Hierarchical GPU-based Operations for Collision and Distance Queries

Dinesh Manocha

University of North Carolina at Chapel Hill

Collaborators

- Christian Lauterbach (UNC)
- Qi Mo (UNC)
- David Luebke (NVIDIA)
- Michael Garland (NVIDIA)
- Shubhabrata Sengupta (UC Davis)

Motivation

Collision queries

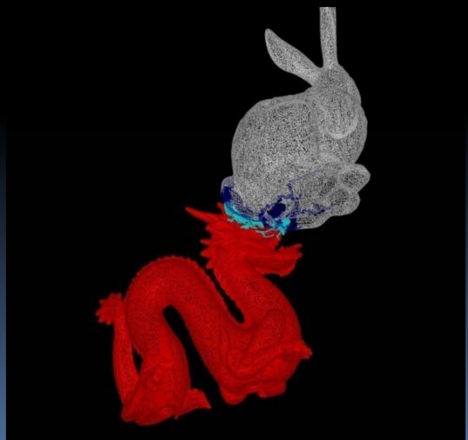
2+ objects

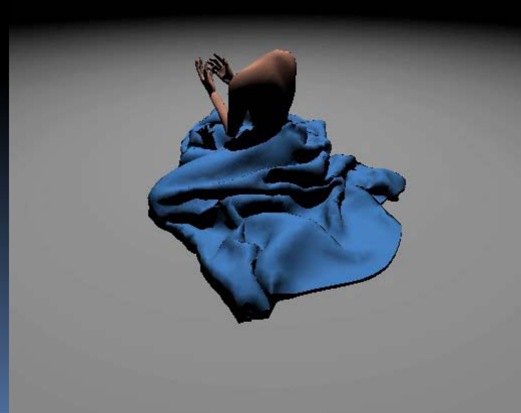
1 object

Intersection

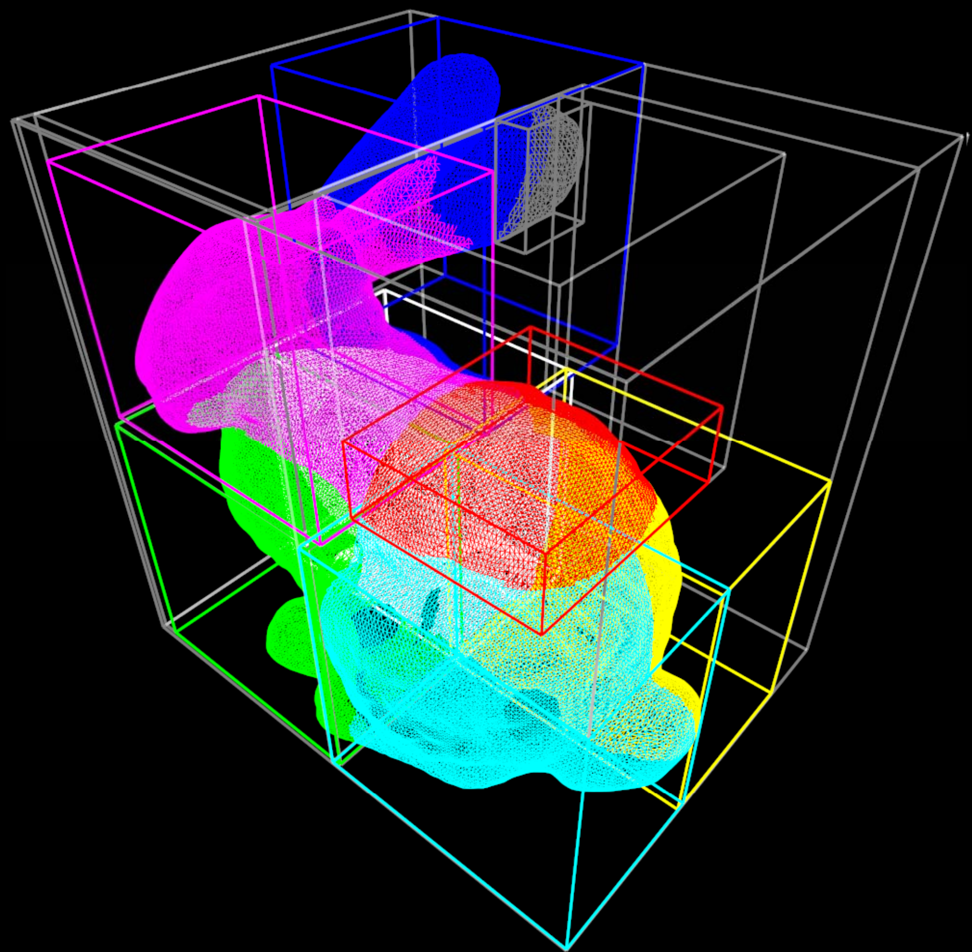
Separation distance

Self-collision





Hierarchies

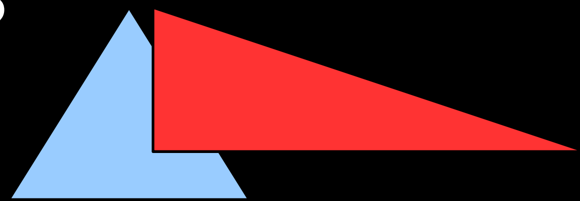


Hierarchy-based proximity queries

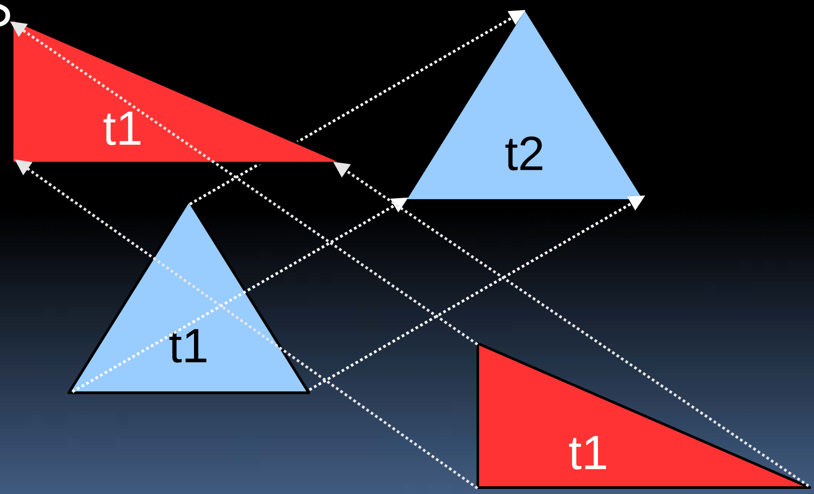
