ArcGIS Runtime: Maximizing Performance of Your Apps

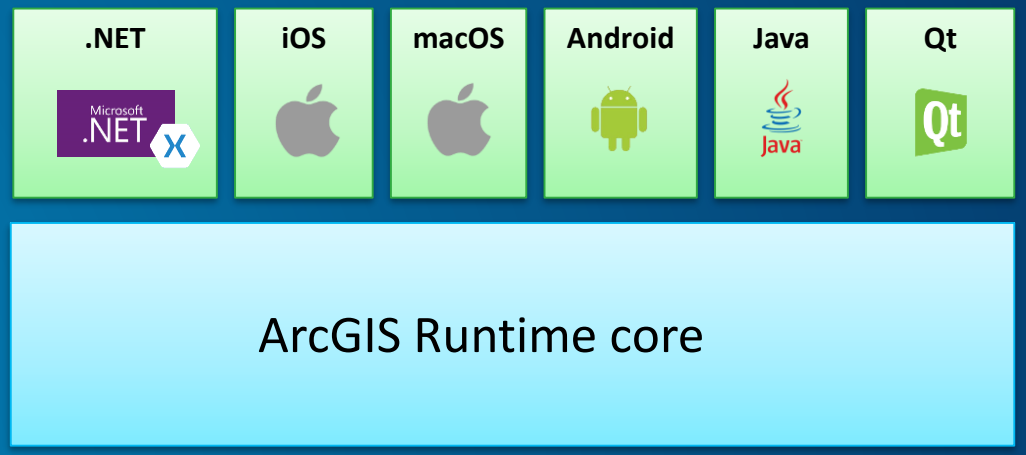
Will Jarvis and Ralf Gottschalk

Agenda

- ArcGIS Runtime Version 100.0 Architecture
- How do we measure performance?
- We will use our internal Runtime Core Test App and NVIDIA Nsight to understand how we render data in the Runtime
  - Static vs Dynamic Rendering Mode
  - Vector Tiles vs Raster Tiles
- Quick Demo of some rendering improvements we are working on

ArcGIS Runtime v100.0 new Architecture

- Most of the Runtime implementation has been moved down to C++
  - APIs can now focus on platform specific logic
  - We build functionality once in C++
    - New functionality can happen faster
- Public API Redesign
  - Unified API design across all platforms
  - Simplify developer workflows
- Thins the interop
  - Positive impact on performance (Java / .NET)

