Real-time Computation of Data Depth Using the Graphics Pipeline

Suresh Venkatasubramanian

AT&T Labs–Research
The Interplay Between Analysis and Visualization

- Most methods for computing data depth solve the problem, and then visualize the answers.
- Much of data analysis is exploratory and interactive.
- Not only do we need fast solutions, we need ways of interacting with (possibly very large) data.

Can we combine analysis and visualization?

- Modern video cards have immense untapped computing potential.
- There is a growing trend in graphics and scientific computing to treat the video card as a fast co-processor.
**Graphics Cards Can Compute!**

A graphics card takes a stream of objects (points, lines, triangles), and renders them on a screen.

Each pixel in the screen can be viewed as a small processing unit.

<table>
<thead>
<tr>
<th>glBlend</th>
<th>$a = a \oplus b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>z-test</td>
<td>$a = \min(a, b)$</td>
</tr>
</tbody>
</table>
The Pipeline And Data Analysis, or Who Cares?

- The interactive nature of data analysis makes speed a crucial consideration.
- Visualization is a key component: the use of graphics cards is natural.
- Demonstrable performance gain in areas like scientific computing.
- Serious efforts are underway to make the computations robust.
- The graphics card as a *streaming co-processor* is becoming common in diverse areas (graphics, robotics, numerical analysis, physical simulation, geometry).
Overview Of This Talk

A brief overview of the graphics pipeline
- How do we write programs for the graphics pipeline?
- The architecture of a card.

Computing various data depth measures in hardware
- A simple algorithm for location depth.
- Implementation in hardware.
- Error Analysis and Performance
- Extensions to simplicial depth, Oja depth, colored location depth, and other depth measures.

Joint work with Shankar Krishnan (AT&T) and Nabil Mustafa (Duke)
The Graphics Pipeline
An Example OpenGL Program

```c
#include <gl.h>
...

// Set lighting
glLight(..);

// Set viewpoint
glOrtho(..);

// Now draw objects

// glLight position
glColor(1, 0, 0);

// glBegin(GL_TRIANGLES)

// glVertex3d(x1, y1, z1)

// glEnd()

gcc triangle.cc -lGL
```
Processing Objects in the GPU: Step 1

The Fixed-Function Pipeline

- Vertices
- Color
- Lighting

CPU → Viewpoint Calculations → Lighting and color transforms → Rasterization → GPU

Fragments
Processing fragments in the GPU: Step 2

The Fixed-Function Pipeline
So where’s the computation?

- Stencil test
  ```cpp
  if (buffer.stencil = K) continue
  else drop fragment.
  ```

- Depth test
  ```cpp
  if (frag.depth < buffer.depth) continue
  else drop fragment.
  ```

- Blending operations
  ```cpp
  buffer.color = buffer.color op fragment.color
  ```
  – General arithmetic and boolean function for blending.
  – General comparison functions.
  – Convolution and histogramming operators.

Each pixel executes the same program in “parallel”
Programable Pipelines

- Viewpoint Calculations
- Lighting and color transforms

Vertex program executes on each vertex.
Fragment program executes on each fragment.
Why is it so fast?

- The processor is highly optimized for streaming operations.
- On a per-unit area basis, far more computational (ALU) units than a standard CPU.
- Because of FIFO nature of computation, almost non-existent memory latency.
- Immense *spatial parallelism*: each pixel can be thought of as a tiny parallel processor (all executing the same program).

Cost Model:

- Each rendering pass is a “unit-cost” operation.
- Reading data back into main memory is expensive.
- Objective is to minimize the number of passes.
- Akin to standard notions of stream computations.