- Build or update hierarchies
- Traverse hierarchies recursively
  - Start with root nodes
  - Do nodes overlap?
    - Yes: Inner nodes: recurse on combinations of children
      Leaf nodes: put primitive pair in separate queue
  - Perform primitive overlap tests

Primitive tests

- Discrete collision: triangle-triangle test
  - Do triangles overlap?



- Continuous collision
  - Did moving triangles overlap at any time between t1 and t2?



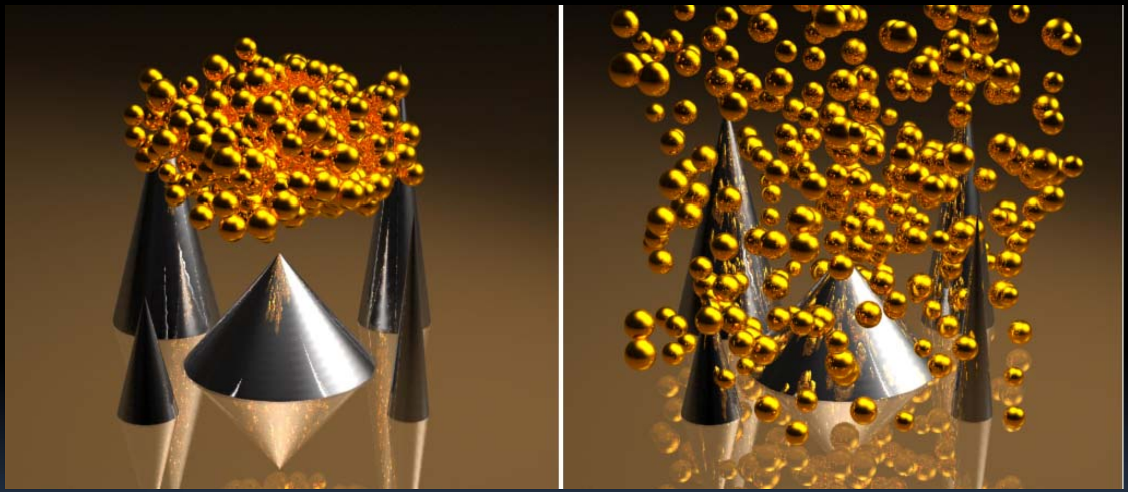
Related work

- Hierarchies for collision
  - AABB trees, sphere-trees, OBBs, k-DOPs,
  - Differences in culling efficiency (avoiding false positives)

