Universidade Federal de Viçosa, MG, Brazil
Rensselaer Polytechnic Institute, Troy NY USA

Employing GPUs to Accelerate Exact Geometric Predicates for 3D Geospatial Processing

Marcelo Menezes
Salles V. G. de Magalhães
Matheus Aguilar
W. Randolph Franklin
Bruno Coelho

2nd ACM SIGSPATIAL International Workshop on Spatial Gems (SpatialGems 2020)

This paper’s contribution

- A faster solution to erroneous computations caused by floating point finite precision computations.
- Errors can cause predicates (conditionals) to be evaluated wrong.
- That can cause topological errors.
- Other techniques are either very slow or may fail.
- GPU + interval arithmetic → performance and exactness.

The problem of roundoff errors

- Floating-point errors: computational geometry challenge.
- Generate topological inconsistencies: global impossibilities.
  - intersection point between two lines may not lie in either.

Rationals: one roundoff error solution

- Solution for roundoff errors: exact arithmetic (e.g. GMP rationals), but challenges:
  - Slower than floats (software-based)
  - Apparently little prior art of working (not just proposed) rational number systems on GPUs

Arithmetic filters and Interval arithmetic (IA)

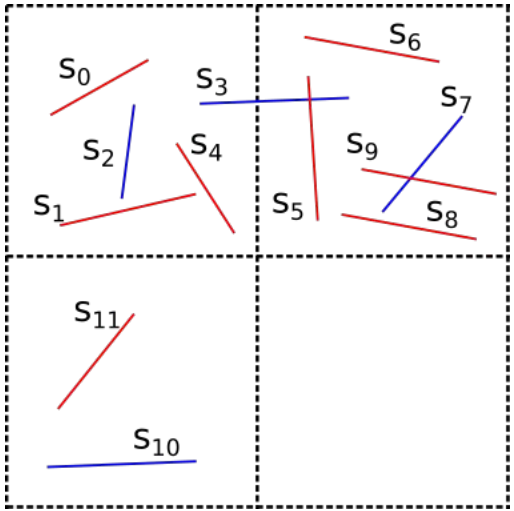
- Technique used in several CG implementations, e.g.: CGAL
- Use exact arithmetic only when **really** necessary
- Basic idea:
  - Computation performed with floats.
  - IA allows us to know if results are reliable.
  - Thus, if reliable → return them.
  - Otherwise (rare), we re-compute using exact arithmetic.

Idea for using exact computation and GPUs

- GPUs:
  - excellent for floating-point arithmetic
  - however, warps of 32 threads should run same instruction stream on adjacent data
- Implement the IA computation on the GPU
- CPU batch offloads evaluation of predicates to GPU.
- Unreliable results are filtered and re-evaluated on the CPU.

Idea for using exact computation and GPUs

- Pre-process data on the CPU (e.g.: index filtering)
- Offload predicates in batch to the GPU
  - Evaluation with FP/IA
- Uncertain results: re-evaluated on the CPU with rationals.