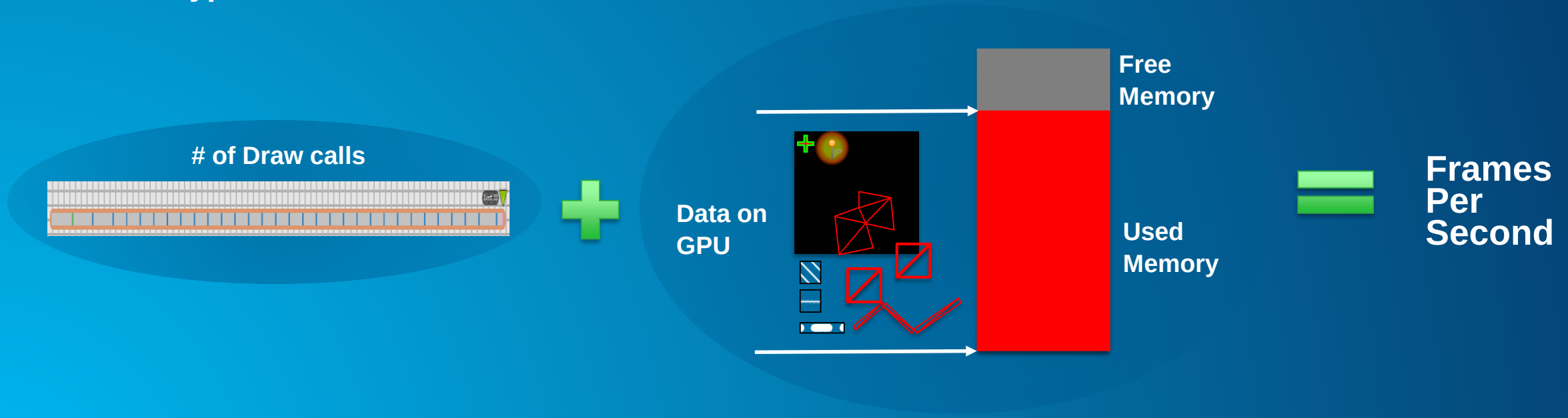


Runtime Display Architecture – Powered by the Core

- Powered by the Runtime Core
  - Supports DirectX 11, OpenGL, OpenGL ES 2 and 3, and Angle (for Qt)
- Active display
  - Operates on a pulse
  - Runtime determines what, if anything needs to draw
  - Executes draw commands CPU or GPU
- The time it takes to execute each pulse determines the refresh interval of the display
  - Measured as Fames Per Second (FPS)
  - More frames = smoother interaction with the map

Runtime Display Architecture - Rendering Performance

- Map interaction starts to suffer when there is more for the GPU to do
  - Amount of data to process
  - Loosely related to number of draw calls
  - Type of draw call

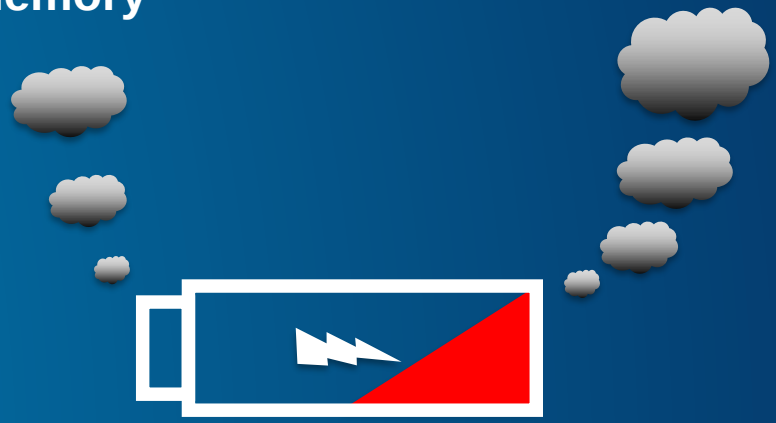


How does the type of call affect performance?

- Execution occurs on the GPU in an asynchronous queue until…
- State change is required
  - Think of this as a context switch
    - We have to switch the shader
    - Runtime example: drawing points then drawing lines or drawing tiles and then drawing graphics
  - When a state change occurs
    - Finish off all remaining draw calls
    - Runtime batches drawing as much as possible to keep the UI interactive and reduce the number of state changes
- State changes are normal and unavoidable
  - But the decisions you make with your data can help minimize their occurrence and maximize the performance of your app

Rendering – 2 Modes

- CPU and GPU based rendering
- CPU
  - Tons of processing power
  - Uses the system memory
  - Powerful processing at the expense of heat generation and battery usage
- GPU
  - Extremely efficient certain types of processing
  - A smaller amount of dedicated memory, or could share system memory
  - Uses less battery and generates less heat