- Better culling of primitive tests
  - [Curtis et al. 08, Tang et al. 08, Tang et al.10]
  - Mostly limited to CPU tests
  - http://gamma.cs.unc.edu/SELFCD (SELFCCD system)

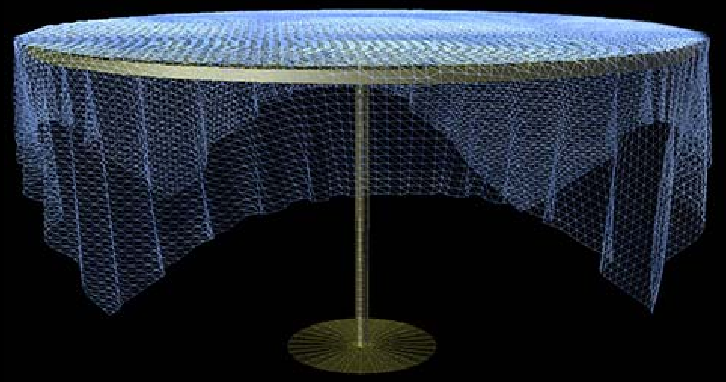
Related work

- Use multi-core CPUs
  - [Kim et al. 08, Kim et al. 09, Tang et al. 09]



Previous work

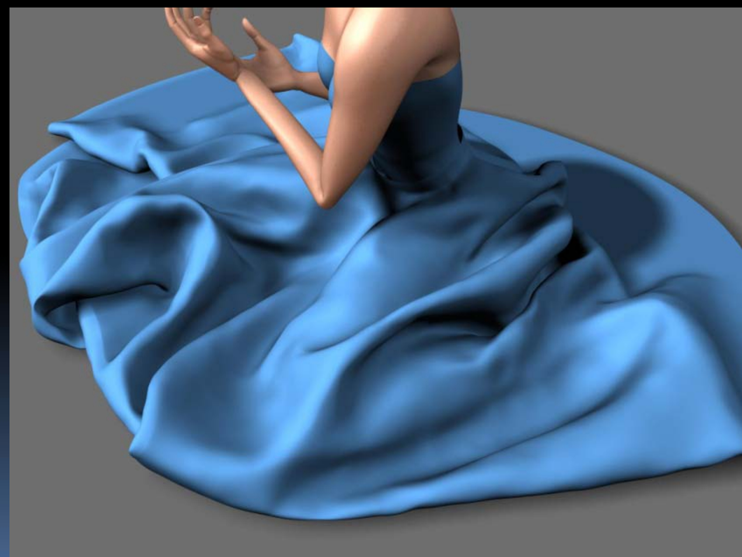
- Image-space GPU-based collision checking



e.g.
[Heidelberger et al. 03, Knott and Pai 03, Govindaraju et al. 03]

- Culling using GPUs

e.g.
[Sud et al. 04, Govindaraju et al. '05, Sud et al. 06, Morvan et al. 08]



