Further Developing GRAMPS

Jeremy Sugerman
FLASHG
January 27, 2009

Introduction

- Evaluation of what/where GRAMPS is today
- Planned next steps
  - New graphs: MapReduce and Cloth Sim
- Speculative potpourri, outside interests, issues

Background: Context

Problem Statement:

- Many-core systems are arising in many flavours: homogenous, heterogeneous, programmable cores, and fixed-function units.
- Build a programming model / run-time system / tools that enable efficient development for and usage of these hardware architectures.

Status Quo:

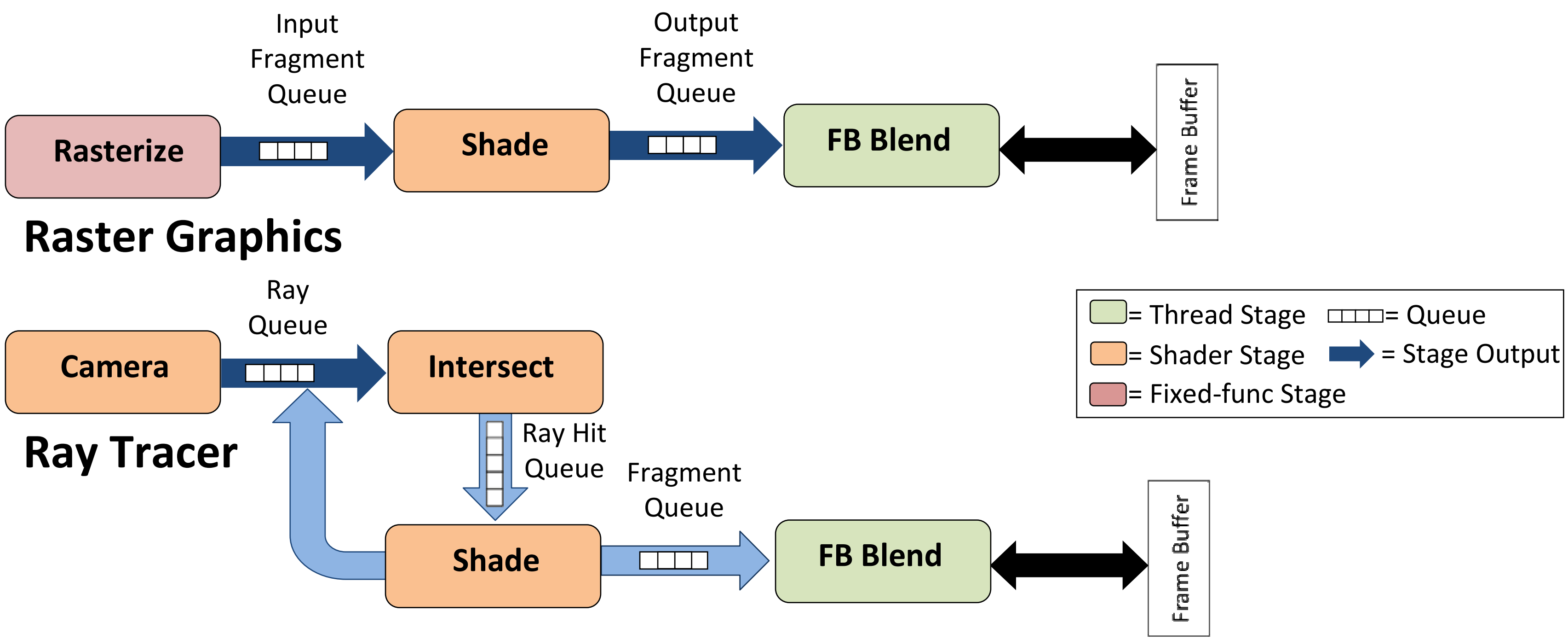
- GPU Pipeline (GPU-only, Good for GL, otherwise hard)
- CPU / C run-time (No programmer guidance, fast is hard)
- GPGPU / CUDA / OpenCL (Good for data-parallel, regular)

Background: History

- *GRAMPS: A Programming Model for Graphics Pipelines*, ACM TOG, January 2009
- Chips combining future CPU and GPU cores
- Renderers for Larrabee and future ‘normal’ GPUs
- Collaborators: Kayvon Fatahalian, Solomon Boulos, Kurt Akeley, Pat Hanrahan

- My current interest: (continue) convincing people a GRAMPS-like organization should inform future app and hardware designs.