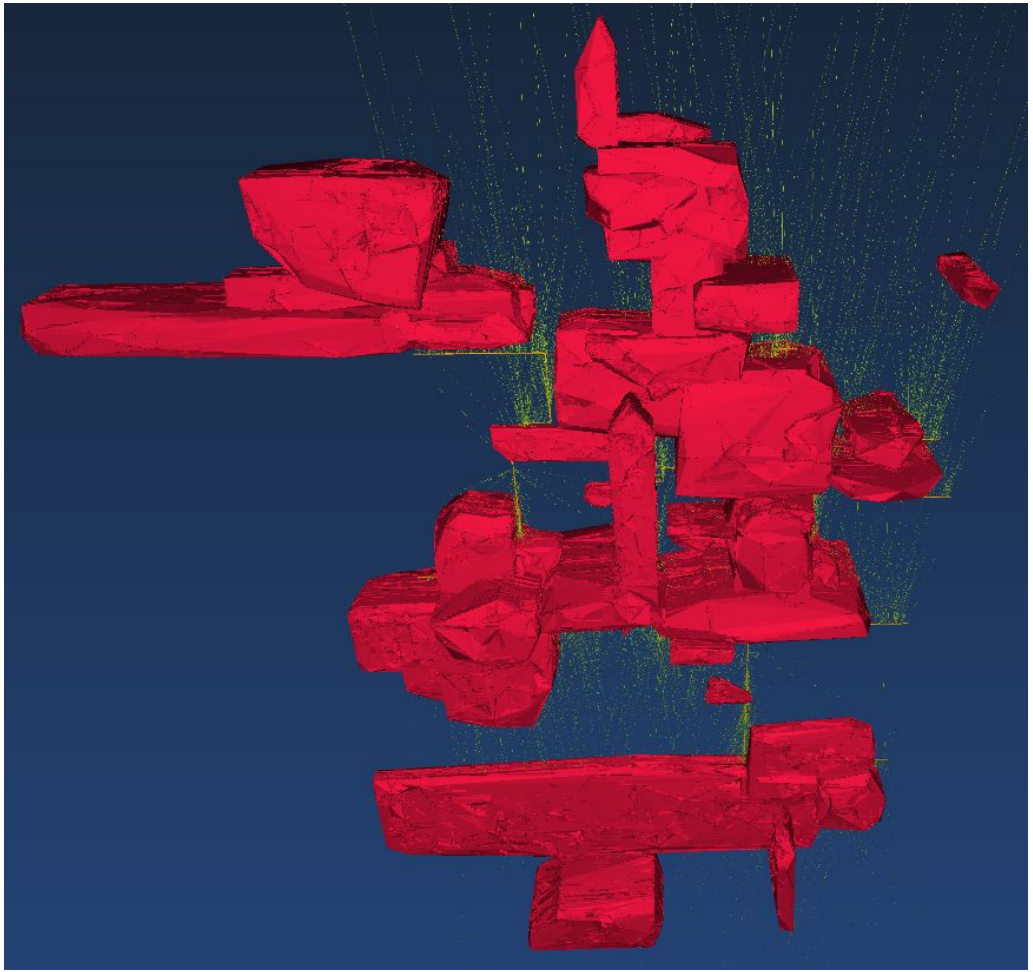


Example of uniform grid index for the 2D red-blue segment intersection

- Only pairs in the same cell may intersect.

Case study

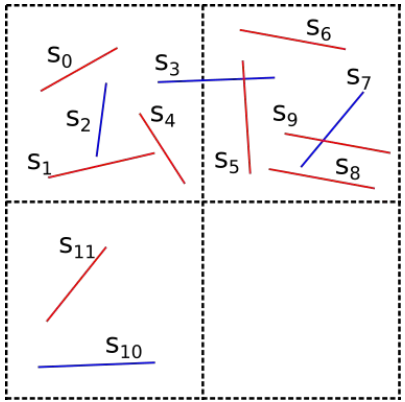
- Presented in a very interesting Sigspatial 2019 paper.
- Detecting pairwise intersections of segments and triangles in 3D.
- Example of application: 3D model of a mining dataset.
  - Segments: sampling drill holes.
  - Triangles: representing minerals polyhedra.



Source: https://github.com/lucasvr/synthetic-mine-maker

Challenges

- First idea:
  - CPU: index generates list of potential intersections.
  - Transfer list of pairs of potentially intersecting triangles to GPU.
  - Transfer back list of intersecting pairs.
- Challenges:
  - Too much work for CPU.
  - Too much data to be transferred between CPU and GPU.

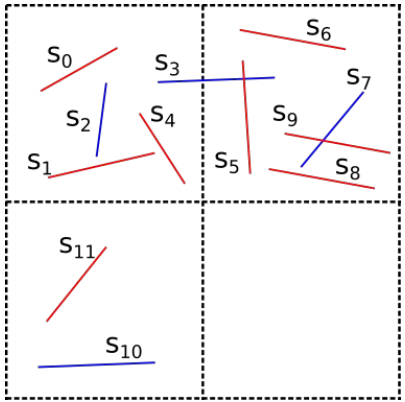


(considering the 2D red-blue intersection problem):

- Index → CPU: creates list with potential intersections: (s0,s2), (s0,s3), (s1,s2), (s1,s3), (s4,s2), (s4,s3), (s6,s3), (s6,s7),....
- **List** copied to GPU
- GPU (processes the list):
  - Thread 0: (s0,s2)
  - Thread 1: (s0,s3)
  - Thread 2: (s1,s2)
  - ...