Previous work

CULLIDE (2003): First GPU-based system for objects with changing topologies

Perform collision queries at image-space resolution

Prior work using GPUs

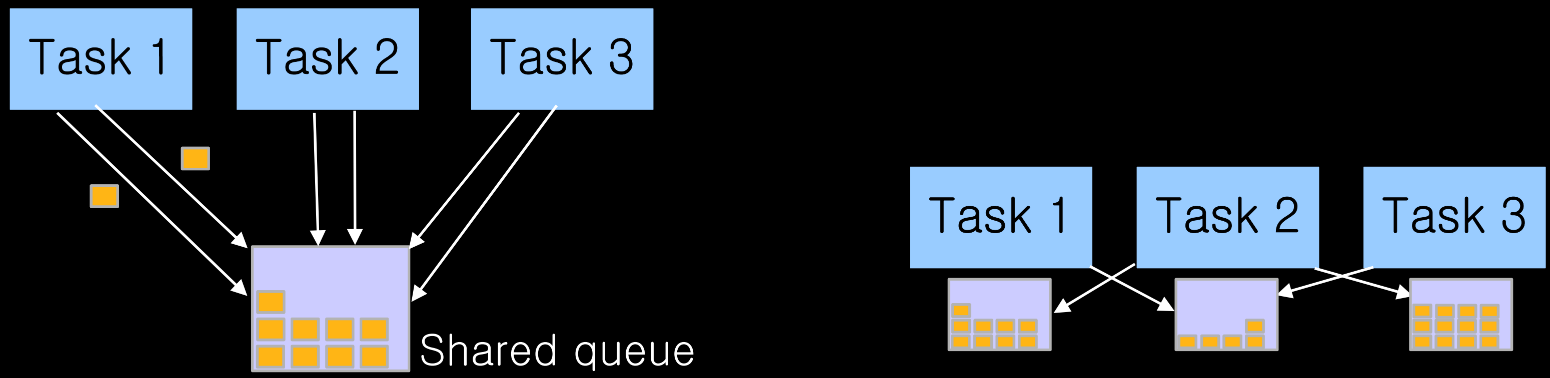
- Used GPUs as a rasterization engine
  - Image-space collision queries
  - Object-space collision queries (conservative rasterization)

Background: GPU architectures

- Moving target, but generally
  - High number of independent cores (16 – 30)
  - Wide vector units on each core (8 – 32)
  - High bandwidth, high latency main memory
- Synchronization between cores
  - Only via main memory
  - No memory consistency model!

Work organization on GPUs

- Standard for recursive hierarchy operations
  - Global work queue, work stealing



- Problem
  - Shared access on GPU only via slow, non-consistent global memory

Lightweight balancing

- Our solution
  - Every thread/core has local queue (non-shared)
  - Keep track of other thread's state occasionally
    - One shared global idle counter
  - If above threshold, break and balance queues

- Avg. ~2-3x performance of work stealing

Parallel Hierarchy Operations

- Can also use vector units
  - Each vector lane handles one intersection pair
  - Potentially thousands of parallel tests
- Local work queue shared between lanes
  - Access synchronized by atomics or prefix sum
  - Does not change outside synchronization

Hierarchy Construction

BVH construction on GPUs

- Uses thread and data parallelism
- Fast linear BVH construction

Interactive construction on current GPUs

Hierarchy Construction

- Top-down methods
  - E.g. recursively split primitives in half
- Bottom-up methods
  - Repeatedly combine primitives into groups
- Derive from scene graph

Hierarchy Construction

Extensive work in interactive ray tracing (RT)

- Mostly used: Surface area heuristic *[Goldsmith and Salmon 87,Havran 00]*
  - Runtime: O(n log n) *[Wald and Havran 06]*
  - Used for kd-trees and BVHs
- Fast approximations via sampling exist *[Hunt et al. 06,Popov et al. 06]*
  - In practice, almost identical RT performance

Parallel Construction

- Parallel kd-tree builds
  - *[Popov et al. 06, Shevtsov et al. 07, Shevtsov et al. 08]*
  - On GPUs: *[Zhou et al. 08]*
- Parallel BVH construction on multi-core CPUs
  - *[Wald et al. 07,Wald 08]*
- Parallel grid construction on multi-core CPUs
  - *[Ize et al. 06]*

Linear BVH construction

How to perform the computation in parallel?

Idea: treat construction as "*sort*"

- Sorting on GPUs has been shown to be fast and parallel (e.g. *[Govindaraju et al. `05]: GPUSort*)

Two problems:

- What to sort on?
- How to get hierarchy from sorted list?

