GPU Computing:
A VFX Plugin Developer's Perspective

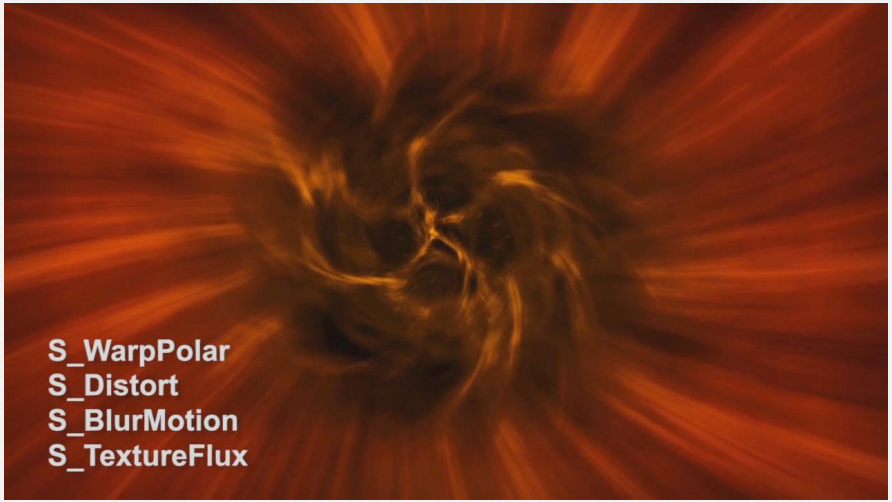
Stephen Bash, GenArts Inc.

GPU Technology Conference, March 19, 2015

GENARTS

GenArts Sapphire Plugins

- Sapphire launched in 1996 for Flame on IRIX, now works with over 20 digital video packages on Windows, Mac, and Linux
- Award winning collection of over 250 effects
- Effects composed from library of hundreds of algorithms: blur, warp, FFT, lens flare, …
  - Algorithms implemented in both C++ and CUDA
  - *… and both must produce visually **identical** results*



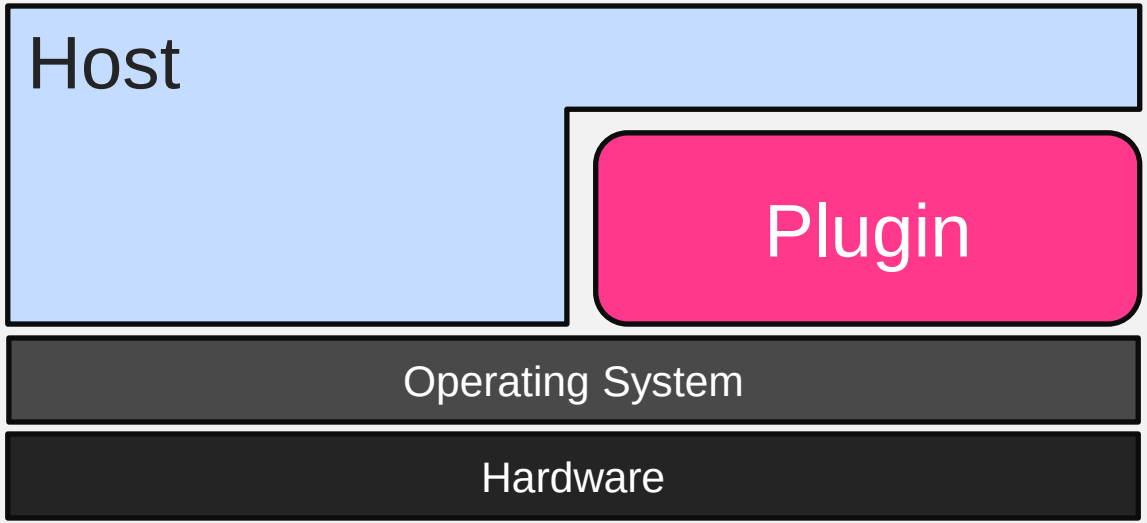
Outline

- Introduction
  - What's a plugin?
  - Why CUDA?
- CUDA programming for plugins
  - What works…
  - … and what doesn't
- Tips and tricks for living in someone else's process
  - Context management
  - Direct GPU transfer
  - Library linking
- Summary

