Streaming Geometric Computations on the GPU

Shankar Krishnan
AT&T Labs - Research

Two Converging Trends in Computing ...

• The accelerated development of graphics cards
  - developing faster than CPUs
  - GPUs are cheap and ubiquitous
• Increasing need for streaming computations
  - original motivation from dealing with large data sets
  - also interesting for multimedia applications, image processing, visualization etc.

What is a Stream?

• An ordered list of data items
• Each data item has the same type
  - like a tuple or record
• Length of stream is potentially very large
• Examples
  – data records in database applications
  – vertex information in computer graphics
  – points, lines etc. in computational geometry

Streaming Model

• Input presented as a sequence
• Algorithm works in a single pass
  - allowed one sequential scan over input
  - not permitted to move backwards mid-scan
• Workspace
  - typically o(n)
  - arbitrary computation allowed
• Algorithm efficiency
  - size of workspace and computation time
Streaming: Data driven to Performance driven

- Primary motivation is computing over transient data (data driven)
  - data over a network, sensor data, router data etc.
- Computing over large, disk-resident data which are expensive to access (data and performance driven)
- To improve algorithm performance

How does streaming help performance?

Von Neumann Bottleneck

- Memory bottleneck
  - CPU processing faster than memory bandwidth
  - discrepancy getting worse
  - large caches and sophisticated prefetching strategies alleviate bottleneck to some extent
  - caches occupy large portions of real estate in modern ship design

Cache Real Estate

Die photograph of the Intel/HP IA-64 processor (Itanium2 chip)
Von Neumann Bottleneck

- Memory bottleneck
  - CPU processing faster than memory bandwidth
  - discrepancy getting worse
  - large caches and sophisticated prefetching strategies alleviate bottleneck to some extent
  - caches occupy large portions of real estate in modern ship design
- Unacceptable for computationally intensive applications

Proposed Solutions

- Systolic Arrays (Kung-Leiserson ’78)
  - computational units arranged in specific topology (like grid or line)
  - data flows from one computational unit to its neighbors

- SIMD Architectures
  - single set of instructions executed by different processors
  - multiple data streams fed to each processing unit

- Vector architectures
  - vector registers and vector processing units improve bandwidth
Stream Architectures

- All input in the form of streams
- Stream processed by a specialized computational unit called kernel
- Kernel performs the same operations on each stream element

Stream Architectures

- Items processed in a FIFO fashion
- Reduced memory latency and cache requirements
- Simplified control flow
- Data-level parallelism
- Greater computational efficiency
- Examples
  - CHEOPS [Rixner et. al. '98] and Imagine [Kapasi et. al. '02]
  - High performance media applications

Graphics Pipeline: Rendering View

- Commands
- Display List
- Imaging Data
- Rasterization
- Texture Memory
- Frame Buffer
- Host
- Geometry Engine
- Raster Manager
- Primitive Operations
- Pixel Operations

Graphics Hardware Pipeline

- Vertex Connectivity
- Vertices
- Transformation
- Primitive Assembly and Rasterization
- Fragments
- Fragment Texturing and Coloring
- Raster Operations
- Pixel Updates

Courtesy: The Cg Tutorial [Fernando and Kilgard]
**Programmable Graphics Pipeline**

- **GPU Front End**
  - Pre-transformed Vertices
  - Pre-transformed Fragments
- **CPU-GPU Boundary**
- **3D API:** OpenGL or Direct3D
- **GPU Command & Data Stream**
- **CPU-GPU Boundary**
- **GPU Capabilities**
  - Large instruction set for general purpose scalar and vector arithmetic
  - General purpose memory access through textures
  - Limited pointer indirection through dependent textures
  - High level language support
    - Cg, HLSL

**GPU: A Streaming Pipelined Architecture**

- Inputs presented in streaming fashion
  - processed data items pass to next phase (1D systolic array?) and does not return
- Data-level parallelism
- Limited local storage
  - data items essentially carry their own state
- Pipelining: each item processed identically
- Not quite general purpose yet, but getting there

**Standard Streaming Model**

- What’s the difference?
  - local memory (constant vs. \( \text{polylog } n \))
  - pipelining restriction
  - multi-pass potential

**GPU Capabilities**

- Large instruction set for general purpose scalar and vector arithmetic
- General purpose memory access through textures
- Limited pointer indirection through dependent textures
- High level language support
  - Cg, HLSL
Diverse Applications

- Visibility, shadow computation
- Occlusion culling
- Motion planning, collision detection
- Physically-based modeling
- Image processing, FFT, wavelet analysis
- Radiosity, radiance, ray tracing
- Linear algebra, differential equations
- Computational geometry, solid modeling
- A lot more ...

