



# Anatomy of a Flash

Chiculita Alexandru | Computer Scientist | Adobe



# Agenda

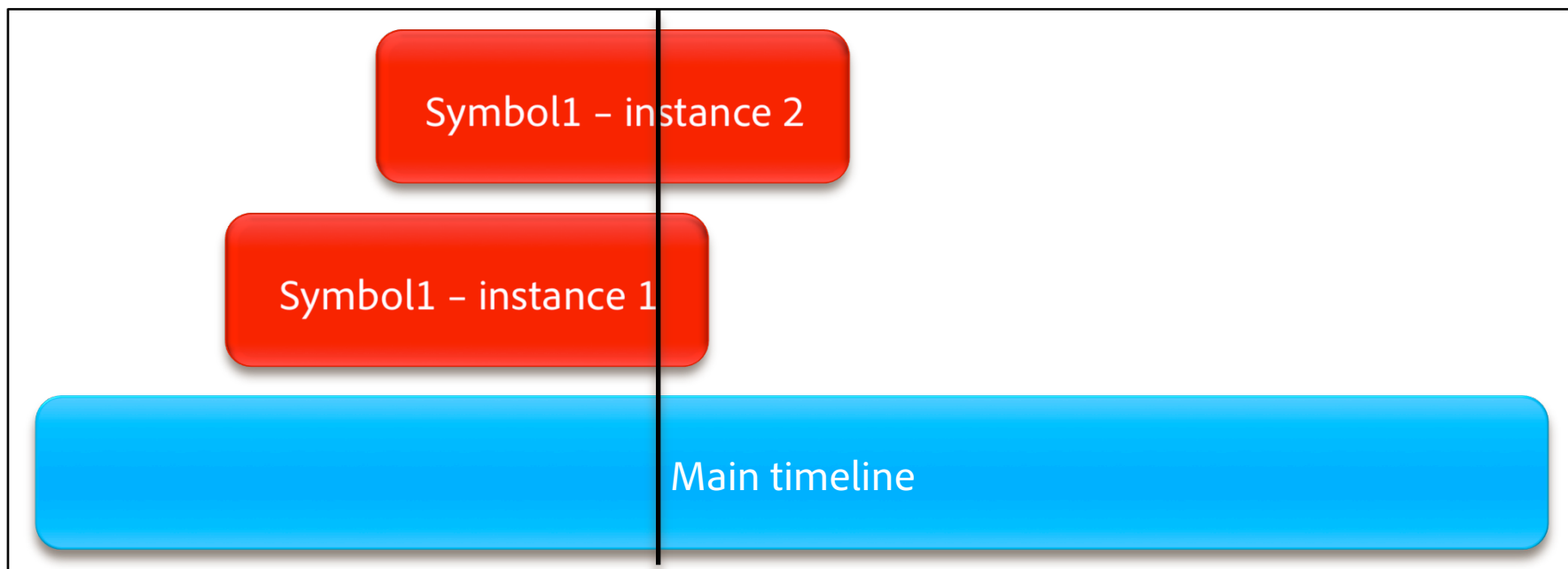
- SWF File Format
- In memory rendering tree
- CPU Rendering Model
- GPU rendering

# SWF File Format

- Tag based file format
  - Header
    - Initial frame-rate, SWF version, width / height, Metadata
  - Each tag is actually a command
    - Flash is actually an interpreter of SWF commands
- Tags are used to define a dictionary of characters
  - Movie Clips – each frame is a series of tags that create a display tree (eg. PlaceObject)
  - Fonts, Byte Arrays, Sounds etc.
- Characters are associated with AS3 classes

# SWF Threads

- Symbols are placed on main timeline using SWF tags or ActionScript API
- Symbols and main timeline advance frames in parallel
  - using the same framerate
- Video and Audio, ActionScript are not affected by the stage framerate



# Characters and AS3 objects

- Every symbol is a character
  - Shapes
  - MovieClips – Bundle of frames
  - Fonts
  - Sounds
  - ByteArray
  - Video
- Symbols placed on stage can have AS3 classes associated
- Scripting can create symbols by using their AS3 class