Introduction

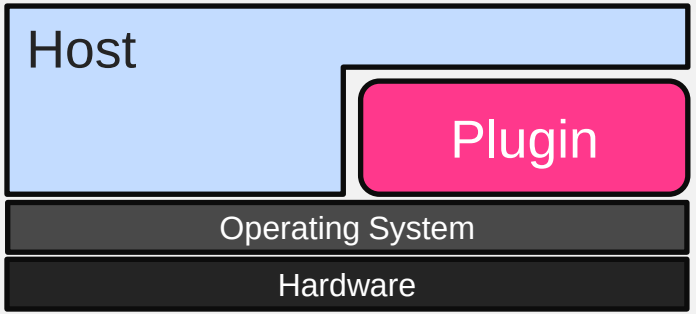
What's a plugin?

- Shared library / DLL / loadable bundle
- API specified by host (program loading the plugin)
- Creates opportunity for third party to add features and value to host



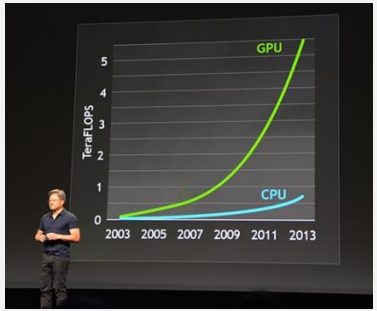
How are plugins different?

- Plugin shares host's process and resources
- Plugin errors can affect host
- Plugin may need to be reentrant and thread safe
  - Lock discipline extremely important
  - Requires careful memory management
- Plugin usually dependent on host for persistence
- Plugin must accept/support the host's system requirements



Why CUDA? Performance!

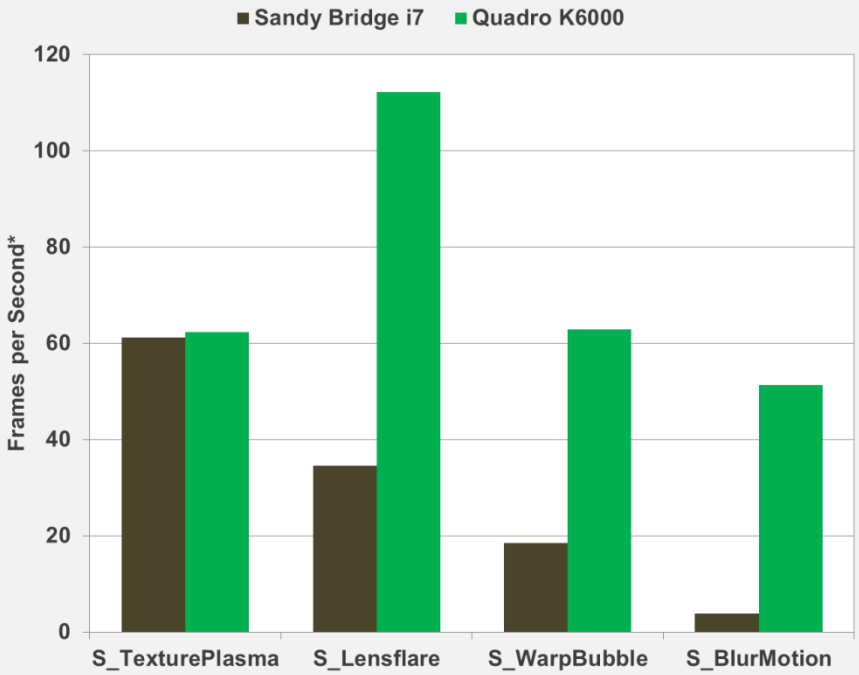
- VFX artists require high quality renders with interactive performance
  - Visual artist's efficiency depends on ***seeing*** the result quickly
- VFX projects are getting bigger
  - DVD 480p = 119 MB/sec
  - HD 1080p = 746 MB/sec
  - The Hobbit 5k stereo = **16.6 GB/sec!**
- Interesting effects are complex
  - Lens flares with hundreds of elements
  - Automated skin detection and touch up
  - Complex warps with motion blur
  - Footage retiming
- CUDA enables interactive effects via powerful GPUs