GPU vs CPU Based Rendering

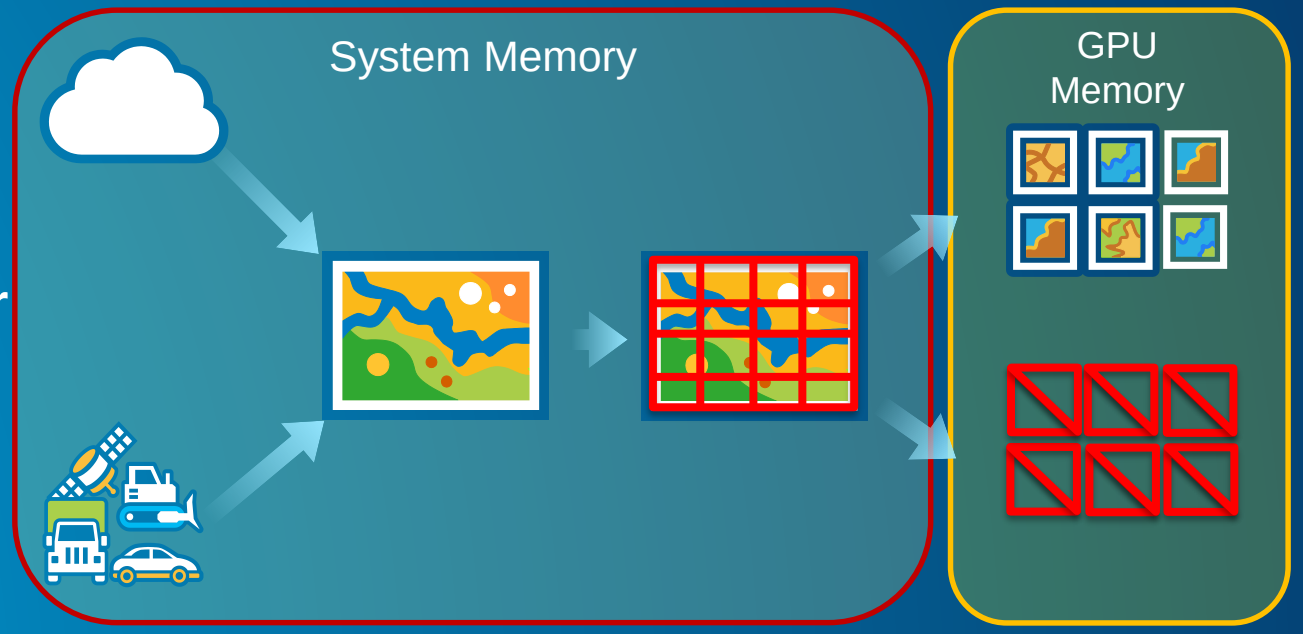
- Why not use the GPU for everything?
  - Well… it gets complicated
- Most maps in the Runtime are created on a Desktop
  - This mean we have to render any data you throw at us
  - Cartographic quality is important
  - Runtime supports a large variety of devices
    - Some great and some… not so great
    - GPU memory and processing power can be and issue
      - We are working on scaled functionality, but there still is a lot of work to do
- Transferring data between the CPU and GPU is expensive
  - It’s a onetime hit, but still a hit

Static and Dynamic Rendering Modes

- Static Mode:
  - CPU based rendering
    - On demand rendering
  - Designed for cartographic quality
  - The amount of data doesn’t really impact your frames per second
- Dynamic Mode:
  - GPU based rendering
  - Always rendering
    - We can do cool things directly on the GPU
  - The amount of data does impact your frames per second

