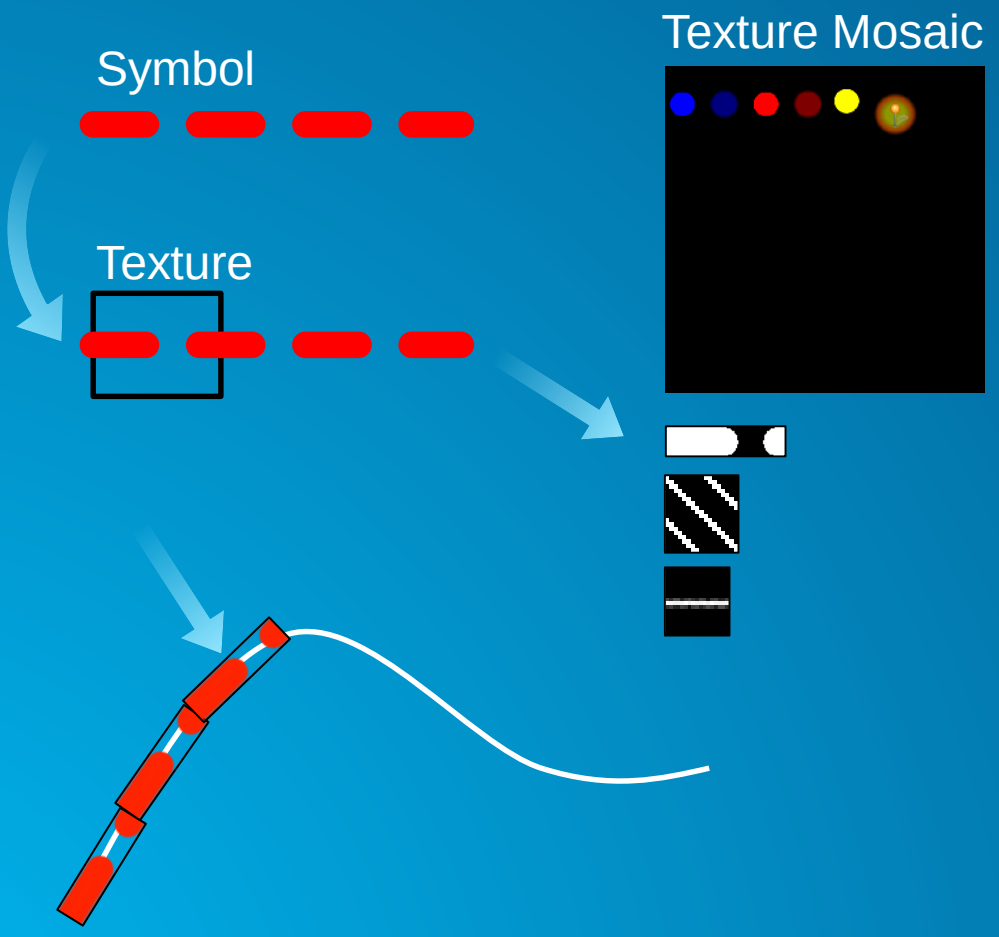
- Can be system memory intensive
- Updates to graphics require redraws of the image
- View changes require full or partial updates to image
- Rendering primarily done through CPU

Dynamic Rendering Mode

- Collections of Graphics on System Memory
- Symbol converted to textures and stored in a texture mosaic on the GPU
- Geometries are converted to triangles
- At render time the textures are applied to the triangles



Dynamic Mode Symbol Rendering



- Textures are created from the symbols
- Points symbols pushed to the texture map
- Lines and Polygons symbols
  - Snapshot of the the symbol pattern
- Texture is overlaid on the geometry
  - Extremely fast
- For simple symbols color can be applied at render time

DEMO

Deep Dive into the Dynamic Rendering Mode

Dynamic Mode Summary

- Entire graphic representation lives on the GPU
  - No CPU update required when panning, zooming, or rotating the map
- Some graphic changes can be applied directly to the GPU state
  - Like moving and animating
- Graphics can remain screen aligned while map is rotating

- Number of and complexity of graphics can impact GPU resources
- Symbology could look a little different
  - Performance is the priority
- Devices do have a finite amount of texture memory

Graphics and Feature Layers – Which Mode Should I Use?