GRAMPS 1.0



- Express apps as graphs of stages and queues
- Expose **Producer-consumer** parallelism
- Facilitate task and data-parallelism
- GRAMPS handles inter-stage scheduling, data-flow

Design Goals

- Large Application Scope– Preferable to roll-your-own
- High Performance– Competitive with roll-your-own
- Optimized Implementations– Informs HW design
- Multi-Platform– Suits a variety of many-core systems

Also:

- Tunable– Expert users can optimize their apps

GRAMPS's Role

- Target users: engine, middleware, SDK, etc. systems savvy developers
- Example: A 'graphics pipeline' is now an app!

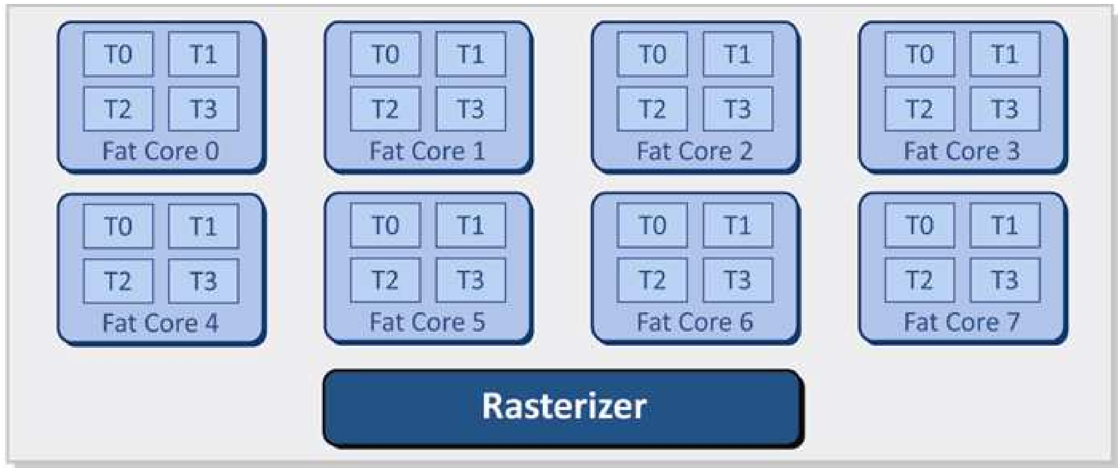
Developer owns:

- Identifying a good separation into stages
- Implementing optimized kernels for each stage

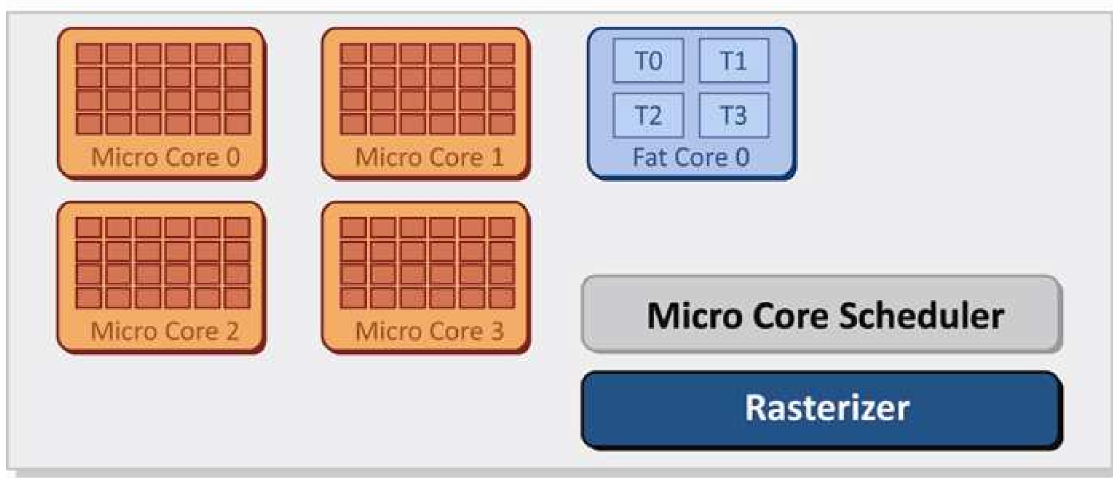
GRAMPS owns:

