Esri International Developer Summit

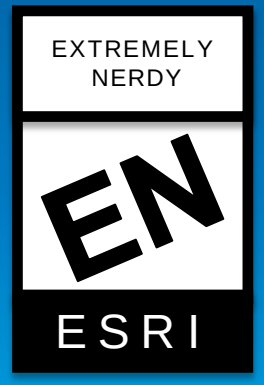
Palm Springs, CA

Squeezing Every Ounce of Performance from ArcGIS Runtime

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



MPL: Multiple Programming Languages
RCE: Runtime Core Exposure
NT: Nerdy Terminology

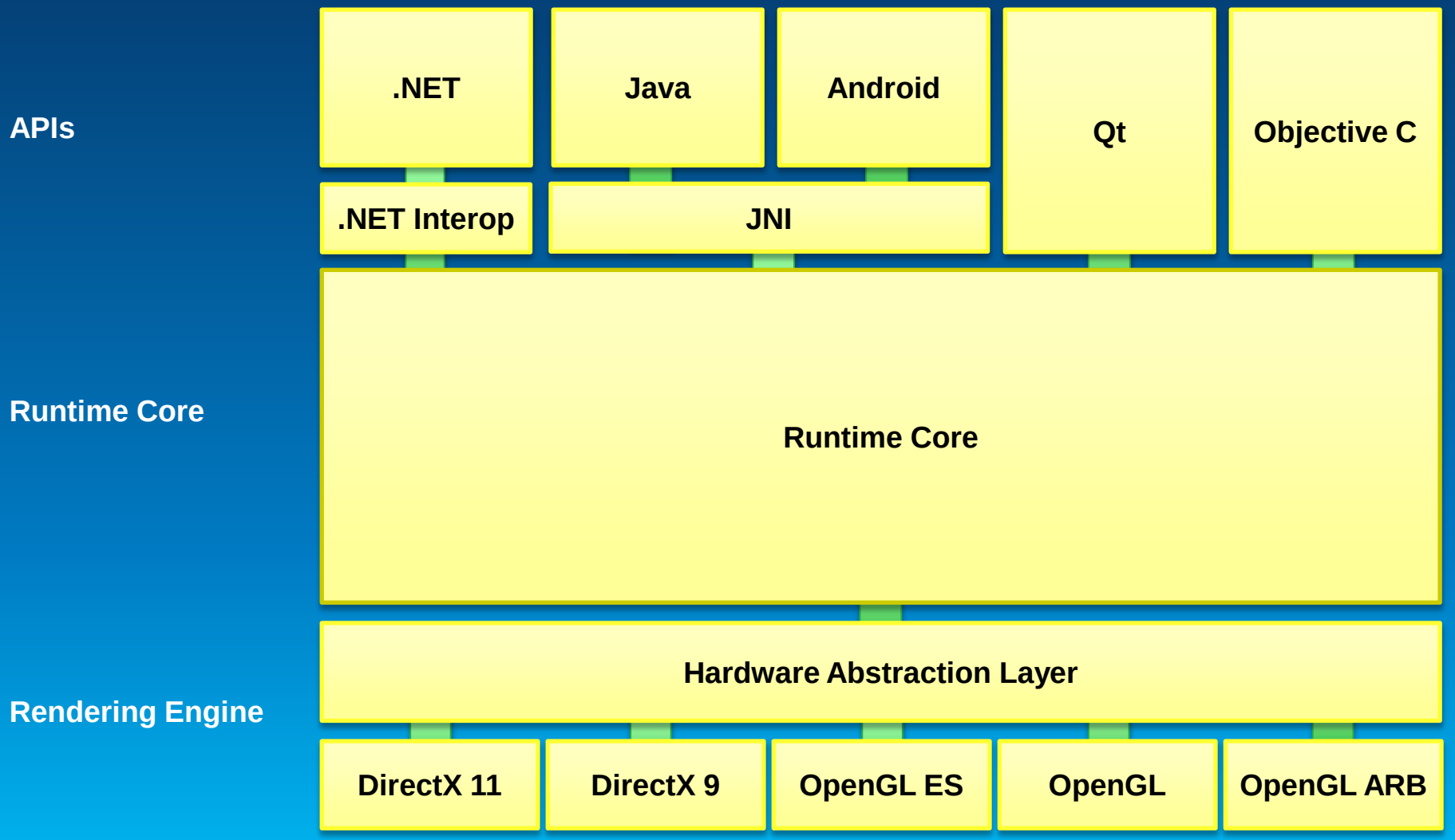
What We Will Cover Today

- Focus of the presentation is on building high performance ArcGIS Runtime Apps
- Deep dive 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 in the Runtime