Streaming Geometric Computations on the GPU

Shankar Krishnan
AT&T Labs - Research

Geometric Algorithms in Hardware

Use computational power of the GPU to implement geometric algorithms

- Large speedups
- Circumvent problems of geometric complexity
- For some problems, CPU-bound solutions are hard
- Can handle other geometric settings (dynamic/kinetic)

Issues of Error

- Geometric input is continuous
- Computation and output are on finite-precision grid
- Error determined by grid resolution and rasterization process
- Approximation algorithms
Solid Modeling using CSG

- Constructive Solid Geometry (CSG)
  - way to model general solids
- Solid represented as Boolean combination of simple primitives
Solid Modeling using CSG

• Constructive Solid Geometry (CSG)
  – way to model general solids
• Solid represented as Boolean combination of simple primitives
• Two possibilities to render these objects
  – compute boundary representation using sophisticated geometric algorithms
    • expensive, robustness is an issue
  – directly render them implicitly
    • no mesh representation
    • effective for quick feedback during design process

Tree Normalization

• CSG expression
  – general expression with unions, intersections and differences
• Can be modified to canonical sum-of-products form
  – algorithm provided by Goldfeather et. al.
• Assumed as input in the rest of talk

Example: Bradley Fighting Vehicle

Bradley Fighting Vehicle

• Over 8000 primitive objects
  – polyhedra
  – ellipsoids, generalized quadrics
  – tori
  – surfaces of revolution
• Over 5000 CSG operations totally
  – individual trees vary from 10s to 100s of CSG operations

Courtesy: Army Research Lab
Previous Work

- Goldfeather et. al. ['85, '89]
  - use of parity
- Epstein et. al. ’88, Rossignac & Wu ’92
  - trickle algorithm
  - depth-interval buffers
- Weigand ’95
  - implementation of Goldfeather on standard graphics pipeline
- Stewart et. al. [’98, ’00, ’02]
  - improvements on Goldfeather

Parity and Depth Test

- Idea introduced by Goldfeather et. al. [’85, ’89]

Main Results

- Use two-sided depth test to sweep an arrangement of objects
- Perform CSG rendering in $O(n)$ fewer passes (Guha et. al. – I3D ’03)
  - optimal, no readbacks
- Extract arbitrary layer of a scene in logarithmic instead of linear passes
- First known lower bound results for algorithms on the GPU

Two-Sided Depth Test

- Conceptualized by Mammen ’89
- Implemented on current GPUs
  - shadow mapping hardware [Bastos, Everitt] on nVidia cards
  - simple fragment program
- Depth peeling applications
  - order-independent transparency
  - opacity light field mapping [Vlasik et. al. 03]
Algorithm (Single Product)

- Compute first level of arrangement
- Determine portions of level contained in intersection
  - depthmask = FALSE, depthtest = ~
  - $\sum$ front faces - $\sum$ back faces = n
  - use two-sided stencil test
- Advance to next level using depth peeling

Example

Algorithm: Union of Products

- Compute each product as before
- Merge depth and color field of current product with that of prefix sum
  - prefix depth stored in second depth texture
  - two pass merge step
- Rendering passes
  - single product: linear in product size
  - sum of products: sum of depth complexity

Example: Helix

Cyl1 – (Helix1 U Helix2 U Cyl2)
Example: Helix

Cyl1 – (Helix1 U Helix2 U Cyl2)

Helix: First Layer

Layer of Arrangement

Resulting Depth Field

Helix: Second Layer

Layer of Arrangement

Merged Depth Field
Bradley: 25 mm Gun

Bradley: Idlerwheel

Bradley: Drivewheel

Discussion

- Two-sided depth test
  - Novel algorithm for rendering CSG trees of general (but simple) closed shapes
    - optimal with respect to methods based on parity/counting
  - Arbitrary layer extraction from a scene
    - $O(\log n)$ rendering passes
    - planning under infeasible constraints
    - geometric optimization problems
Geometric Optimization on GPUs

What is Geometric Optimization?