CUDA for VFX Plugins

CUDA for Plugins: The Good

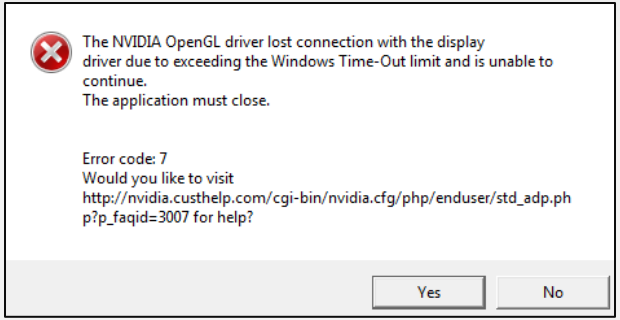
- CUDA provides significant speed gains for our effects
- CUDA is OS-independent
- Cost effective performance for customers
  - Cheaper and easier to upgrade GPU
- Hosts are beginning to support direct GPU transfer of images



** Plugin only performance rendering 1080p*

CUDA for Plugins: The Bad

- Long running kernels cause Windows to reset driver
  - Reset can break/crash host
- NVidia cards are scarce in Macs
- GPU sharing with host is relatively undocumented
  - Many hosts monopolize GPU resources
- Host APIs lack tools to coordinate over multiple GPUs

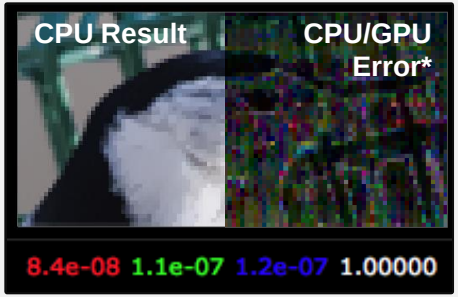


CUDA for Plugins: When Things Go Wrong

- Provide CPU fallback for all effects
  - A single black frame can ruin a long project
  - Also allows heterogeneous render farms
- Implementations can differ, but results have to visually match
- Test infrastructure keeps us honest

- Example: S_EdgeAwareBlur
  - Preprocessor stores result differently on CPU and GPU
  - Three different blur implementations
  - Final results are not numerically identical, but are visually indistinguishable

```
// Try to execute on GPU
bool render_cpu = true;
if (supports_cuda(gpu_index)) {
  if (execute_effect_internal(gpu=true, ...))
    render_cpu = false; // GPU render succeeded
}

// Execute on CPU
// If GPU render failed, this will retry on CPU
if (render_cpu)
  execute_effect_internal(gpu=false, ...);
```