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
    - Powerful Desktop Graphics Cards
      - NVIDIA, AMD, Intel
    - Heat and battery efficient Graphical Processor Units (GPUs)
      - Many render differently than desktop GPUs

Understanding the ArcGIS Runtime Architecture



Graphics Card Basics – What you should know

- Graphics card contains a dedicated processor (GPU) and memory which allows us to offload work from the CPU
  - GPU is designed to perform complex rendering calculations
- 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 Card 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 in GPU memory rendering is fairly cheap
- Some layers have different Rendering Modes
  - Rendering mode has a direct impact on where data is stored

Core Rendering Modes

- Static
  - Tiled Layer, 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

Why different modes?

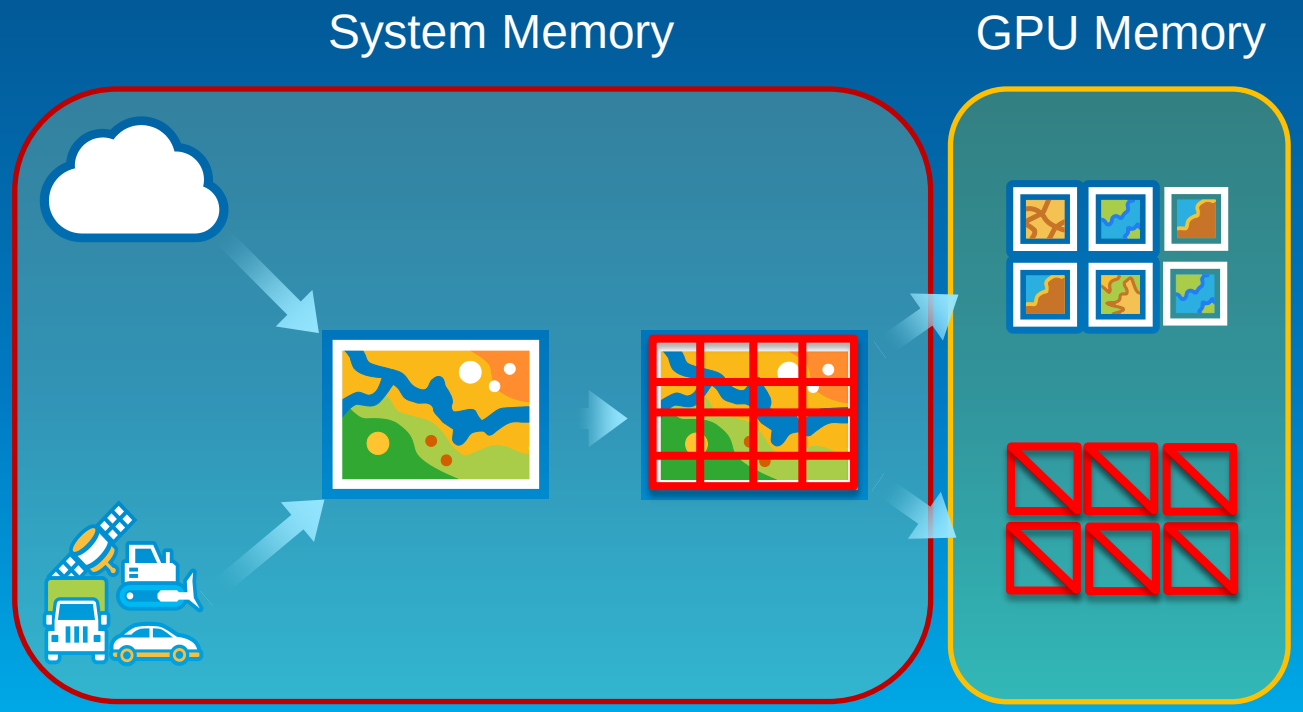
- We are building a GIS solution
- We don’t know the data ahead of time
- Must support a variety of use cases and workflows
- Performance is paramount
  - You choose the right mode for your use case and device