# Flash rendering model

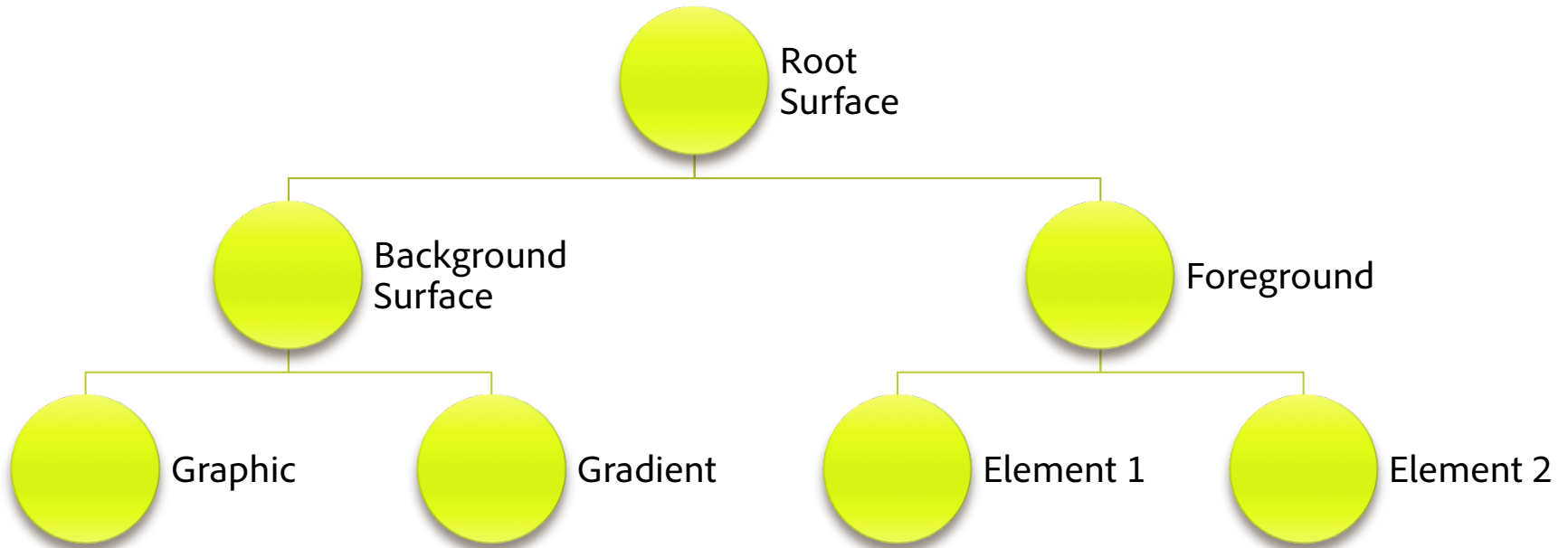
- Rendering Tree
  - Both AS3 and AS3 DisplayObjects are converted to SObjects internally
  - SObjects have a character associated
    - Based on the character type it has different drawing methods, but it all resumes to drawing fills with different source of colors
      - Shapes parse the SWF and create a bunch of edges
        - Edges accumulate in the rasterizer object using blend stacks
        - Strokes are dynamically converted to fills
      - Color sources:
        - Bitmaps, Video, Solids, Gradients, Pixel Bender, Masks (special color)
  - The rasterizer calculates the color for each pixel by sorting the edges from top to bottom, left to right using the blend stacks
    - Rasterizer is parallelized by dividing the screen by the number of CPU cores
    - Antialiasing is applied only on edges of the fills (ex. increased speed when rendering solids)

# Flash rendering phases

- Computation
  - Maximum 3 invalidation regions are computed
- Edge & Colors
  - SObjects tree is traversed and a list of edges is created
  - Edges have colors associated
    - Strokes are converted to edges
    - Colors are sources of display data, eg. Bitmaps, Video, Solid fills, Gradients
- Rasterization
  - Edges are sorted and a color is calculated for each pixel – pixels are touched only once
- Presentation
  - After the main rasterizer is done painting, the memory buffer is copied to the screen