- Handling all inter-stage interaction (e.g., queues, buffers)
- Filling the machine while controlling working set

What We've Built (System)



CPU-Like: 8 Fat Cores, Rast



GPU-Like: 1 Fat Core, 4 Micro Cores, Rast, Sched

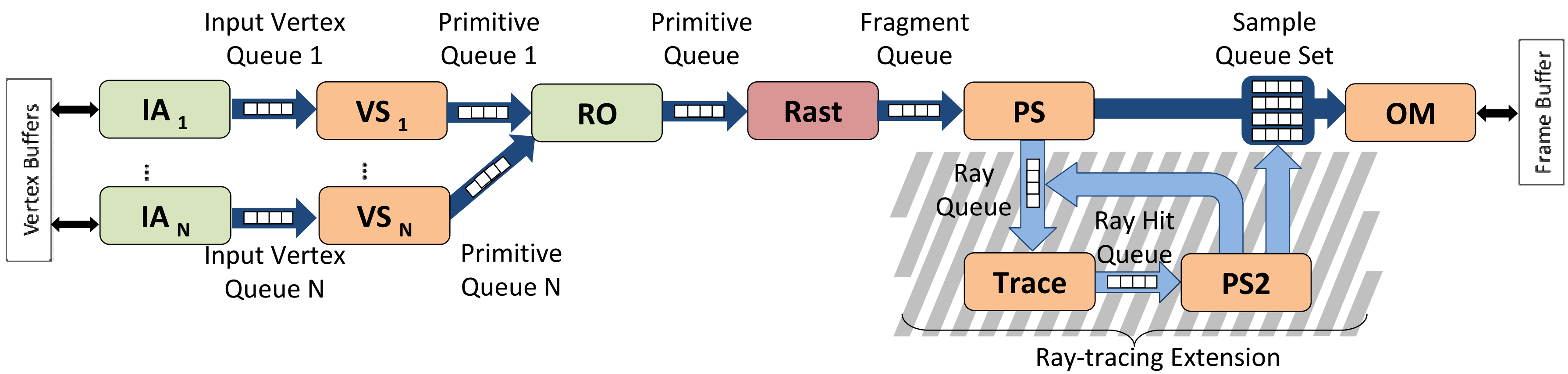
What We've Built (Run-time)