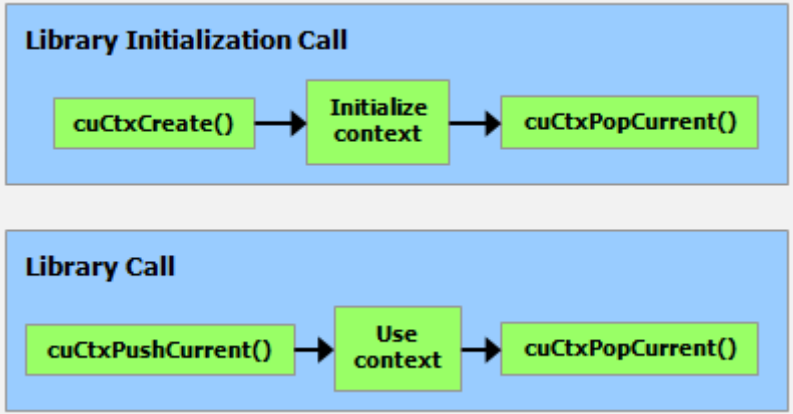


** Color enhanced to show detail*

Tips and Tricks

CUDA Context Management

- Host might use CUDA
  - Need to isolate plugin errors (e.g. unspecified launch failure) from host
- CUDA contexts are analogous to CPU processes and isolate memory allocations, kernel invocations, device errors, and more
- Plugin can use the driver API to create its own context and perform all operations in that private context



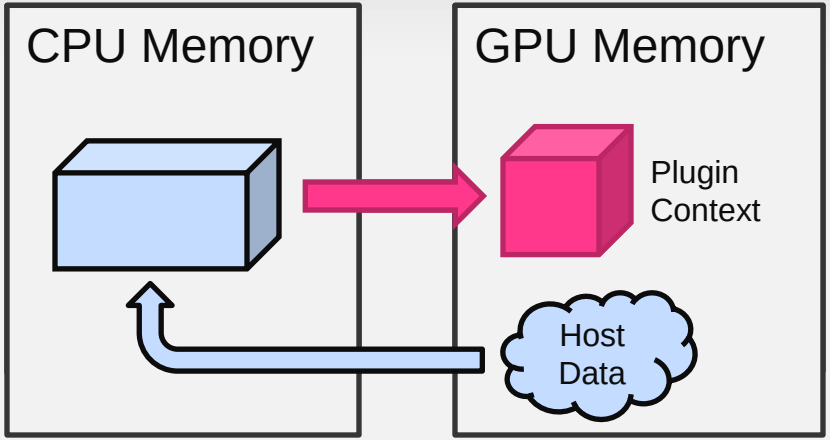
Library context management
CUDA 6.5 Programming Guide, Appendix H

CUDA Context Management

- Requires use of driver API
- To support running on machines with different driver versions, load driver at runtime rather than linking it directly
  - On Mac weak link the CUDA framework
- If an error occurs, destroying context will free plugin's GPU memory and reset device to non-error state

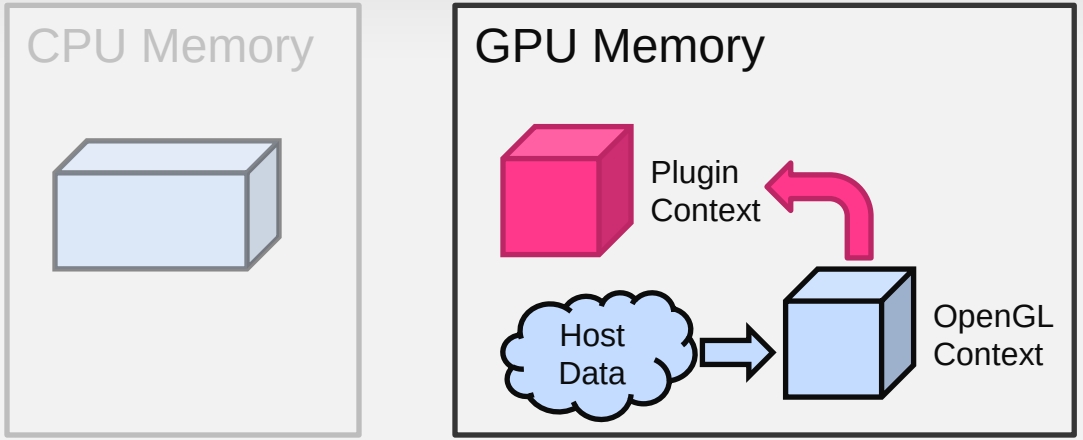
```
// Persistent state
static CUcontext cuda_context = NULL;
static CUdevice cuda_device = -1;  // initialized elsewhere

CudaContext::CudaContext(bool use_gl_context) {
  if (!cuda_context) {  // Create new context
    if (use_gl_context)
      cuGLCtxCreate(&cuda_context, 0, cuda_device);
    else
      cuCtxCreate(&cuda_context, 0, cuda_device);
  }

  cuCtxPushCurrent(cuda_context);
}

CudaContext::~CudaContext() {
  cuCtxPopCurrent(NULL);
}
```