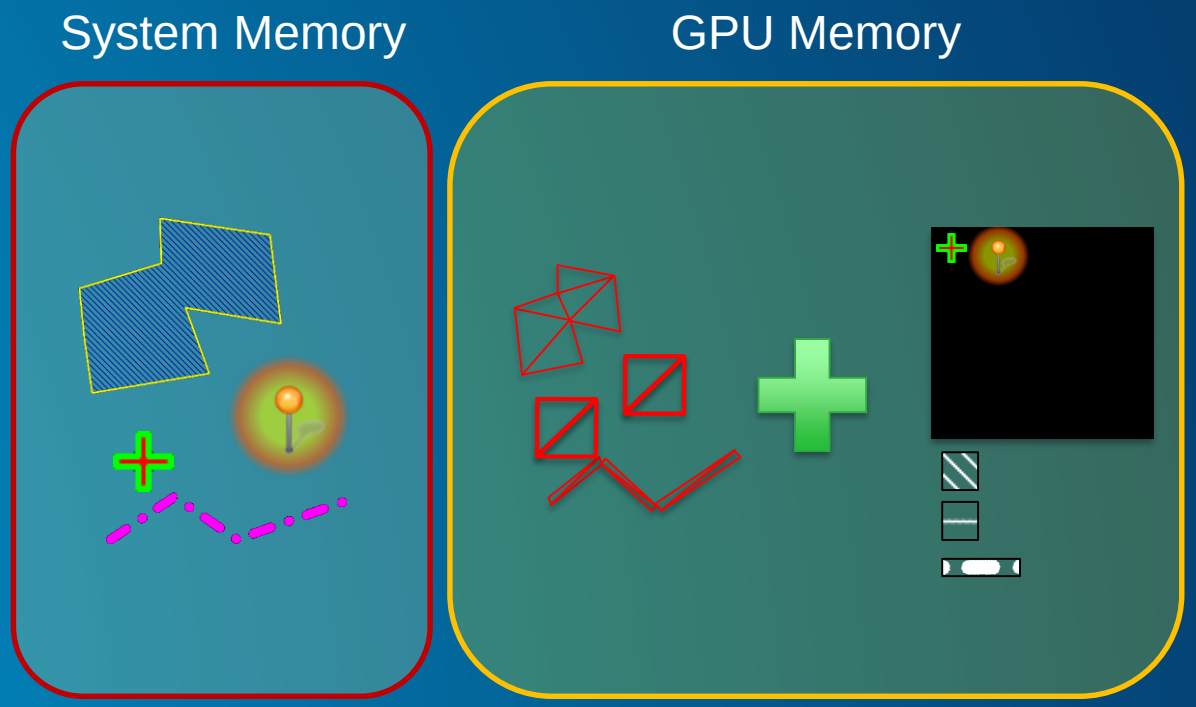
Static Rendering Mode for Graphics

- Graphics rendered as paths onto an image
  - Cartography is maintained
  - Great for Advanced Symbology – Multilayered Pro Symbology / Military Symbology
- Image is broken into blocks for partial updates
- Blocks are converted into textures
- Triangles are created based on textures for proper placement
- All static graphics layers will render together
  - No state change required

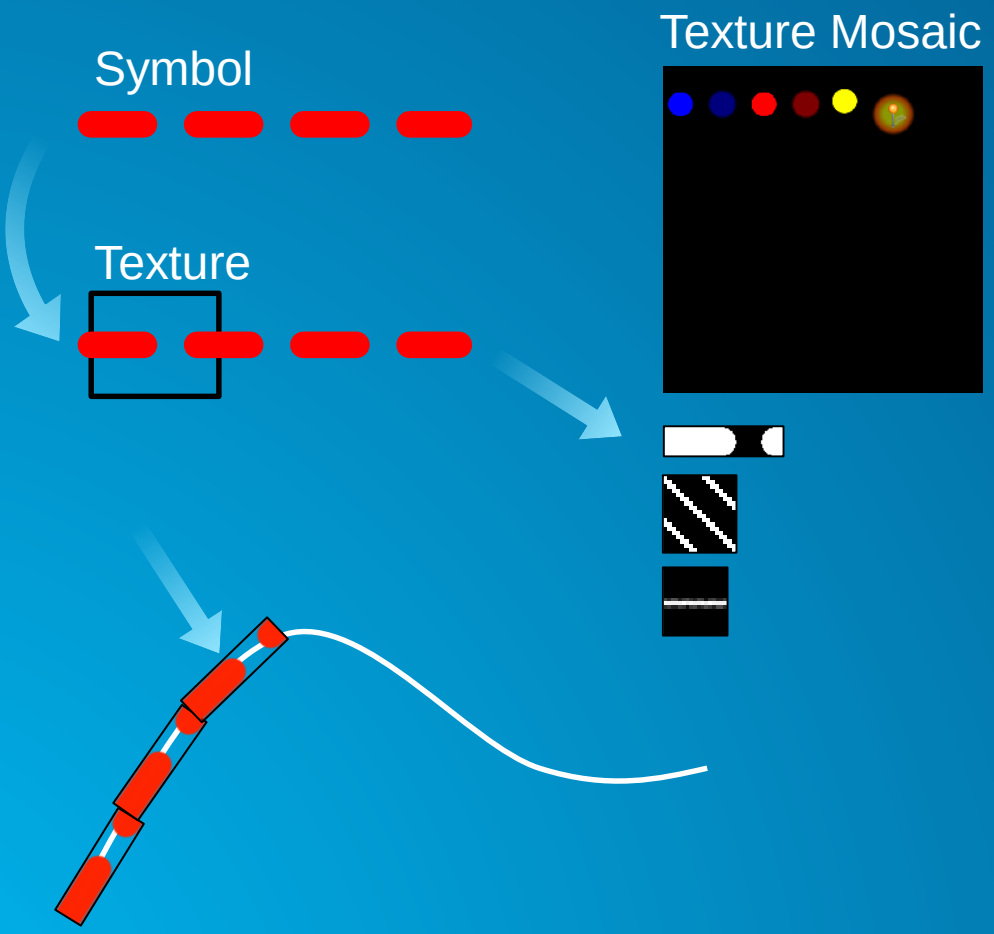


Dynamic Rendering Mode for Graphics

- We convert the symbols into textures and store them in a texture mosaic on the GPU
- Geometries are converted to triangles
  - Could lead to a lot of triangles
  - Geometry complexity will use more GPU resources
- At render time the textures are applied to the triangles
- Moving or rotating dynamic graphics performs great because
  - Dynamic graphics are always drawing