What to sort on

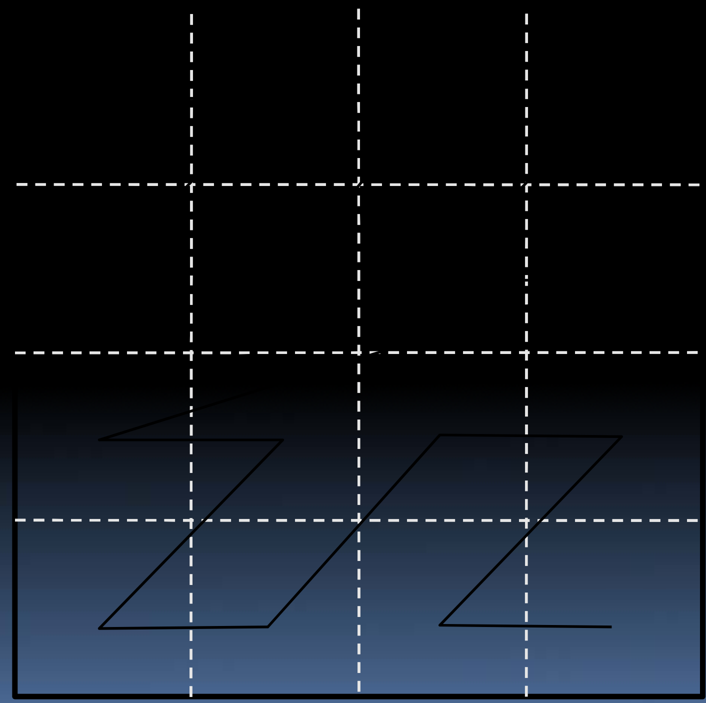
We need linear ordering of objects

Order along 3D space-filling curve

Main property: points close on curve also close in 3-D space

Fast method: Morton/Z-order

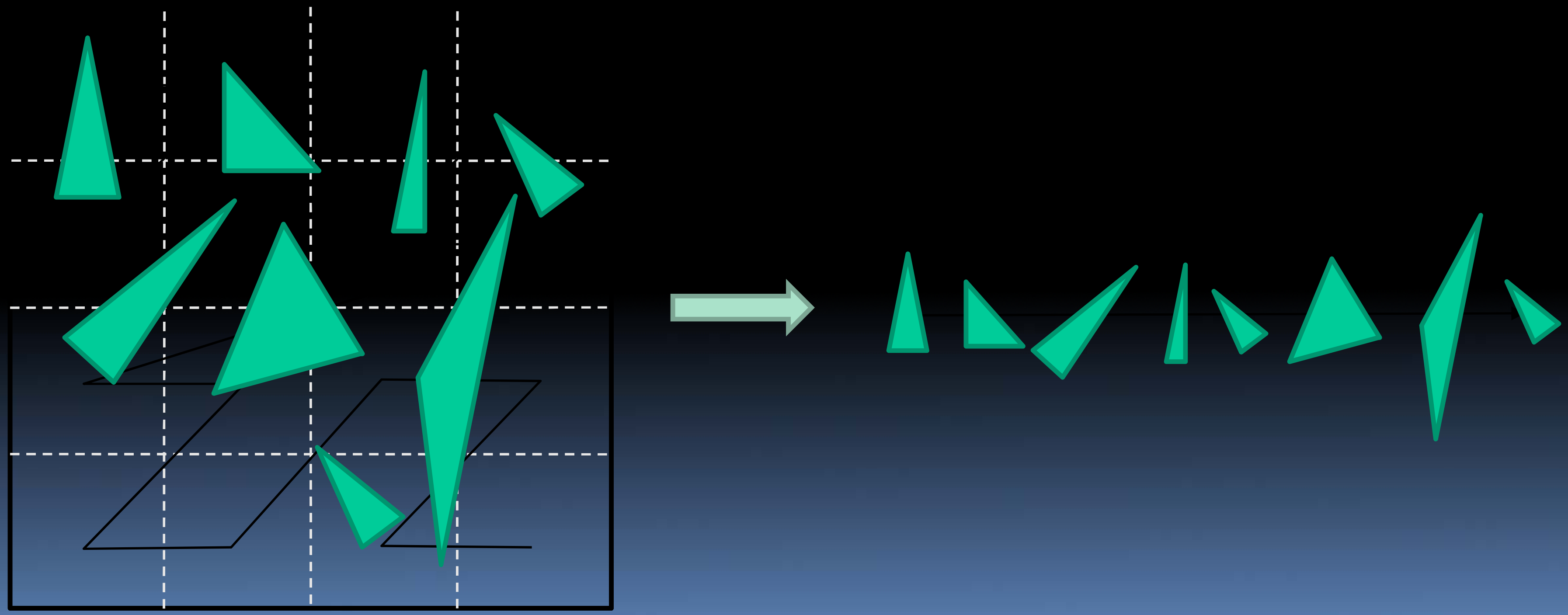
Assign number (Morton/ Z-code) to each primitive



