Efficient Point-Based Rendering Using Image Reconstruction

Alle Veenstra & Maurice Mulder
Efficiency in rendering point based datasets

- Splatting
  - $O(n \times a + m)$
- Image reconstruction
  - $O(n + m)$

$n$ - number of points
$m$ - size of the screen
$a$ - number of pixels per point
The motivation

Not many hardware implementations of image reconstruction

With frame buffer objects it can be done
Data Flow

1. 1-Pixel Projection for each point
2. Pull-Push Interpolation (Reconstruction)
   1. Pull Phase (create pyramid)
   2. Push Phase (use Pyramid to fill gaps)
3. Shading of projected and interpolated pixels
Data Flow

1. 1-Pixel Projection for each point
2. Pull-Push Interpolation (Reconstruction)
   1. Pull Phase (create pyramid)
   2. Push Phase (use Pyramid to fill gaps)
3. Shading of projected and interpolated pixels
1. Projecting Points to Single Pixels

Point Attributes:

- **Required Attributes:**
  - Position
  - Normal
  - Radius of Influence
    - Used by Pull-Push Interpolation
    - Corresponds to Splat Size

- **Any other attributes**
  - Color, Texture Coordinates, etc.
  - will also be interpolated.
Advantages:

- At most 1 Pixel rasterized per Point.
  - A lot less overdraw than Splatting

- Only 1 Object-order pass
  - Major Disadvantage of splatting techniques:
    They require two passes:
    - Visibility Pass
    - Attribute Pass (Blending)
Data Flow

1. 1-Pixel Projection for each point
2. Pull-Push Interpolation (Reconstruction)
   1. Pull Phase (create pyramid)
   2. Push Phase (use Pyramid to fill gaps)
3. Shading of projected and interpolated pixels
2.1 Pull-Push Interpolation: Pulling

- Pyramid is Constructed
  - Bottom Up
  - Dimensions of each higher layer reduced by a factor 2
  - Attribute values are averaged over corresponding valid pixels (See Figure)

- Pixels:
  - binary flag \{valid, invalid\}
  - Average of 4 invalid pixel is an invalid pixel
Occluded Pixels

- Pixels are invalid when:
  - no points are projected in them
  - it is occluded by other pixels

- When are pixels occluded?
- Each Pixel has a depth interval:
  - [Point's Z-coordinate, Point's Z-coordinate + radius]
- Pixel is occluded when:
  - depth-interval does not intersect with depth interval of closest pixel.
Data Flow

1. 1-Pixel Projection for each point
2. Pull-Push Interpolation (Reconstruction)
   1. Pull Phase (create pyramid)
   2. Push Phase (use Pyramid to fill gaps)
3. Shading of projected and interpolated pixels
2.2 Pull-Push Interpolation: Pushing

- Pulling Bottom-Up
- So.. Pushing Top-down
- Invalid (Unfilled and Occluded) Pixels are recalculated

- Each Pixel in a lower level has 4 pixels above it.
- Initial weight are as in figure ->

- Only Valid Pixels count!
- When 1 or more pixels are invalid weight are renormalized.
Elliptical Box Filters

- To Reconstruct Silhouette:
  - Elliptical Box Filters
    - Computed while Pulling.
  - Limits region of influence for every point
- Ellipse: Projected circle
  - Orientation defined by Normal
  - Radius is Point Attribute
  - Displacement Vectors are also stored
    - to move the ellipse center in courser pyramid levels
Elliptical Box-Filters

- Ellipses for courser Levels:
  - Normals and Radii are Averaged, Defining a new Ellipse.
  - Displacement Vectors Are also Averaged to define a new center.
2.2 Pull-Push Interpolation: Pushing
Data Flow

1. 1-Pixel Projection for each point
2. Pull-Push Interpolation (Reconstruction)
   1. Pull Phase (create pyramid)
   2. Push Phase (use Pyramid to fill gaps)
3. Shading of projected and interpolated pixels
3. Deferred Shading

- Normals and Colors have been interpolated
- So Phong can be calculated for every Pixel
GPU implementation

Ping pong rendering derivative
Uses frame buffer objects
GPU: Bottom-up pull phase
GPU: Copy phase
GPU: Top-down push phase
Results

50M-60M versus 25M-38M for splatting (P/S)

Some problems with aliasing

No good comparison in paper
## Result

<table>
<thead>
<tr>
<th>model</th>
<th># points</th>
<th>fps</th>
<th>time per frame*</th>
<th>fps</th>
<th>time per frame*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>25 K</td>
<td>82</td>
<td>12 ms (0.2 ms, 12 ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Armadillo</td>
<td>173 K</td>
<td>71</td>
<td>14 ms (1.4 ms, 13 ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Happy Buddha</td>
<td>544 K</td>
<td>61</td>
<td>16 ms (5.1 ms, 11 ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asian Dragon</td>
<td>3610 K</td>
<td>26</td>
<td>39 ms (28 ms, 10 ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thai Statue</td>
<td>5000 K</td>
<td>20</td>
<td>49 ms (38 ms, 10 ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>82 ms (39 ms, 38 ms)</td>
</tr>
</tbody>
</table>
Questions