Safe Harbor

We may make statements regarding planned or future development efforts for our existing or new products and services. These statements are not intended to be a promise or guarantee of future availability of products, services or features but merely reflect our current plans and based on factors currently known to us. These planned and future development efforts may change without notice. Purchasing decisions should not be made based upon reliance on these statements.
Autodesk serves three major industries

ARCHITECTURE, ENGINEERING & CONSTRUCTION

PRODUCT DESIGN & MANUFACTURING

MEDIA & ENTERTAINMENT

Image courtesy of Axis Studios
Who am I?

Jerran Schmidt – Sr. Principal Engineer

- OGS – One Graphics System
  - The common graphics system for nearly all major Autodesk products

- Fusion

- Simulation
Talk outline – Vulkan in Fusion Simulation

- Challenges for Simulation Post Processing
- Solution Evolution
- Examples
- Vulkan Compute Engine
- Portability
Simulation in Fusion
Simulation Workflow

- Setup
  - Assign Boundary Conditions
  - Assign Materials

- Solve
  - Produce Results

- Post-Processing
  - Derive Insights
Challenges for Simulation Post-Processing

- Cloud solve
  - CPU/GPU various toolkits including CUDA and OpenCL
  - Long running solves
    - Generating lots of data
  - Large meshes
    - Variation of mesh topology

- Cloud data storage
  - Large datasets
    - Large number of physics quantities calculated at each point in the mesh for each timestep
  - Generative/DoE

- Insights derived from large amounts of data
  - Visualizations themselves are relatively small
Examples

- Variety of mesh topologies
  - Uniform voxel grids
  - Non-uniform voxel grids
  - Tetrahedral meshes
  - Hexahedral meshes

- Compute operations
  - Isosurfaces
  - Slicing
  - Flow Lines
  - Down-Sampling
  - Voxelization
  - Normalization
  - Timeseries Probe
Performance

- Operations are typically real-time
- Need to be an immediate response to user actions
- Server-side resolution scaling of result datasets helps for low-bandwidth/low-power cases
- Dual-mode
  - Low resolution preview
  - High resolution result
Solution Evolution

- *OpenGL visualization*
- CPU compute feeding OpenGL visualization
- GPU compute in the visualization pipeline via OpenGL Geometry Shaders
- GPU compute feeding visualization via OpenGL Compute Shaders
  - Support of OpenGL Compute Shaders on Mac?
Re-thinking the Solution: Flexible, Scalable, Portable

- Balancing client requirements
  - Bandwidth
  - Latency
  - Local storage
  - Compute capability

- Options for both cloud and local Compute to provide visualization data

- Rendering on the Client
Vulkan Compute Engine

- Cross Platform
- Run headless on the cloud; visualization via any API on the client
- Run locally as part of a Vulkan visualization pipeline on the client
- Readback Output SSBOs and transfer data to renderer or pass the SSBO handle to the Vulkan renderer
Author in GLSL

- "Operations" authored as GLSL compute shaders

- Compute Engine
  - Connects data to input attributes*
  - Sets uniform parameters
  - Executes shader to produce results

*not actually input attributes, rather just SSBOs
Driving Compute – Parameters

- SPIRV-Cross

- Reflection
  - Generate descriptor sets `ShaderResources::`
    - `uniform_buffers`
    - `storage_buffers`
    - `push_constant_buffers`
  - We only use combined image samplers so `sampled_images` suffices

- Build UBOs and Push Constant buffers
  - Create CPU-side structures mapping uniform variables to their location within the uniform buffer

```cpp
struct UBOVar{
    std::string name;
    std::shared_ptr<VulkanBuffer> buffer;
    VkDeviceSize offset;
    VkDeviceSize size;
    bool dirty;
};
```
Driving Compute - Input Attributes

- Use reflection to validate inputs

- Input (and Output) attributes are Storage Buffers
  - ShaderResources::storage_buffers
  - input/output prefix naming convention

- Inputs of varying arity & type; for example
  - Cauchy Stress, etc – mat3
  - Velocity, Displacement, etc – vec3
  - VM Stress, Time, Pressure, etc – float
  - Subassembly Component Id, etc – uint

- Separate shaders for different input arity & type
  - Just #ifdefs at shader generation time
  - Offline compile to SPIR-V
CPU Fallback – Leverage SPIR

- Author in GLSL
- Compile to SPIR-V
- SPIRV-Cross compile to ISPC
  - [https://github.com/GameTechDev/SPIRV-Cross](https://github.com/GameTechDev/SPIRV-Cross)
- Execute SPIR-V on GPU via Vulkan and compile ISPC for execution on CPU
What about texture samplers? All in on Vulkan!

- Author in GLSL
- Compile to SPIR-V
- Same code path for CPU and GPU execution thanks to Swiftshader
- Leverage graphics features in compute
- Unified on Vulkan
Debugging

- Vulkan Validation Layers!
- RenderDoc & Nsight
  - Need swapchain
  - Most common issue is uniform structure misalignment
- VK_EXT_custom_border_color
  - Having known-invalid values for sampling results textures when mapping to geometry
- Profiling
  - PSO Cache
    - Reflection isn’t currently too costly as shaders are mostly simple but improvements could be made via PSO cache
In Fusion

Electronics Cooling

Injection Molding
Platform Portability

- Unified Code Base
  - MoltenVK for Mac

- x86 & ARM
  - LunarG SDK for x86
  - Proven source-level portability
    - Built for Mac/Linux ARM
      - Vulkan Headers
      - SPIRV-Cross
      - glslang
    - LunarG SDK now has M1 support

MaterialX + Vulkan on ARM