Challenges

- Second idea:
  - CPU: creates index.
  - GPU: generates potentially intersecting pairs “on-the-fly” and process (one thread per segment or triangle)
  - Transfer back list of intersecting pairs.
- Challenges:
  - Load balancing.

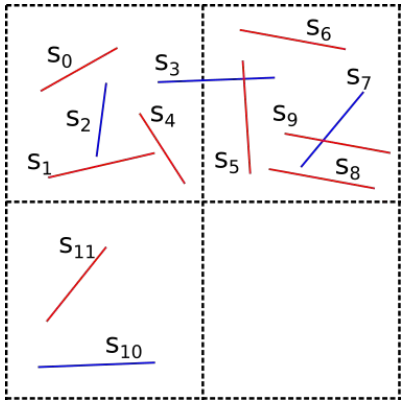


(considering the 2D red-blue intersection problem):

- **Grid** copied to GPU.
- GPU:
  - Thread 0: (s0,s2), (s0,s3)
  - Thread 1: (s1,s2), (s1,s3)
  - ...
  - Thread 7: (s11,s10) (unbalance)
  - ….

Challenges

- Third idea:
  - CPU: creates index.
  - GPU: generates potentially intersecting pairs “on-the-fly” and process (one thread per pair)
  - Transfer back list of intersecting pairs.
- Challenge: (solved on the paper)
  - How does a thread know which pair to process?



(considering the 2D red-blue intersection problem):

- **Grid** copied to GPU.
- Each thread:
  - Finds out which pair of segment it will process.
    - Thread 0: (s0,s2)
    - Thread 1: (s0,s3)
    - Thread 2: (s1,s2)

Experiments

- AMD Ryzen 5 1600 machine + Geforce GTX 1070Ti GPU (non-professional GPU!).
- Mining dataset (from sigspatial 2019): 1 000 000 triangles and 7846 segments.

Experiments

- Generate list of pairs on the CPU and process on GPU:
  - Too high pre-processing/memory overheads.
  - Good load balancing.

	Sequential	List of pairs	One segment/thread	One triangle/thread	Mult. blocks/cell
Pre-processing	-	0.67	0.00	0.00	0.00
Mem. (CPU to GPU)	-	0.21	0.02	0.01	0.02
Intersection	2.48	0.09	5.96	0.69	0.10
Mem. (GPU to CPU)	-	0.02	0.00	0.00	0.00
Rationals	0.00	0.03	0.00	0.00	0.00
Ensure uniqueness	0.01	0.01	0.02	0.02	0.02
Total	2.50	2.14	6.00	0.72	0.15

Experiments

- Index processed on GPU.
- Each GPU thread processes a segment.
- Segment tested for intersection against all triangles intersecting its bounding-box (index employed on GPU).
- Slow, low occupancy (much less segments than triangles), load unbalance.

	Sequential	List of pairs	One segment/thread	One triangle/thread	Mult. blocks/cell
Pre-processing	-	0.67	0.00	0.00	0.00
Mem. (CPU to GPU)	-	0.21	0.02	0.01	0.02
Intersection	2.48	0.09	5.96	0.69	0.10
Mem. (GPU to CPU)	-	0.02	0.00	0.00	0.00
Rationals	0.00	0.03	0.00	0.00	0.00
Ensure uniqueness	0.01	0.01	0.02	0.02	0.02
Total	2.50	2.14	6.00	0.72	0.15

Experiments

- Index processed on GPU.
- Each GPU thread processes a triangle.
- Triangle tested for intersection against all segments intersecting its bounding-box (index employed on GPU).
- Lower time (better occupancy), but problem with load balancing (some triangles bbox: intersect much more segments than others)