- Number of Graphics vs GPU memory and power
  - Static Mode creates the same number of textures regardless of number of graphics
  - Dynamic Mode graphics live in GPU memory, more graphics, more GPU work
- Battery Life
  - Static CPU – Dynamic GPU
  - GPU is much more efficient with battery life
- What is the tipping point?
  - Depends on the use case, data, and the target device

DEMO

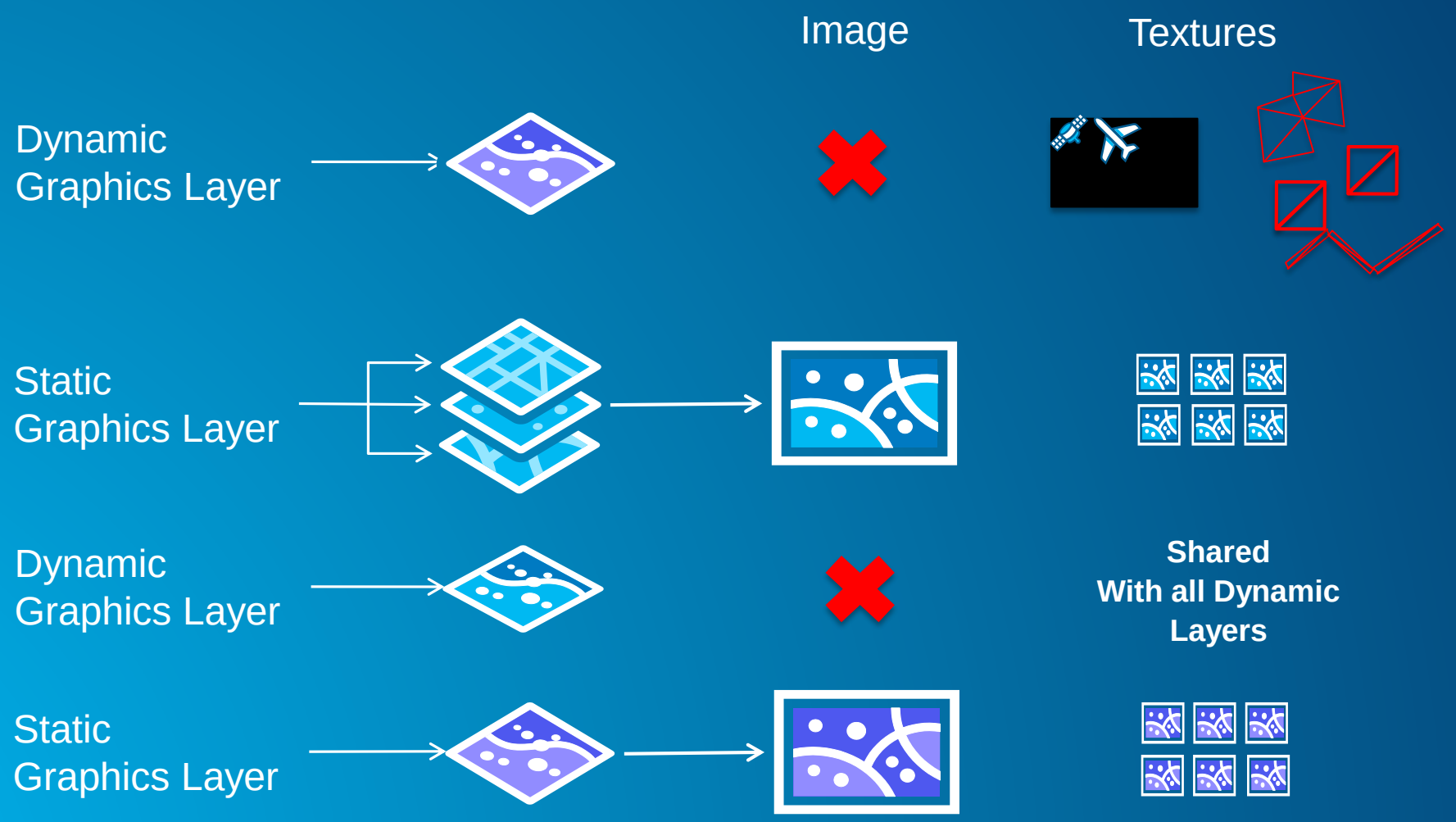
Static and Dynamic Rendering Modes

1 Graphic vs 10,000 Graphics

Graphics and Feature Layers – Which mode should I use?

- Dynamic mode
  - Number and order of layers not as important
  - Resources are shared between all layers on the GPU
- Static Mode
  - Number and order of layers matters
  - Layer ordering can affect resource sharing
- Core will consolidate layers whenever possible
  - Minimize textures
  - Optimize system memory usage