DEMO

ArcGIS Runtime Rendering Modes

Static Mode Rendering Explained

- Collections of Graphics on System Memory (Geometry and Symbol)
  - Path rendered onto image
- Image is broken into blocks for partial updates
- Blocks are converted into textures for the GPU to render
- Triangles are created based on textures



DEMO

Deep Dive into Static Rendering Mode

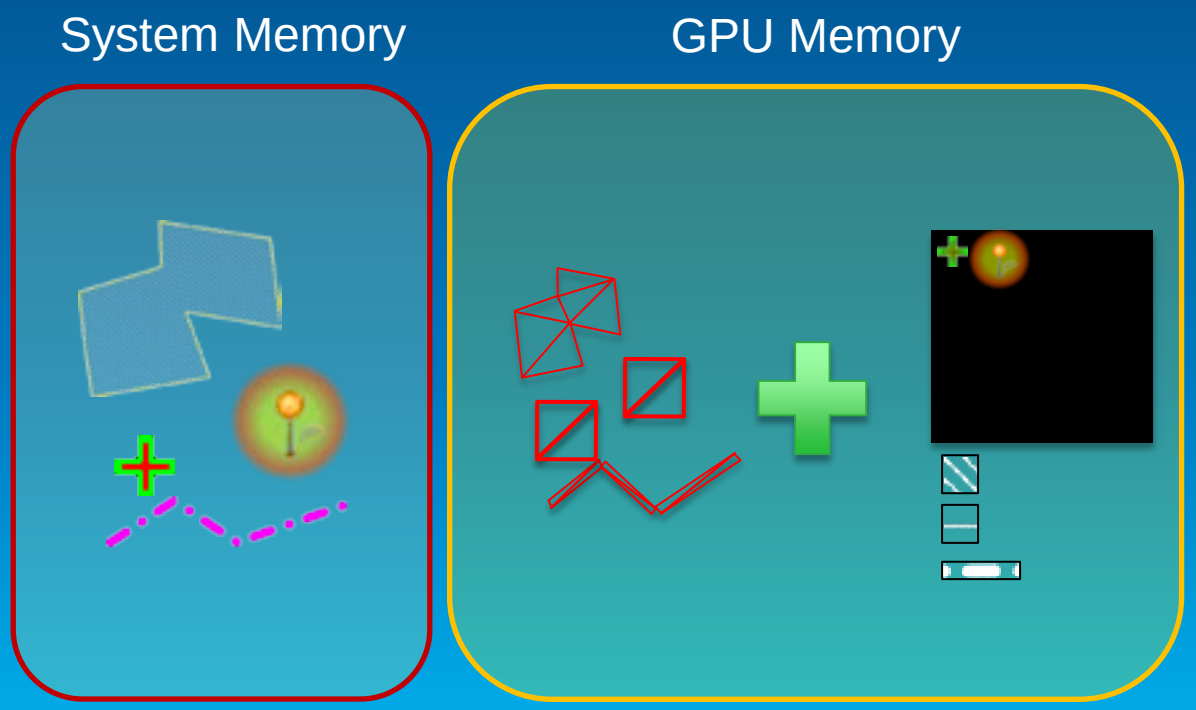
Static Mode Summary

- Designed for cartographic quality
  - Whole graphic is rendered as a path
- Scales up well
  - Increasing or decreasing the number of graphics within a layer does not affect number of textures on the GPU

