GRAMPS: A Programming Model
For Graphics Pipelines
Jeremy Sugerman,
Kayvon Fatahalian, Solomon Boulos,
Kurt Akeley, Pat Hanrahan
Central to the rise of 3D hardware and software.

A stable and universal abstraction

Shaped the evolution of the field…

… while leaving enormous room for innovation.
The Graphics Pipeline is evolving

Direct3D 11 ųle Shading

[Diagram showing the graphics pipeline with components such as Input Assembler, Vertex Shader, Hull Shader, Tessellator, and Output Merger, along with additional components like Domain Shader, Geometry Shader, and Stream Output.]
“GPU” is evolving, too

- Continued drive for algorithmic innovation and advanced rendering techniques
- First class programming models for compute:
  - OpenCL, compute shaders, vendor specific, …
- New / different hardware implementations:
  - E.g., Larrabee, CPU-GPU combinations / hybrids
  - Even NVIDIA and AMD GPUs are very different
From fixed to programmable (again)

Idea: Evolve the pipeline itself from preset configurations to a programmable entity
GRAMPS

- Programming model and run-time for parallel hardware
- **Graphs** of stages and queues
- GRAMPS handles scheduling, parallelism, data-flow

Example: Simple GRAMPS Graph
The Graphics Pipeline becomes an app!

- Structure/setup is (application) software
  - Customized or completely novel renderers
- Reuses current hardware: FIFOs, shader cores, rast, ...
- Analogous to the transition to programmable shading
  - Proliferation of new use cases and parameters
  - Not (unthinkably) radical
Writing a GRAMPS application

Design the execution graph:

- Input
- Vertex
- Rast
- Pixel
- Merge

Design the stages:
- Shaders
- Threads (and Fixed Function stages)

Instantiate and launch.
More Detail – Queues

- Queues operate at a “packet” granularity
  - “Large bundles of coherent work”

- GRAMPS can optionally enforce ordering
  - Required for some workloads, adds overhead
### More Detail – Shaders

- **Shaders**: Like pixel (or compute) shaders, stateless
  - Automatic instancing, pre-reserve/post-commit
- “Collection” packets: shared header and N elements
- **New**: “Push” operation to coalesce variable outputs
- Threads: Like POSIX threads, stateful
  - Explicit reserve/commit on queues
- Fixed Function: Effectively non-programmable Threads
Queue sets enable binning-style algorithms

One logical queue with multiple lanes (or bins)
  - One consumer at a time **per lane**
  - Many lanes with data allows many parallel consumers
Quick Comparison to “Streaming”

- Streaming: “squeeze out every FLOP”
  - Goals: throughput, bulk transfer, arithmetic intensity
  - Intensive static analysis, program transformation
  - Bound space, data access, execution time

- GRAMPS: “interesting applications are irregular”
  - Goals: throughput, dynamic, data-dependent code
  - Aggregate work at run-time, heterogeneous hardware
  - Streaming apps are GRAMPS apps
Evaluation: Design Goals

- Broad application scope: preferable to roll-your-own
- Multi-platform: suits many hardware configurations
- High performance: competitive with roll-your-own
- Tunable: expert users can optimize their apps
- Optimized Implementations: inform, and are informed by, hardware
Broad Application Scope

Rasterization Pipeline (with ray tracing extension)

Ray Tracing Graph

Tiler → Sampler → Camera → Intersect → Shade → Shadow Intersect → Blend

- Thread
- Queue
- Shader
- Stage Output
- Fixed-Func
- Push Output
Multi-Platform: Two (Simulated) Machines

**CPU-Like**: 8 Fat Cores, Rast

**GPU-Like**: 1 Fat Core, 4 Micro Cores, Rast, Sched
Priority #1: Show scale out parallelism
   – Can GRAMPS exploit the application parallelism and fill the machine?

Priority #2: Show ‘reasonable’ bandwidth / storage requirements for queueing
   – What is the worst case total footprint of all queues?
   – A scheduling problem: trade-off with possible parallelism
Very simple static prototype scheduler (both platforms):

- Static stage priorities:
  - Limited pre-emption points
  - **No** dynamic weighting of current queue depths
Three scenes x { Rasterization, Ray Tracer, Hybrid }
Parallelism is 95+% for all but rasterized fairy (~80%).
Queues are small: < 600KB CPU-like, < 1.5MB GPU-like
Order costs footprint
Tunability – Understanding Performance

- GRAMPSviz:

- Also: raw counters, statistics, text log of run-time activity
Tunability – Lessons Learned

- Execution Graph topology / design:
  - Sort-Middle
  - Sort-Last

- Sizing critical queues:
Summary

- After a long era of stability, the Graphics Pipeline is undergoing rapid change.

- GRAMPS enables software-defined custom pipelines.
  - The Graphics Pipeline becomes an app
  - Prototypes show plausible performance, resource needs
  - Handles heterogeneous parallelism well
  - Applicable beyond rendering and beyond GPUs
Thank You

- Our funding agencies:
  - Stanford Pervasive Parallelism Lab
  - Department of the Army Research
  - Intel Corporation
  - Rambus Stanford Graduate Fellowship
  - Intel PhD Fellowship
  - NSF Graduate Research Fellowship