Static Mode Layer Consolidation

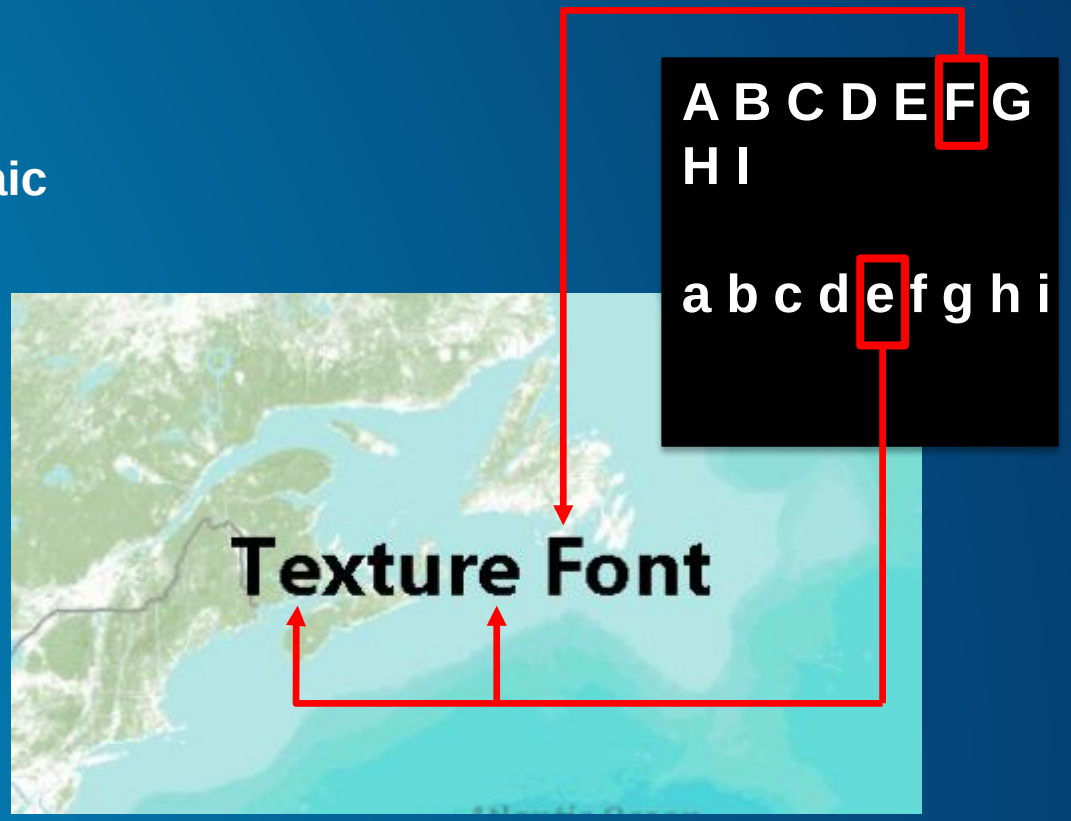


DEMO

Static Mode Layer Consolidation

Text Rendering

- Static Mode
  - Text Rendered Directly to the Image
- Dynamic Mode
  - New in 10.2.4! Texture Fonts
  - Fonts converted to Glyphs onto the Texture Mosaic
  - Glyphs can be reused for all Text Symbols



Text Symbols – Which mode should I use?

- Cannot choose mode with labeling
  - Labels are dynamic – move, appear, disappear
  - Point and Polygon labels tend to be screen aligned
    - Only Dynamic Mode can do this effectively
- Static Mode
  - Static Text, lots of fonts, size changes
  - Concerned about Texture Memory
- Dynamic Mode
  - Same font and size
  - Label like effect when panning and zooming

Graphics and Feature Layers – Which mode should I use?

- Dynamic Mode
  - Geometries are converted to triangles (vertices)
  - My require additional vertices to be generated to properly represent the geometry than the original feature contained
  - Polygons also have outlines which need to be converted to vertices
  - Both of these require additional GPU resources to represent the data
  - Will require 2 draw operations to render a single polygon
- Static Mode
  - Number of textures are always the same
- Complexity of your geometry has an impact on performance

DEMO

Complex Polygon

Graphics and Feature Layers – Which mode should I use?

- **Static or Dynamic - Is there a rule I can follow with polygons?**
  - **Depends on the number of polygons and the device**
  - **Multiple large features with millions of vertices with bring even the strongest GPU to its knees**

Graphics and Feature Layers – Best Practices

- Static Mode
  - Is best for Feature Services
    - Unknown number of features
    - Unknown symbols
    - Unknown geometry complexity
- Dynamic Mode
  - Sketch or interactive layers
  - Real-time feeds with rapid updates
  - Points

Additional Tips for Improved Performance

- When adding and creating graphics use arrays
- Use a renderer whenever possible instead of individual symbols for each graphic
  - Avoids additional logic in core to handle duplicate symbols
- Create a new layer for each geometry type
  - Affects Dynamic Mode not Static mode
- Avoid layer level transparency
  - Requires additional draw operations and texture generation
  - Stick with symbol level transparency whenever possible
  - This also applies to other static layer drawing

Transparency

Symbol Level

Layer Level

DEMO

Performance Tips

What new for the Quartz Release?

- Hardware Abstraction Layer Performance Optimizations
- Additional DX11 functionality
- OpenGL 4.3 (Desktop) and OpenGL ES 3.1 (Mobile)
  - If available we'll use it
  - Better performance
  - 3D support
- Compute Shaders
  - Real-time GPU Driven Spatial Analysis
- Vector Tiles
  - High performance Base Maps

It's shiny…

DEMO

Sneak Peek

Rate This Session

www.esri.com/RateMyDevSummitSession