Dynamic Rendering Mode Symbol Rendering



- Points symbols pushed to the texture map
- Lines and Polygons
  - Snapshot of the the symbol pattern
- Texture is overlaid on the geometry
  - Extremely fast and inexpensive to render
  - Slight compromise in cartographic quality for lines and polygons (Advanced Symbols)
  - Gaps in textures from extreme angles are smoothed
    - Does require additional processing
    - At v100.0 this processing has been moved to the GPU

Static vs Dynamic Mode - Graphics

Will Jarvis

Static vs Dynamic Mode for Graphics

CPU

Static Rendering

Static Data
Only draws on change

GPU

Dynamic Rendering

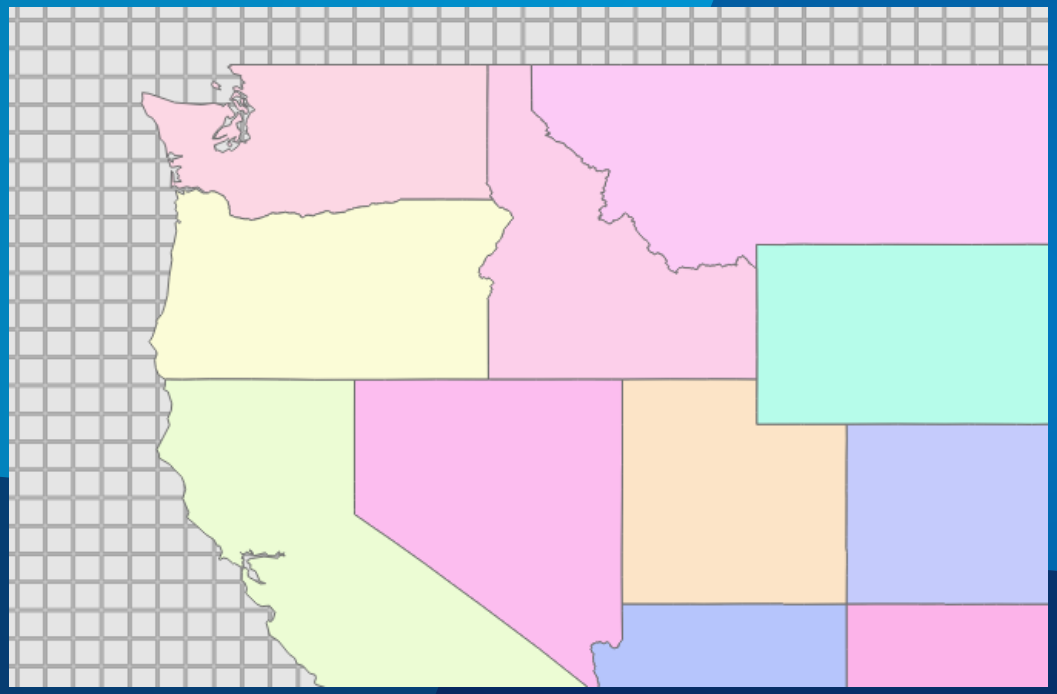
Map View Pulse

Dynamic Data
Draws every pulse

Resulting image pushed to GPU

Polygon Rendering

- Polygons are effectively 2 geometries
  - Polygon and it’s outline
    - This could lead to lots and lots of triangles
    - Ultimately leads to more data to process on the GPU
  - Does that mean there is a state change when switching between polygons and lines?
    - Yes and no
    - Polygons with Simple Symbols as the outline rendered with one shader
      - No state change required
    - Polygons with Advanced Symbols
      - Due to the complexity of the symbol more processing is required for cartographic accuracy
- Static Mode
  - Same number of textures regardless of number of polygons or outlines