	Sequential	List of pairs	One segment/thread	One triangle/thread	Mult. blocks/cell
Pre-processing	-	0.67	0.00	0.00	0.00
Mem. (CPU to GPU)	-	0.21	0.02	0.01	0.02
Intersection	2.48	0.09	5.96	0.69	0.10
Mem. (GPU to CPU)	-	0.02	0.00	0.00	0.00
Rationals	0.00	0.03	0.00	0.00	0.00
Ensure uniqueness	0.01	0.01	0.02	0.02	0.02
Total	2.50	2.14	6.00	0.72	0.15

Experiments

- Index processed on GPU.
- Multiple GPU threads per grid (index) block (proportional to number of pairs).
- Proportional → better load balancing/occupancy.
- 17x speedup, 25x speedup for the intersection step.

	Sequential	List of pairs	One segment/thread	One triangle/thread	Mult. blocks/cell
Pre-processing	-	0.67	0.00	0.00	0.00
Mem. (CPU to GPU)	-	0.21	0.02	0.01	0.02
Intersection	2.48	0.09	5.96	0.69	0.10
Mem. (GPU to CPU)	-	0.02	0.00	0.00	0.00
Rationals	0.00	0.03	0.00	0.00	0.00
Ensure uniqueness	0.01	0.01	0.02	0.02	0.02
Total	2.50	2.14	6.00	0.72	0.15

Conclusions

- GPU can be employed to accelerate exact computation.
- Good speedups: 17x speedup, 25x speedup for the intersection step.
- Achieve both: performance and quality.
- Future work:
  - Optimize solution using shared memory
  - Map/Mesh intersection using GPUs.
  - Other geometric problems. (e.g. point location)

Reviews

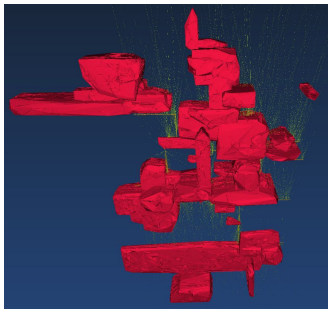
- We thank the reviewers for their time and valuable comments.
- We were asked to make only minor modifications.
- Some sentences were not clear on the paper. We improved them.
- One of the reviewers asked us to present more details about performing other operations with IA (for example, multiplication)
    - Because of space limitations, we included a reference explaining how to perform several operations with IA. (there are also some details at the end of this talk)

Thank you!

	Sequential	List of pairs	One segment/thread	One triangle/thread	Mult. blocks/cell
Pre-processing	-	0.67	0.00	0.00	0.00
Mem. (CPU to GPU)	-	0.21	0.02	0.01	0.02
Intersection	2.48	0.09	5.96	0.69	0.10
Mem. (GPU to CPU)	-	0.02	0.00	0.00	0.00
Rationals	0.00	0.03	0.00	0.00	0.00
Ensure uniqueness	0.01	0.01	0.02	0.02	0.02
Total	2.50	2.14	6.00	0.72	0.15

Contact: marcelo.menezes@ufv.br

```
1  class CudaInterval {
2  public:
3      __device__ __host__ CudaInterval(const double l, const double u)
4          : lb(l), ub(u) {}
5      ...
6      __device__ CudaInterval operator+(const CudaInterval& v) const {
7          return CudaInterval(__dadd_rd(this->lb, v.lb),
8              __dadd_ru(this->ub, v.ub));
9      }
10     ...
11     __device__ int sign() const {
12         if (this->lb > 0) // lb > 0 implies ub > 0
13             return 1;
14         if (this->ub < 0) // ub < 0 implies lb < 0
15             return -1;
16         if (this->lb == 0 && this->ub == 0)
17             return 0;
18         // If none of the above conditions is satisfied, the sign of the
19         // exact result cannot be inferred from the interval, Thus, a flag
20         // is returned to indicate an interval failure.
21         return 2;
22     }
23     ...
24 private:
25     double lb, ub; // Stores the interval's lower and upper bounds
26 };
```



Source: https://github.com/lucasvr/synthetic-mine-maker