Morton order

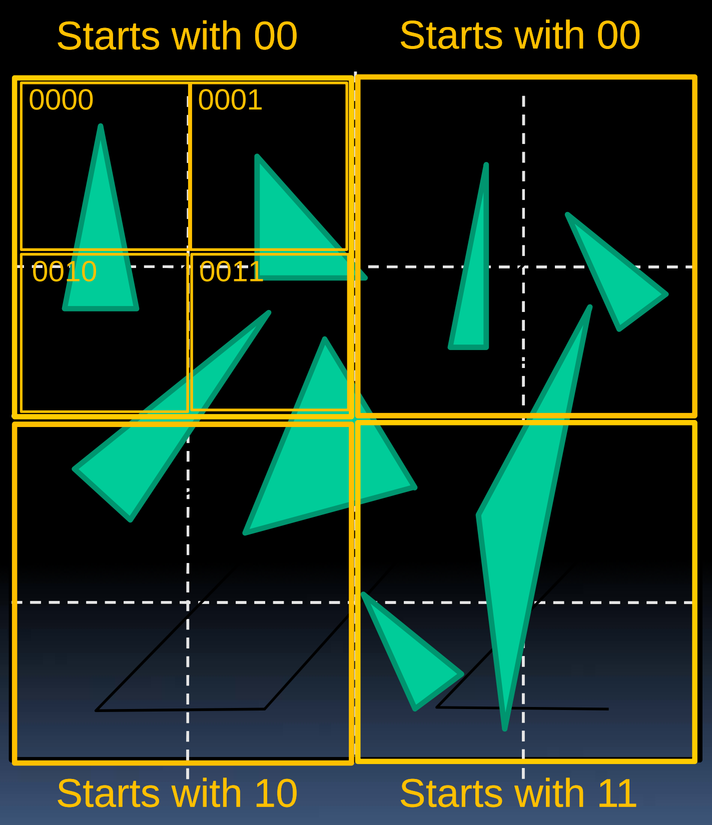
Easy to compute Morton code:

- Quantize coordinates
- Interleave bits from each coordinate
- Sort by Morton code (we use radix sort)