- Setup API; Thread, Shader, Fixed stage environments

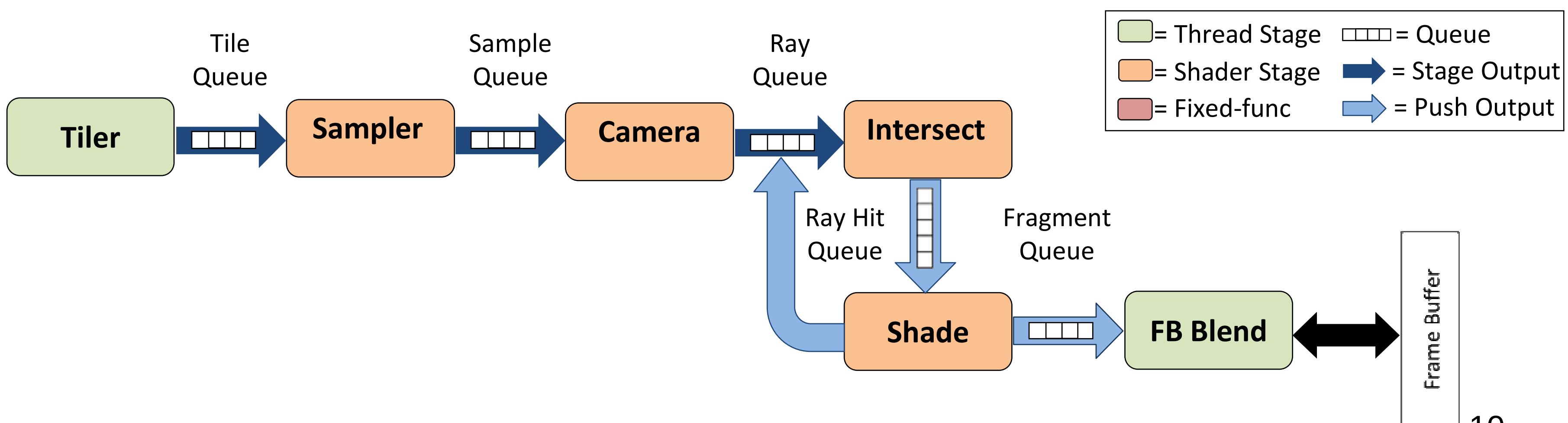
- Basic scheduler driven by static inputs
  - Application graph topology
  - Per-queue packet ('chunk') size
  - Per-queue maximum depth / high-watermark
  - Ignores: online queue depths, execution history
  - Policy: run consumers, pre-empt producers

What We've Built (Apps)

Direct3D Pipeline (with Ray-tracing Extension)



Ray-tracing Graph



Taking Stock: What Did We Learn?

- At a high level, the whole thing seems to work!
  - Nontrivial proof-of-concept apps are expressible
  - Heterogeneity works
  - Performance results do not preclude viability
- Stage scheduling is an arbitrarily hard problem.
- There are many additional details it would help to simulate.
- (Conventional) GPU vendors want much more comprehensive analysis.
- Role of producer-consumer is often overlooked

Digression: Some Kinds of Parallelism

Task (Divide) and Data (Conquer)

- Subdivide algorithm into a DAG (or graph) of kernels.
- Data is long lived, manipulated in-place.
- Kernels are ephemeral and stateless.
- Kernels only get input at entry/creation.

Producer-Consumer (Pipeline) Parallelism

- Data is ephemeral: processed as it is generated.
- Bandwidth or storage costs prohibit accumulation.

Possible Next Steps

- Increase persuasiveness of graphics applications
  - Model texture, buffer bandwidth
  - Sophisticated scheduling
  - Robust overflow / error handling
  - Handle multiple passes / graph state change
  - …
- Follow-up other ideas and known defects
  - Model locality / costs for cross-core migration
  - Prototype on real hardware
  - Demonstrate REYES, non-rendering workloads

Design Goals (Revisited)

- Application Scope: okay– only (multiple) renderers
- High Performance: so-so– only (simple) simulation
- Optimized Implementations: good
- Multi-Platform: good
- (Tunable: good, but that's a separate talk)

➢ **Strategy**: Broad, not deep. Broader applicability means more impact for optimized implementations.