Polygon Rendering

Will Jarvis

Static vs Dynamic – Tile Rendering

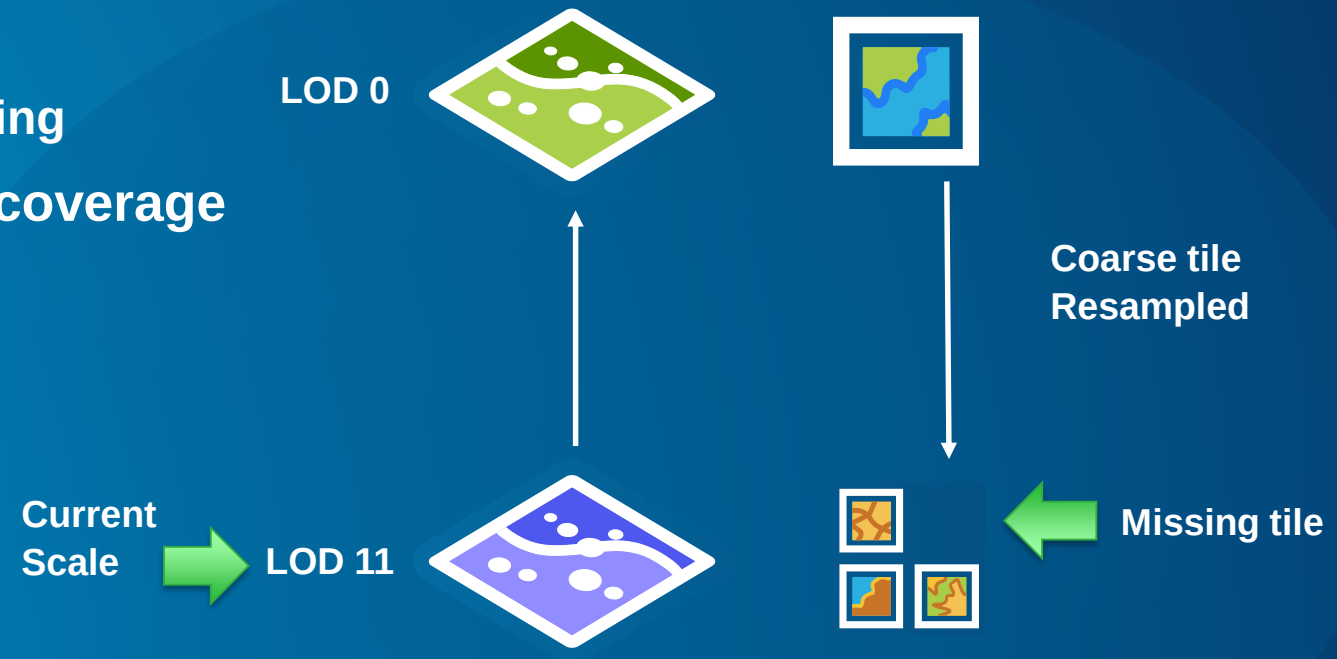
- **Raster Tile Layers**
  - Static Mode of Tiled Layers
  - Collection of Image Tiles cooked with a set schema at specific Levels of Detail (LOD)
  - Fast performance
  - It’s just an image no client level processing is required
- **Vector Tile Layers**
  - Dynamic Mode of Tiled Layers
  - Highly compressed vector data cooked with a set schema at specific Levels of Detail (LOD)
  - Client must render the data on the fly

Raster vs Vector Tiles – Raster Tile Layers

- Images cooked at a specific DPI
  - Easily converted to textures
    - Extremely fast rendering performance
  - Large areas might take up quite a bit of disk space
  - Depending on the DPI they might not looks that great on high res devices
  - Map authoring does not matter as the entire map is backed into tiles
- At v100.0 they behave slightly different depending on the layer collection the tiles reside in

Raster vs Vector Tiles – Raster Tile Layers

- When used in the Basemap Layers collection
- We use “Skin of the Earth” style rendering
  - Coarse tiles are resampled as a place holder for missing tiles until they become available
  - This way we will always render something
- Designed specifically for continuous coverage data