Hierarchy Construction from Order

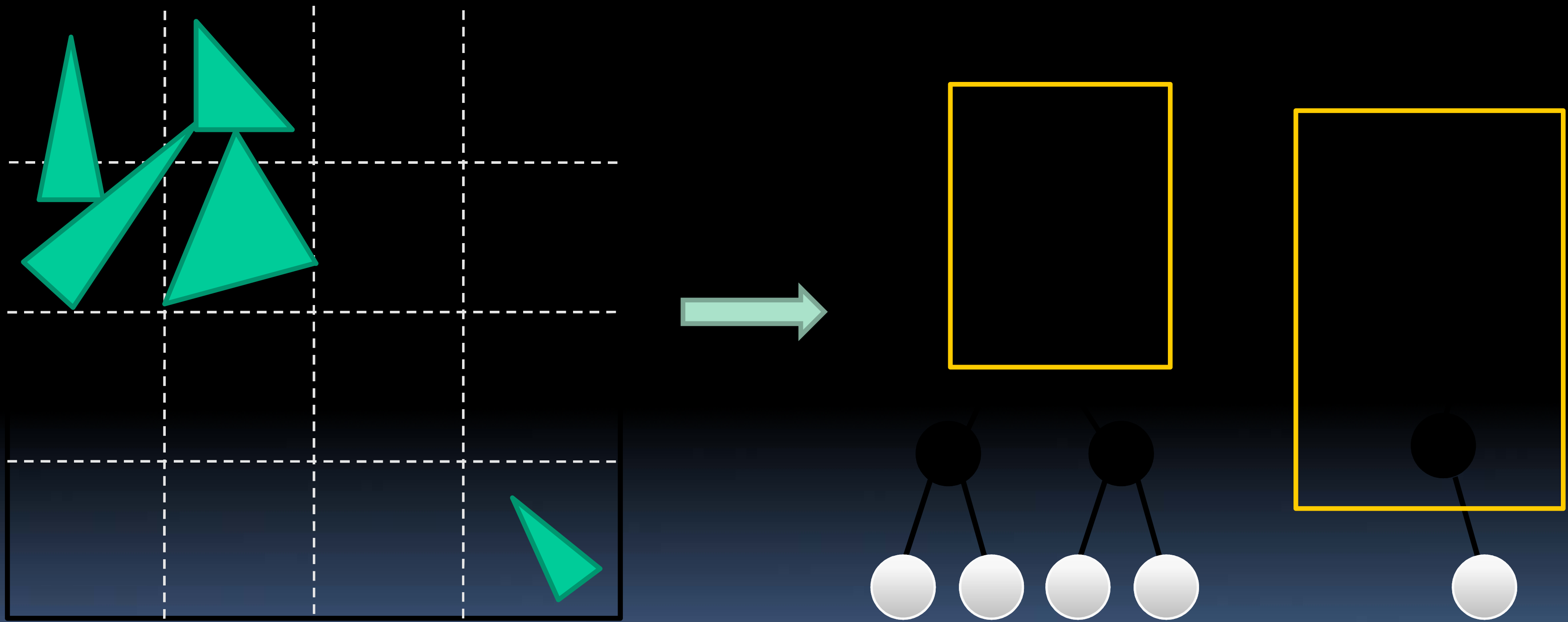
Key: Bit differences in Morton numbers



Split Unrolling

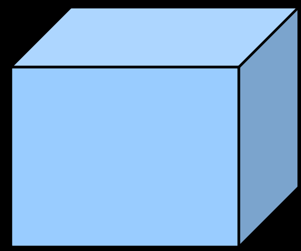
One caveat: can produce empty splits

Collapse these sequences afterwards



Bounding volumes

- Most popular (on CPUs): AABBs
  - Cheap tests, compact storage
  - But: more false positives

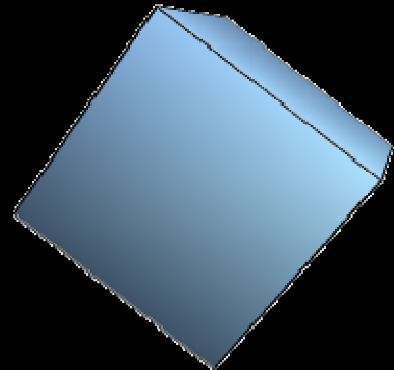


- On GPUs
  - More computational intensity is good!
  - More steps / memory loads are bad

→ Use tighter bounding volumes

Bounding volumes

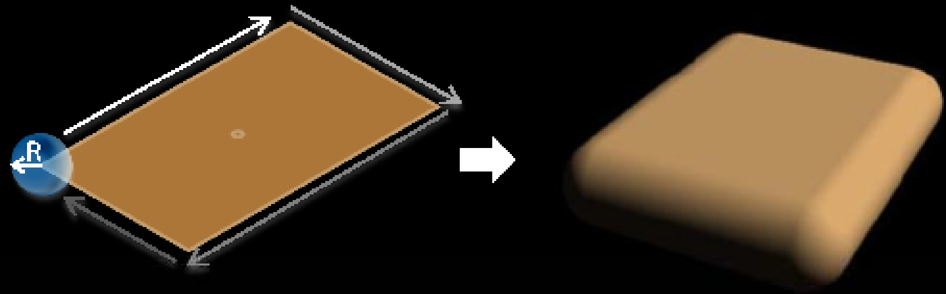
- We use oriented bounding boxes (OBBs) on GPUs
  - Operations: about 1-2 order of magnitude more instructions



- But:
  - Hierarchy construction only ~25% slower for OBBs
  - Better culling efficiency (fewer overall tests)
  - Overall performance win (especially for continuous collision and distance queries)

Bounding volumes

- Separation distance:
  Rectangular swept spheres (widely used in PQP)



  - Also has expensive construction
  - Similar advantages, easy extension of OBBs

Front tracking

- Exploit temporal coherence
  - Simulations typically have small timesteps
- Store last intersecting pair for each subtree
- Next frame: still intersecting?
  - Yes: test primitives
  - No: go up in tree until intersection found