Broader Applicability: New Graphs

- "App" 1: MapReduce
  - Popular parallelism-rich idiom
  - Enables a variety of useful apps

- App 2: Cloth Simulation (Rendering Physics)
  - Inspired by the PhysBAM cloth simulation
  - Demonstrates basic mechanics, collision detection
  - The graph is still very much a work in progress…

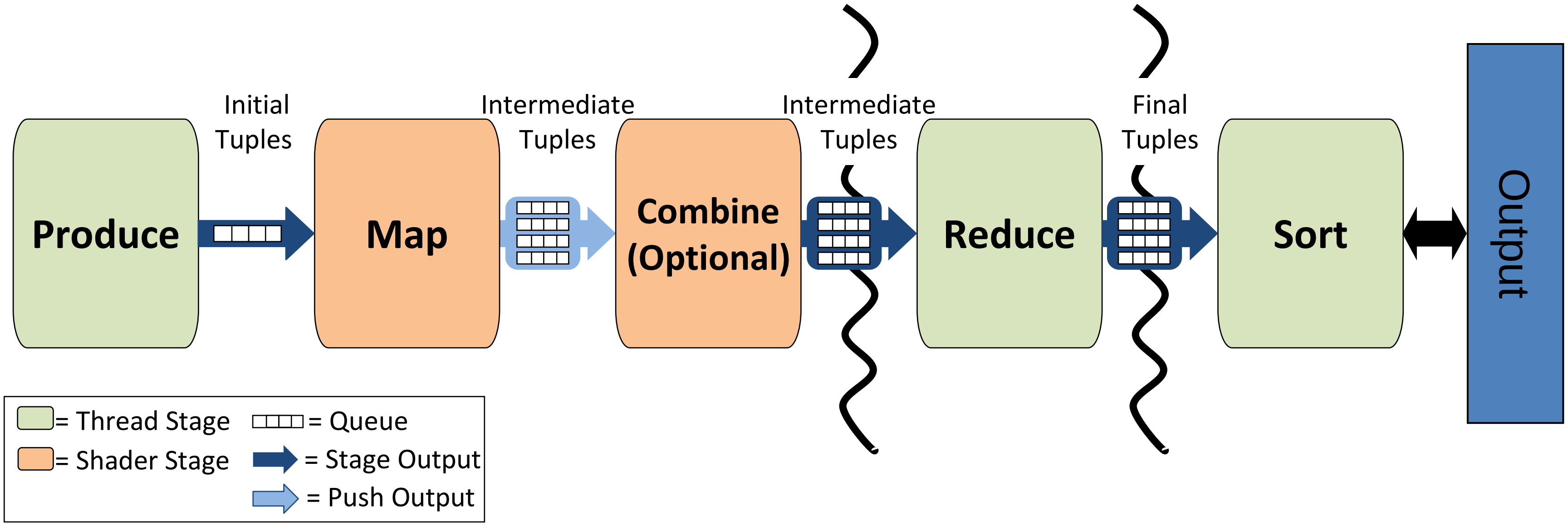
MapReduce Specification

"ProduceReduce": Minimal simplifications / constraints

- Produce/Split (1:N)
- Map (1:N)
- (Optional) Combine (N:1)
- Reduce (N:M, where M << N or M=1 often)
- Sort (N:N conceptually, implementations vary)