- Some SObjects are converted to bitmaps at runtime
  - Each time a children is invalidated the cached bitmap is invalidated
  - Scaling or rotating the SObject will also invalidate the cache
    - use `cacheAsBitmapMatrix` to prevent that
    - `Matrix3D` can be used to avoid invalidating the cache when applying transforms
- Cases that create surfaces on the fly
  - Bitmap Caching properties
  - Filters & Pixel Bender

# Example



- Two rasterizations
  - Root
  - Background – happens only once if we use static content

- Multiple ways to display the rendered memory buffer on screen
- Flash uses “wmode” embed parameter to choose the presentation mode
  - Transparent, opaque- the browser takes the memory buffer and paints it in its own rendering stack, it requires rendering all the elements behind Flash and all above it
  - Window - the browser allocates a native window for Flash. Invalidates are handled directly by Flash using CPU copy (ex. on Windows, GDI is used, so it might actually be accelerated)
  - Direct - same as window, but copy on screen is done using DirectX or OpenGL, StageVideo renders directly to a texture and rendered behind the Stage
  - GPU - enabled only on mobile devices - tessellates shapes into meshes that are rendered in GPU, PixelBender not working yet
- AIR Mobile `initialWindow.renderMode` tag
  - CPU
  - GPU

# AIR Rendering on Devices

- iPhone
  - GPU blend
    - Only surface composition is done in hardware
    - Will only make sense in tandem with bitmap caching
- Android
  - GPU Vector
    - Everything is rendered in hardware
      - Shapes are converted to triangles
      - Filters do not work in 2.5 – will be converted to shaders in future version
- Might be slower than CPU in some cases
  - Scaling and morphing shapes will require tessellation, a slow process that executes in CPU, use bitmap caching in these cases
  - Prevent invalidations inside SObjects by placing moveable parts in different trees (eg. background object should not contain the foreground elements)

# Bitmap Caching

- Automatic
  - Easier to use
  - Might be invalidated more often than needed
- Manual
  - Use `BitmapData.draw` to cache `DisplayObjects` as bitmaps
  - Useful for animations, pre-render frames to different bitmaps
- GPU does a great job at composition bitmaps, so it will always be faster
- CPU might actually be slower in some cases when using bitmap caching
  - Rendering a solid circle in CPU is faster than rendering a transparent bitmap
  - Bitmap caching applied on bitmaps doesn't make sense

# Single Thread execution

- Most of Flash executes only on the main thread
  - Some exceptions are
    - Sockets
    - Asynchronous AS3 APIs
    - Rasterizer & Filters use all the cores to accelerate rendering
    - Pixel Benders are compiled to assembly code and run on all cores
- ActionScript execute on the same thread as UI rendering
  - So, no ActionScript on the stack when rendering is in progress
    - except `BitmapData.draw`, but rendering doesn't call back to ActionScript and AS waits the renderer to complete the operation
    - `EventDispatcher.dispatchEvent` will execute immediately and will not allow rendering intermediary frames, it is not a yield command

# Mouse Events and Stage framerate

- Frame rate will not affect the frequency of the MouseEvents
  - Flash will deliver as much events as possible to increase resolution when drawing for example
  - It will only render changes at the following frame
- Consequences
  - Display object tree might be updated much more than actually needed
  - ActionScript will execute for each event but will render just once
- Solutions
  - “EnterFrame” will dispatch just before rendering, but if player notices it’s a little behind and it has to skip frames, will execute multiple times without actually rendering on screen
  - “Render” will only dispatch once before rendering

# Network Requests

- Sockets, RTMP, RTMFP are handled directly by Flash using platform socket APIs
- In Flash URL requests are handled using the browser stack
  - Some browsers limit the number of simultaneous loads
- In AIR URL Requests are handled by the platform APIs
  - Mac: Cocoa
  - Windows: WinINet (same library used by Internet Explorer)
  - Linux, Mobile: cURL



# Links

- <http://tv.adobe.com>
- <http://tv.adobe.com/watch/max-2010-develop/deep-dive-into-flash-player-rendering>
- [http://www.adobe.com/devnet/flashplayer/articles/fplayer10\\_1\\_hardware\\_acceleration.html](http://www.adobe.com/devnet/flashplayer/articles/fplayer10_1_hardware_acceleration.html)
- [achicu@adobe.com](mailto:achicu@adobe.com)



**Adobe**