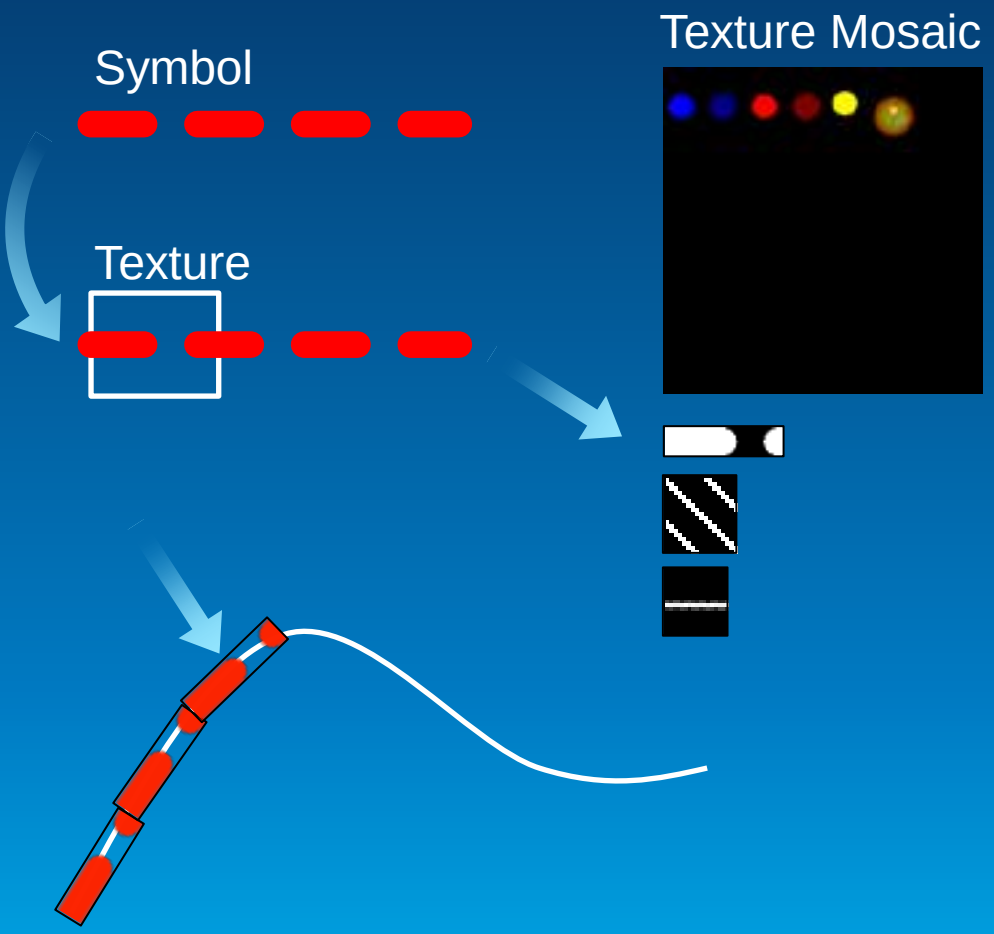
- Can be system memory intensive
- Updates to graphics require redraws of the image
- View changes require full or partial updates to image
- Rendering primarily done through CPU

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
- Simple symbols
  - Color can be applied at render time

DEMO

Deep Dive into 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 and complexity of graphics can impact GPU resources
- Symbology could look a little different
  - Performance is the priority

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
- Graphics in Dynamic Mode live in GPU memory
- What is the tipping point?
  - Depends on the use case, data, and the target device
  - All comes down to user interaction and use case