In each pass, only a fixed set of operations can be performed.
Data Depth Computation
Halfspace Depth: Primal and Dual

Depth of point in primal $\equiv$ Minimum depth of line in dual
Template For Hardware-Based Approach

1. Construct dual arrangement. For each point in the dual, determine its depth.
2. For each point on a line in the dual, draw it in the primal plane with an associated value equal to its depth.
3. At each point in primal, retain the smallest value encountered.
Step 1: Computing Dual Depth

- Draw trapezoid for each line.
Step 1: Computing Dual Depth

- Draw trapezoid for each line.
- Increment counter at each touched pixel.
Step 1: Computing Dual Depth

- Draw trapezoid for each line.
- Increment counter at each touched pixel.
- Draw next line.
Step 1: Computing Dual Depth

- Draw trapezoid for each line.
- Increment counter at each touched pixel.
- Draw next line.
- Increment counter as before.
**Step 1: Computing Dual Depth**

- Draw trapezoid for each line.
- Increment counter at each touched pixel.
- Draw next line.
- Increment counter as before.
- Repeat for all lines.
Step 1: Computing Dual Depth

- Draw trapezoid for each line.
- Increment counter at each touched pixel.
- Draw next line.
- Increment counter as before.
- Repeat for all lines.

At end of Step 1, all pixels in dual have correct depth.
Step 2: Drawing In The Primal Plane

For all points lying on dual lines...
Step 2: Drawing In The Primal Plane

For all points lying on dual lines...

Draw primal line with dual depth value.
Step 2: Drawing In The Primal Plane

- For all points lying on dual lines...
- Draw primal line with dual depth value.
- Repeat...

For all points lying on dual lines...
Step 2: Drawing In The Primal Plane

For all points lying on dual lines...

Draw primal line with dual depth value.

Repeat...
Step 2: Drawing In The Primal Plane

- For all points lying on dual lines...
- Draw primal line with dual depth value.
- Repeat...
- Update pixel with minimum value seen.
Step 2: Drawing In The Primal Plane

- For all points lying on dual lines...
- Draw primal line with dual depth value.
- Repeat...
- Update pixel with minimum value seen.

At end of Step 2, all pixels in primal have correct depth
Bounded Duals

The screen has bounded size! (typically $[-1, 1]^2$)
If two points are almost above each other in the primal, the dual point is near $\infty$.

Solution: use multiple duals.

Definition. A point is bounded if it lies in the range $[-1, 1] \times [-2, 2]$.

Theorem. There exists two dual mappings $\mathcal{D}_1, \mathcal{D}_2$ such that each intersection point in the dual arrangement is bounded in one of them.

Proof Sketch: Each dual covers a different portion of the space of directions $S^1$. □
**Pixelization Error**

The screen has bounded resolution\. No exact solution is possible.

**A Grid Algorithm:**
For a **given** point set $P$, determine grid resolution $W$ needed to compute an answer correctly.

- In general, the desired grid resolution is a simple function of the input point set.
- The higher the grid resolution, the slower the running time.
Levels of Detail

Because of the relative speed of computation, we can compute a fast approximate answer, and refine the answer by *zooming* into regions of interest.
Running Time

- Step 1 can be performed in two passes (one for each dual).
- One readback is required to obtain the dual depth values.
- Step 2 can also be performed in one pass. However, $W^2$ objects are rendered (which could be much larger than $n$).

<table>
<thead>
<tr>
<th>Size</th>
<th>Running time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.6</td>
</tr>
<tr>
<td>100</td>
<td>0.9</td>
</tr>
<tr>
<td>500</td>
<td>1.9</td>
</tr>
<tr>
<td>1000</td>
<td>2.5</td>
</tr>
<tr>
<td>5000</td>
<td>6.3</td>
</tr>
<tr>
<td>10000</td>
<td>11.1</td>
</tr>
</tbody>
</table>
Running Time

- Step 1 can be performed in two passes (one for each dual).
- One readback is required to obtain the dual depth values.
- Step 2 can also be performed in one pass. However, $W^2$ objects are rendered (which could be much larger than $n$).

<table>
<thead>
<tr>
<th>Size</th>
<th>Running time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.6</td>
</tr>
<tr>
<td>100</td>
<td>0.9</td>
</tr>
<tr>
<td>500</td>
<td>1.9</td>
</tr>
<tr>
<td>1000</td>
<td>2.5 (1.9)</td>
</tr>
<tr>
<td>5000</td>
<td>6.3 (3.2)</td>
</tr>
<tr>
<td>10000</td>
<td>11.1 (4.5)</td>
</tr>
</tbody>
</table>
Movie
Other Depth Measures
Can We Build Upon This?