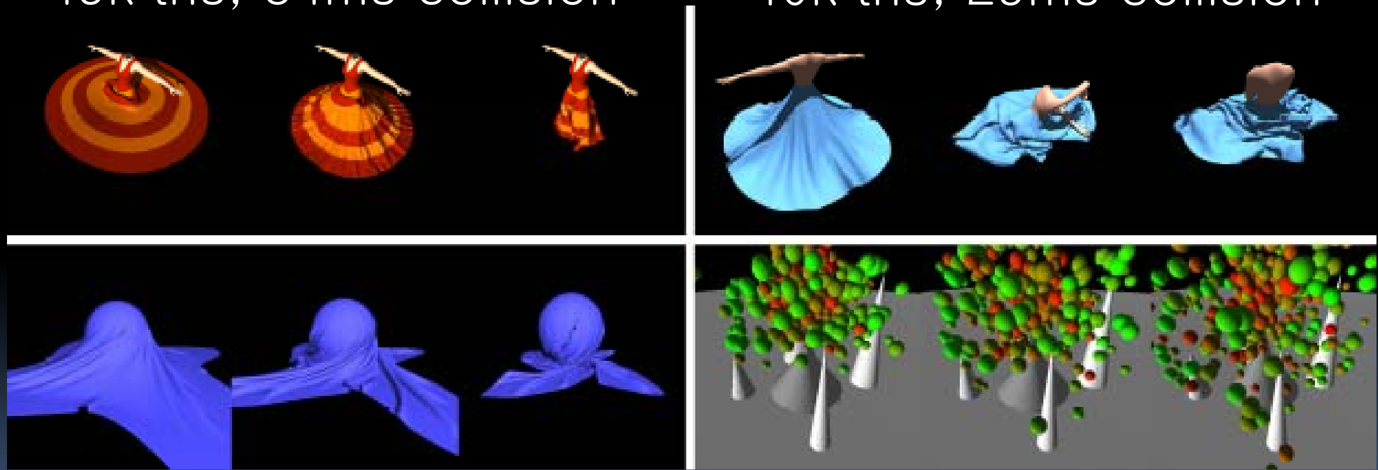
Front tracking

- Advantage
  - Less steps in intersection
  - Not necessarily less work, but higher parallelism
- Overall
  - ~10-25% less overall time for our benchmarks

Results

- Implemented in CUDA on NVIDIA GTX 285
  - Hierarchies built and updated fully on GPU
- Self-collision
  - Includes collision and update of BVH per frame

49k tris, 34ms collision	40k tris, 29ms collision



92k tris, 38ms collision	146k tris, 74ms collision

Video

- gProximity Video on NVIDIA GTX 285

Results

- Recently ported to NVIDIA GeForce 480 desktop GPU
  - 2.5 – 3X improvement over NVIDIA GeForce 285

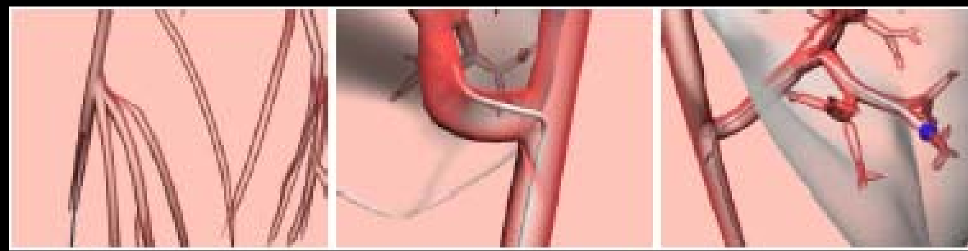
Results

- Low performance overhead for continuous
  - Worst case 50% slower
- OBB outperforms AABB on complex models
- Overall speed comparable to 8 –16-core CPU *[Kim et al. 09]*
- Much faster than previous GPU approaches (10X faster) *[Govindaraju et al. 03; Govindaraju et al. 05; Sud et al. 06]*

Separation distance

- Separation distance
  - Using RSS as bounding volume

245k tris, 55ms query



12k tris, 28ms query

  - Still 8x faster than previous GPU approaches

Limitations

- Not very good speedup for two-object collision
  - Need extremely large models for parallelism
- Separation distance
  - Similar problem for small models
  - More synchronization required

Future work

- Impact of newer GPU architectures
  - E.g. better caches
- Integration with culling methods for self-collision
  - E.g. [Curtis et al. 08: R-Triangles]
- Application of work method to recursive algorithms
- Massively parallel collision checks for motion planning/navigation

Conclusion

- GPUs can be good at speeding up collision queries using hierarchies
- GPU architectures lead to different choices
  - Computationally expensive bounding volumes good
  - Continuous collision with low overhead

Acknowledgments

- Funding agencies:
  - NSF
  - ARO
  - DARPA/RDECOM
  - NVIDIA
  - Intel
  - Models courtesy of Disney, Kineo CAM
- Liangjun Zhang, Will Moss

Thanks!