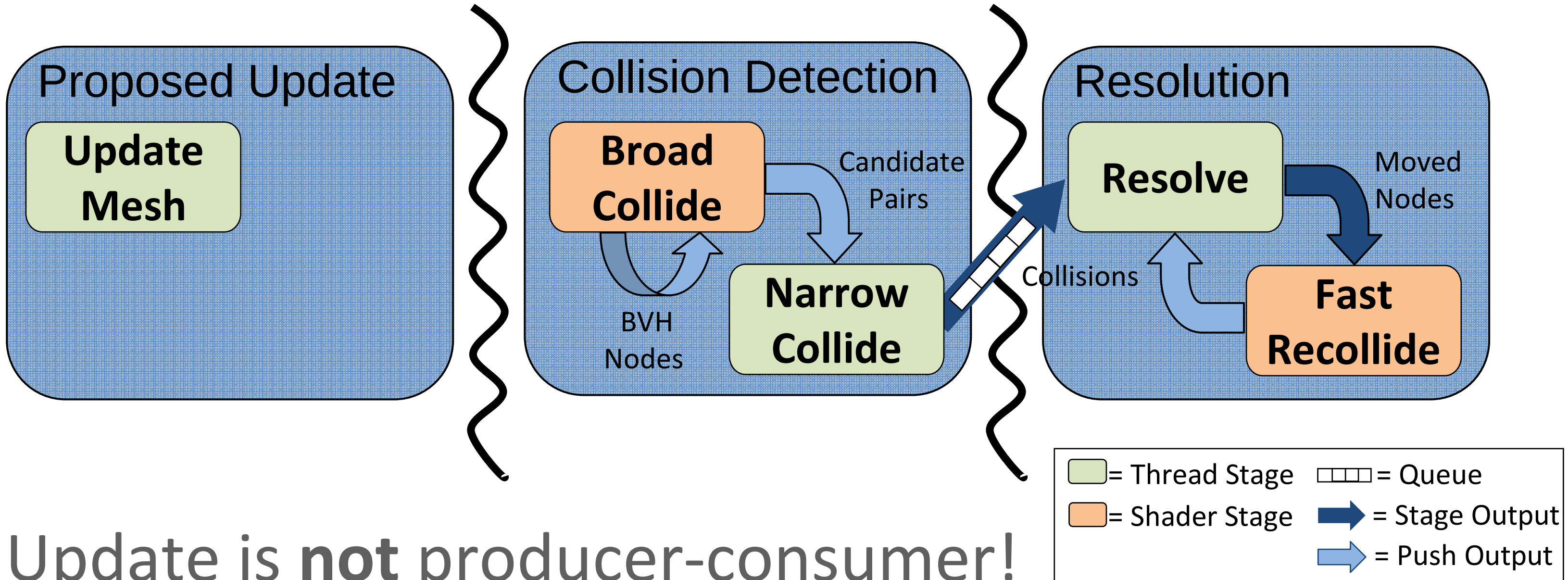
(Aside: REYES is MapReduce, OpenGL is MapCombine)

MapReduce Graph



- Map output is a **dynamically instanced** queue set.
- Combine might motivate a formal **reduction shader**.
- Reduce is an (automatically) **instanced thread** stage.
- Sort may actually be parallelized.

Cloth Simulation Graph



- Update is **not** producer-consumer!
- Broad Phase will actually be either a (weird) shader or multiple thread instances.
- Fast Recollide details are also TBD.

Potpourri Projects

- Dynamic Scheduling– at least current queue depths
- Memory system– more real access times, compute / cap memory bandwidth
- Locality/Scalability (maybe)– validate the overheads of the run-time, model data/core migration costs.
- Standalone GRAMPS– decouple run-time from simulated hardware, perhaps port to real hardware

Outside Interests

- Many PPL efforts are interested in GRAMPS:
  - Example consumer for the OS / Run-time interface research.
  - Example workload for (hardware) scheduling of many-threaded chips.
  - Example implementation of graphics and irregular workloads to challenge Sequoia II.
- Everyone wants to fit it above/below their layer (too many layers!)
- All would profit from Standalone GRAMPS

Overlapping Interests

- REYES is the third major rendering scheme (in addition to OpenGL/Direct3D and ray tracing).
- During GRAMPS 1.0, “Real-time REYES” was always on our minds.

- Forked into the micropolygon pipeline project
  - (Kayvon, Matt, Ed, etc.)
- Expect overlap in discussion and/or implementation as they consider parallelization.

That's All Folks

- Thank you for listening. Any questions?
- Actively interested in collaborators
  - (Future) Owners or experts in some parallel application, engine, toolkit, pipeline, etc.
  - Anyone interested in scheduling or porting to / simulating interesting hardware configurations

- http://graphics.stanford.edu/papers/gramps-tog/
- http://ppl.stanford.edu/internal/display/Projects/GRAMPS