Various algorithm modules can be implemented in hardware:

- Envelope calculations.
- Dual mappings.
- Distance fields
  - Voronoi Diagrams
  - Power Diagrams
  - General Metrics
- Median (and k-selection in general)
  - Can be used to extract levels from an arrangement.
Simplicial Depth

Count number of simplices not containing $p$ and subtract from $\binom{n}{3}$. [RR96]

Sort points radially around $p$. 
Simplicial Depth

Count number of simplices not containing $p$ and subtract from $\binom{n}{3}$. [RR96]

- Sort points radially around $p$.
- Take horizontal line $\ell$ through $p$ and rotate anticlockwise till it hits a point $q$. 

Real-time Computation of Data Depth – p.25/28
Simplicial Depth

Count number of simplices \textit{not} containing \( p \) and subtract from \( \binom{n}{3} \). [RR96]

- Sort points radially around \( p \).
- Take horizontal line \( \ell \) through \( p \) and rotate anticlockwise till it hits a point \( q \).
- All pairs of points on either side of \( \ell \) define simplices \textit{not} containing \( p \).
Simplicial Depth

Count number of simplices *not* containing \( p \) and subtract from \( \binom{n}{3} \). [RR96]

- Sort points radially around \( p \).
- Take horizontal line \( \ell \) through \( p \) and rotate anticlockwise till it hits a point \( q \).
- All pairs of points on either side of \( \ell \) define simplices *not* containing \( p \).
Simplicial Depth

Count number of simplices not containing $p$ and subtract from $\binom{n}{3}$. [RR96]

- Sort points radially around $p$.
- Take horizontal line $\ell$ through $p$ and rotate anticlockwise till it hits a point $q$.
- All pairs of points on either side of $\ell$ define simplices not containing $p$. 
Simplicial Depth

Count number of simplices *not* containing $p$ and subtract from $\binom{n}{3}$. [RR96]

- Sort points radially around $p$.
- Take horizontal line $\ell$ through $p$ and rotate anticlockwise till it hits a point $q$.
- All pairs of points on either side of $\ell$ define simplices *not* containing $p$.
- Repeat for all lines.
Simplicial Depth

Count number of simplices not containing $p$ and subtract from $\binom{n}{3}$. [RR96]

- Sort points radially around $p$.
- Take horizontal line $\ell$ through $p$ and rotate anticlockwise till it hits a point $q$.
- All pairs of points on either side of $\ell$ define simplices not containing $p$.
- Repeat for all lines.
Simplicial Depth

Count number of simplices not containing $p$ and subtract from $\binom{n}{3}$. [RR96]

- Sort points radially around $p$.
- Take horizontal line $\ell$ through $p$ and rotate anticlockwise till it hits a point $q$
- All pairs of points on either side of $\ell$ define simplices not containing $p$.
- Repeat for all lines

Number of simplices one on side of $\ell$ can be computed from number of points on one side of $\ell$.

Halfspace depth computation can be used to compute simplicial depth
**Oja Depth**

**Definition (Oja Depth).** *Given a point set $P$, the Oja depth of a point $q$ is the sum of the volumes of all simplices of $P \cup \{q\}$ that contain $q$ as a vertex.*

Contribution to the depth of $q$ by the pair $p, p'$ is precisely

$$d(q, l(p, p')) \cdot d(p, p') / 2$$

Thus the depth of a point $q$ can be written as

$$\text{depth}(q) = \sum_{\ell \in \mathcal{L}} w_\ell \cdot d(q, \ell)$$

This defines a weighted *distance field*, where each object $\ell$ has weight $w_\ell$, and the influence of $\ell$ is proportional to the distance from it.

All such distance fields can be computed in the graphics pipeline very efficiently.
Other Measures

- Line of best fit
- LMS estimator.
- Best fit circle
- Colored halfspace depth
  - Each point is colored, and the depth of a point is expressed in terms of the number of unique colors.
Conclusions

- Graphics cards provide a natural fast platform for many kinds of geometric computations.
- For visualization- and interaction-heavy problems, this is a viable approach.
- When viewed from the perspective of streaming envelope computations, different problems can be solved using similar methods.

Future Directions:

- Other depth measures? More sophisticated approaches that exploit the full power of the pipeline?
- Underlying computational questions: What makes certain problems streamable?