• Computing statistical measures and approximate representations to geometric data
  - given a set of points, what is its diameter?
  - given a collection of triangles, find the smallest enclosing OBB?
  - given two intersecting convex polytopes, find the smallest translation vector of one to separate them – penetration depth?
  - given a collection of 2D shapes, pack them into smallest axis-aligned rectangle – polygon compaction?

Problem Characteristics

• Objective functions are (piecewise) algebraic
  - mostly linear
• Can be formulated as
  - lower/upper envelopes
  - overlay of multiple envelopes
• Hardware provides unified solutions to most of these problems
  - provably approximate solutions

Geometric Optimization

• Solving exactly is computationally expensive
  - best fit OBB: $O(n^3)$
  - each problem needs specialized solution
• Still interesting from a streaming point of view
  - can we design algorithms that are efficient, yet provably approximate?
• Can GPUs be used to solve these problems?
Point-Hyperplane Duality

- Points in primal map to lines in dual — and vice versa
- Convex hull of points in primal — upper and lower envelope in dual
- Direction vector in primal — maps to point in the dual domain — central projection

Duality

Central Projection

- Cover space of directions with bounded cube

Gaussian sphere

Bounded Dual
3D Diameter
- Diameter pair realized in the convex hull
- Dualize all the points
  - RGB space encodes point coordinates
- Upper and lower envelope determines antipodal pairs
- Two rendering passes to determine diameter
- Frame buffer resolution decides approximation factor

List of Problems Solved
- Extent measures
  - 2D and 3D Width and Diameter
  - 2D and 3D Oriented Bounding Box
  - 1-Median, 1-Center and Closest pair
- Shape Matching/Fitting
  - Hausdorff and Summed-Hausdorff metrics under translation
  - best-fit line and circle
- Layered Manufacturing
- Path Planning (translation and rotation)

Penetration Depth
- Given two objects A and B, find smallest translation vector $\mathbf{t}$ such that $(A + \mathbf{t})$ is disjoint from B
- Equivalent to minimum distance from origin to $(A \odot -B)$, the Minkowski sum
- Minkowski sum
  - quadratic complexity even for convex objects

Minkowski Sum
$$ M = A \circ B = \{a+b \mid a \in A \text{ and } b \in B\} $$
Penetration Depth

PD for Convex Objects
- Convex objects are closed under Minkowski sums
- Let \( M = A \odot -B \)
- Dualize all vertices of \( A \) and \( B \)
- Two observations
  - lower envelope of \( M \)'s dual is sum of lower envelopes of \( A \) and \( B \)'s dual
  - min. dist. from point \( p \) to \( M = \) min. dist. from \( p \) to all planes tangential to \( M \)

PD Algorithm
- Compute lower and upper envelope of dual planes to \( A \) and \( -B \)
- Sum corresponding envelopes
- Compute minimum
- Location of minimum in dual gives translation vector

PD Results
Original Position
After Separation
Discussion

• Problem reformulation can lead to pipelined, streaming algorithms
• For many applications, back-end fast geometric computations are needed
• Resulting algorithms are very efficient
  – comparable, if not faster than sophisticated software techniques
• Current techniques restricted to 2D and 3D

Open Issues

• Can we extend standard theoretical model of streaming to this somewhat restricted notion?
  – implications of multi-pass potential?
• What are limitation of stream architectures?
  – problems that can/cannot be addressed in this framework
• Issues of programming language design, compilers, OS and hardware design

Pipelined Streaming: Conclusions

• Stream architectures
  – alternate model for high-performance computing
  – GPU is readily accessible, easy-to-use platform for working with streams
    • numerous applications with demonstrable performance gain
  – strictly weaker than general streaming
    • probably stronger than circuit models
Questions?

Contact
krishnas@research.att.com