Direct GPU transfer



- Naive GPU-accelerated host copies data back to CPU memory for plugin

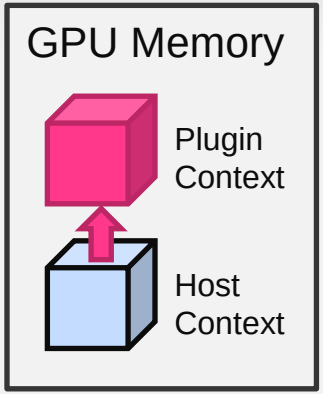
Direct GPU transfer



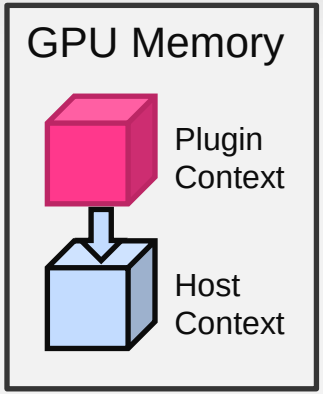
- Naive GPU-accelerated host copies data back to CPU memory for plugin
- OpenGL is the cross-platform solution for sharing between multiple GPU languages
  - May require extra memory copies if host isn't natively OpenGL
  - OpenGL/CUDA interop on Mac is really slow

Direct GPU transfer: CUDA to CUDA

- Multiple options for transferring data when both host and plugin use CUDA:
  - cuMemcpyPeer (driver API)
  - cudaMemcpy (runtime API)
  - Custom kernel
- Still exploring CUDA/CUDA transfers with hosts



Host to Plugin



Plugin to Host

Host to Plugin Bandwidth (GB/s)

	cuMemcpyPeer	cudaMemCpy	Kernel
Windows	3.21	4.05	X
Mac	2.27	2.19	54.57
Linux	4.55	4.53	55.75

Plugin to Host Bandwidth (GB/s)

	cuMemcpyPeer	cudaMemCpy	Kernel
Windows	59.93	59.82	53.98
Mac	60.47	60.93	54.48
Linux	63.05	63.05	55.79

Results from Quadro K5000

Linking and Loading

- Running in host's process means dynamic loader sees host's dependent libraries before plugin
  - Plugin may get a different version of library or symbol than it expects
- Library/symbol conflicts manifest in many (usually strange) ways

- On Windows: use (private) side-by-side assemblies to get the correct library
- On Mac and Linux: statically link CUDA runtime (as of CUDA 5.5)
  - To avoid conflicts you *must* instruct `ld` to hide resolved global symbols and strip the final result
    - Mac: See `ld -exported_symbols_list` and `-unexported_symbols_list` (only one is necessary)
    - Linux: See linker scripts (http://stackoverflow.com/a/452955)

- CUFFT and CUBLAS can be statically linked as of CUDA 6.5
  - Device link required to statically link CUFFT
  - `nvcc -dlink` or `nvlink` takes any number of static libraries/object files and produces a single object file to include in the final traditional link

Summary

Summary

- CUDA has a lot of benefits for plugin developers
- As a plugin or host developer, think about resource sharing with the other
  - Context management
  - Direct GPU transfers
  - Library loading (or static linking)
  - Error handling and communication

- Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!

Stephen Bash

stephen@genarts.com

Questions (and eye candy)