DEMO

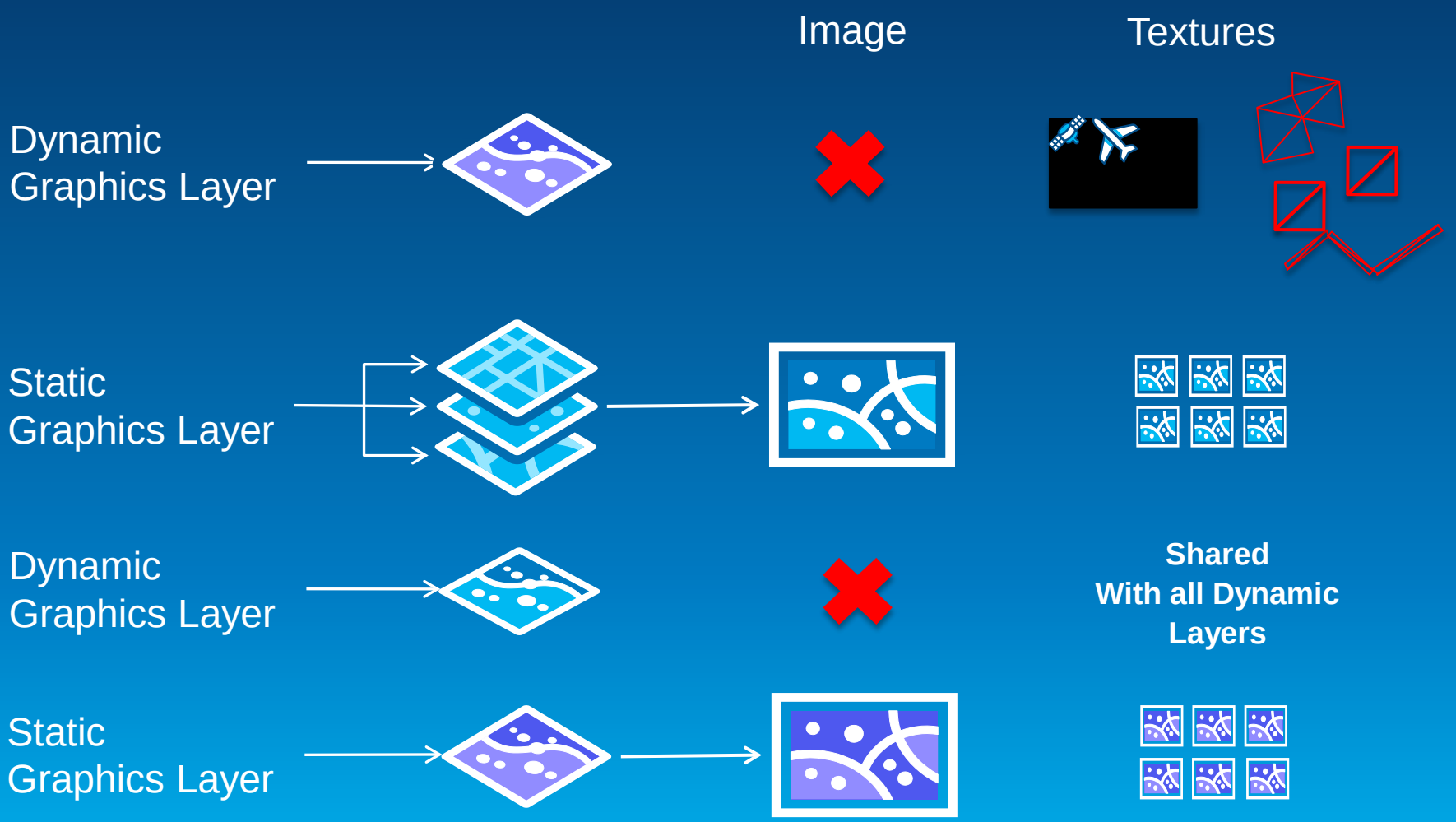
Static and Dynamic Rendering Mode

1 Graphic vs 10,000 Graphics

Graphics and Feature Layers – Which mode should I use?

- Core will consolidate layers whenever possible for maximum performance
- 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
  - Each layer renders to an image
  - Layer ordering can affect resource sharing

Static Mode Consolidation



DEMO

Graphics Layer Static Mode Consolidation

Graphics and Feature Layers – Which mode should I use?

- Complexity of your geometry
- In Dynamic Mode geometries are converted to triangles (vertices)
  - My require additional vertices to be generated to properly represent the geometry
- 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

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 symbol and 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

Transparency

Symbol Level

Layer Level

DEMO

Batching

10.2.2 Performance Enhancements

- Internally calling move when setting a new geometry on a point
  - Prevents regeneration of triangles
  - Just reposition geometry on graphics card
- Reduced number of instructions to GPU
- Reduced context switching on GPU
- Added more low level batching functionality
- Better reuse of resources and vertices

http://www.esri.com/events/devsummit/session-rater