Today

• 3 Topics
  – **Histograms**
    How often does a certain value appear in a population?
  – **Compactification**
    Produce a compact list of primitives from a sparse input, e.g. features
  – **Voronoi diagrams**
    For every pixel, compute the index of the point in a set closest to it

• Underlying principle: Gather & Scatter
Change

• Until now:
  – Iterate over all pixels with index $i$
  – Read from some constant inputs (uniforms, textures,...)
  – Produce one or a low number for each pixel $i$
• Today, $i$ will vary, e.g. in the vertex program
• This is called “scatter”, the old one “gather”

Gather

Scatter
Problems

• What primitive: Points, lines, ...
• Where?
• What combination: sum, multiply, ...
• How to skip? Culling, discarding
• No order guarantees!

Gather

Scatter
Image Histogram

- Draw a point for every pixel
- Use a vertex shader to sort it into a bin
- Combine using additive blending
Discussion

• **Pro:** Order-independent, parallelizes well
• **Con:** Scatter and atomic pressure
Visualization

• Result is 1D (or one pixel height-2D) texture
• Hard to visualize
• Cant draw to screen directly
• Solution:
  – Draw into 1D / flat texture
  – Display this texture in a new way
    • By stretching it
    • By drawing lines (Assignment)

• Drawing into textures: FBOs
• Basically these allow to render into a texture
CODE

Programming breakout

Histograms
Compactification: Problem

- The result of a computation is often sparse
- Example feature detection
- **What we want is a **compact **feature list, e.g. their**
  - Locations
  - Values

<table>
<thead>
<tr>
<th>Image</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Compact features locations</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>values</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution: PrefixScan

• **Two-step solution**
  – First: Compute output count you have at input element \( i \)
  – Second: Compute output and put it where it belongs

<table>
<thead>
<tr>
<th>Image</th>
<th>0 0 1 1 1 1 1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output count</td>
<td>0 0 1 0 0 1 0 0 0</td>
</tr>
<tr>
<td>Prefix scan</td>
<td>0 0 0 1 1 1 2 2 2</td>
</tr>
<tr>
<td>Output</td>
<td>0 0 1 0 0 -1 0 0 0</td>
</tr>
<tr>
<td>Compact features locations</td>
<td>2 5</td>
</tr>
<tr>
<td>values</td>
<td>1 -1</td>
</tr>
</tbody>
</table>
Example: HistoPyramids

• For 2D Images
• Using MIP hierarchy
• Two steps:
  1. Building
  2. Compactification

![Image of HistoPyramids example](image-url)
**Build step**

- Put 0/1 into level 0 (finest!) of a MIP map
- In higher levels, sum all 4 pixels below it
- Multiple passes
- “Final” pixel contains count

```
<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Level 0</th>
<th>Level 1</th>
<th>L.2</th>
<th>L.3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 1 0 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 0 0 1</td>
<td>0 1 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 0 0 0 1</td>
<td>0 1 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 0 0 1 0</td>
<td>0 1 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1 1 1 0 0</td>
<td>2 0 0 2</td>
<td>2 2</td>
<td>2 2</td>
<td>2 2</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 2 2 0</td>
<td>4 4</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
```
Compactification step

- Start a thread for every
Compactification step

• Start a thread for every
Compactification step

- Start a thread for every
CODE

Programming breakout
Histo Pyramids
Voronoi Diagram
Paralle Voronoi Diagram

• Trivial:
  – For all pixels in parallel
  – Loop over all centers and find the smallest

• $O(nm)$ for $n$ pixels and $m$ centers, parallel over $n$

• Goal:
  – Get $m$ down to constant by making some assumptions
  – Parallel over $n$ and $m$
$m$ constant

- Assume a maximal distance $r$
- Visit only those pixels in a radius $r$ around every center
- Gather-type operation:
  - For all $n$ pixels in parallel
  - Find centers closer than $r$, maybe using $k$-d trees?
• Just draw 2D disks / boxes / ... of size $r$
• This **visits** only those pixels in a radius $r$ around every center
• Scatter-type operation:
  – For all $m \cdot r^2$ pixels in parallel
  – Find minimum index using z-buffering

---

**Splats**

**z-buffer**

**id-buffer**
CODE

Programming breakout
Voronoi Diagrams