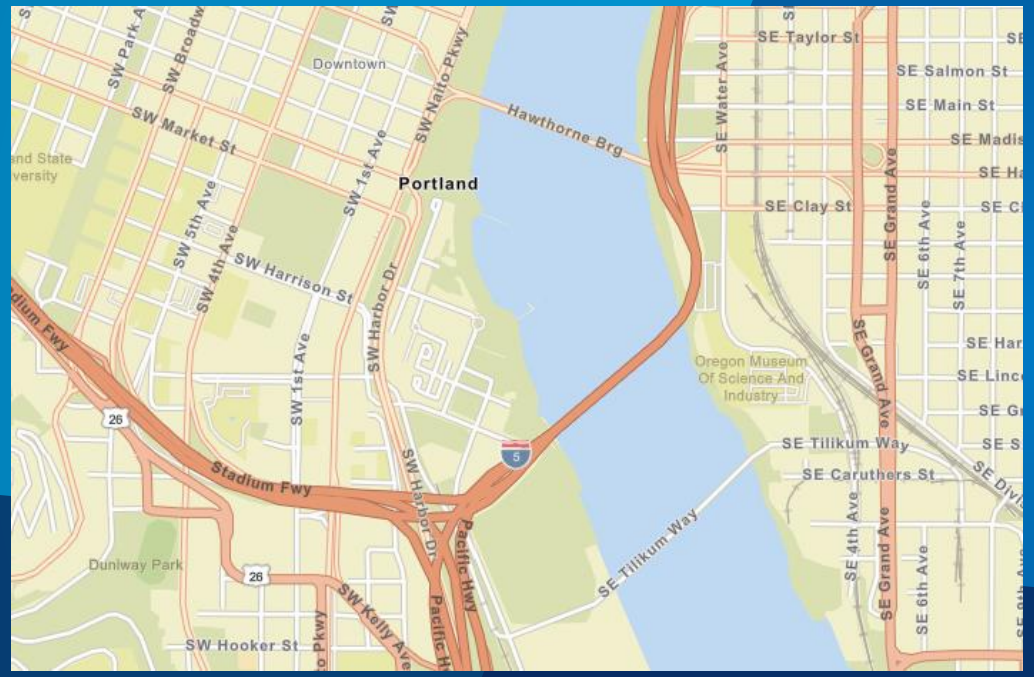
Raster vs Vector Tiles – Raster Tile Layers

- When used in the Reference Layers collection
  - Designed to only render the LOD we are at
  - Specifically for tiles with lots of transparency
    - Place names, streets, etc
  - When zooming to a new LOD previous LOD is cleared out

- When used in the Operational Layers collection
  - Basically a mix of both Basemap and Reference layers
  - Resampling is much less
  - Keep older tiles around when necessary to avoid the data from flashing

Raster vs Vector Tiles – Vector Tile Layers

- Work great on high DPI devices
  - Small in size
    - Indexed vector tiles are even smaller
  - Data is more dynamic
    - Labels and point data stay screen aligned
- Performance is data driven
  - Client needs to process and render the data, just like dynamic graphics
- Proper map authoring is critical to Vector Tile performance
  - More data, more layer result in more draw calls
  - These draw calls are fairly inexpensive, but the more your have…

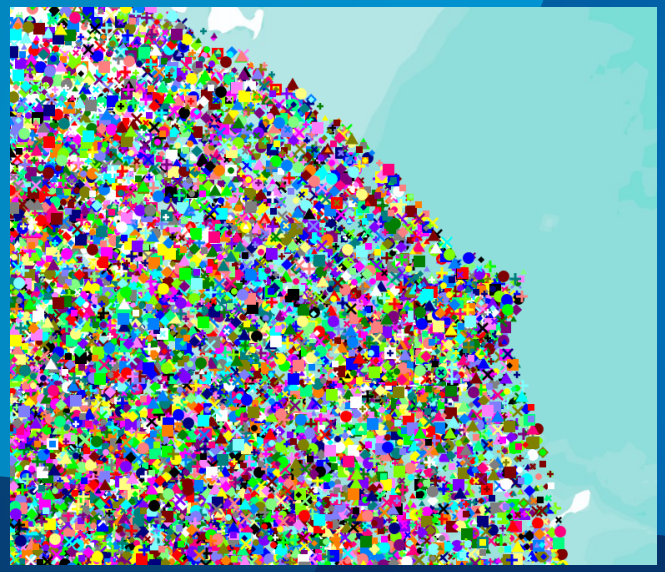


Raster vs Vector Tile Rendering

Will Jarvis

Fonts

- Wait, Fonts? What does that have to do with performance?
- ArcGIS Runtime v100.0 has an automatic font fallback mechanism
  - We try to find the font specified by the developer
    - If it does not exist we leverage the platform to find the next suitable font
    - If a font does not have a glyph we find the most suitable font that does
      - Example: Korean text, but your font has no Korean glyphs
      - This can take time depending on the text
- No font is universal for all platforms
  - It is the author / developer's responsibility to verify if the font exists



A Sneak Peek Into the Future

Ralf Gottschalk


esri
THE
SCIENCE
OF
WHERE