Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering
Approximate depth-peeling for Transparency
Transparency via depth-peeling

- Standard depth-peeling
  - advantages:
    - no pre-process
    - no sort
    - suitable for per-pixel lighting
  - drawback:
    - may require several rendering passes

=> “approximate depth-peeling” ?
Approximate depth-peeling

- Idea:
  - bound the number of rendering passes
  - + approximate blending for the last layer
    - blending heuristic: $c(x) = \sum_{i} \phi'_i(x) \alpha_i c_i$
    - no deferred shading
Approximate depth-peeling
(results with 2 layers)

complete depth-peeling

only the first layer + a 2nd layer with approx blending
Approximate depth-peeling
(results with 3 layers)

complete depth-peeling

only the first 2 layers + a 3th layer with approx blending
Splat rasterization
Point Cloud Rendering

- Ray-cast a reconstructed surface (MLS)
  - Best quality but slow, requires pre-process...

- Rasterization (splatting)
  - best quality criteria:
    - perspectively correct splat rasterization
    - per-pixel shading (=> deferred shading)
    - high frequency filtering (aliasing)
  - performance criteria:
    - use the best of current GPU
    - incremental calculations for the rasterization
Splat rasterization

- Decomposed as two stages:
  - "splat setup" stage
    - compute the screen space shape of the splat
    - implemented in a vertex program
  - rasterization stage
    - generate the fragments with correct depth and weight
    - implemented in a fragment program (+ point sprite)
splat rasterization implementations

<table>
<thead>
<tr>
<th></th>
<th>perspective OK</th>
<th>EWA filtering</th>
<th>suitable for incremental computation</th>
<th># instr. setup</th>
<th># instr. raster</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>EWA Splatting [Zwicker01]</em></td>
<td>❌</td>
<td>✔</td>
<td>✔</td>
<td>software</td>
<td></td>
</tr>
<tr>
<td>[Guennebaud03]</td>
<td>❌</td>
<td>✔</td>
<td>✔</td>
<td>51</td>
<td>6</td>
</tr>
<tr>
<td>Perspec. Accu. [Zwicker04]</td>
<td>✔️</td>
<td>✔</td>
<td>✔</td>
<td>93</td>
<td>8</td>
</tr>
<tr>
<td>[Botsch05] (ray casting)</td>
<td>✔️</td>
<td>✔</td>
<td>❌</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>[PBG06]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔</td>
<td>58</td>
<td>3</td>
</tr>
</tbody>
</table>
Perspective splatting

reconstruction kernel $\phi$ (gaussian)

local splat space

screen space

$$\begin{bmatrix} xz \\ yz \\ z \end{bmatrix} = M \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s & t & p \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$
Perspective splatting

\[
\begin{bmatrix}
u \nu \\ v \nu \\ w
\end{bmatrix} =
\begin{bmatrix}
(t \times p)^T \\
(p \times s)^T \\
(s \times t)^T
\end{bmatrix} \cdot
\begin{bmatrix}
x \\ y \\ 1
\end{bmatrix}
\]
Depth value

- We have \( w = \alpha \frac{1}{z} \) and \( depth = a \frac{1}{z} + b \)

- Hence:

\[
\begin{bmatrix}
uw \\
vw \\
w \\
depth
\end{bmatrix} =
\begin{bmatrix}
(t \times p)^T \\
(p \times s)^T \\
(s \times t)^T \\
\alpha \frac{(s \times t)^T}{(s \times t)^T} p
\end{bmatrix} +
\begin{bmatrix}
0 \\
0 \\
b
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]
GPU implementation

\[
\begin{bmatrix}
uw \\
vw \\
w \\
\text{depth}
\end{bmatrix} = \begin{bmatrix}
\begin{bmatrix} t \times p \end{bmatrix} T \\
\begin{bmatrix} p \times s \end{bmatrix} T \\
\begin{bmatrix} s \times t \end{bmatrix} T \\
\begin{bmatrix} s \times t \end{bmatrix} T + \begin{bmatrix} 0 \\
0 \\
b \end{bmatrix} T
\end{bmatrix}
\]

\[
\phi(u, v) = \phi \left( \begin{bmatrix} uw \\
vw \\
w \end{bmatrix}, \begin{bmatrix} vw \\
w \\
w \end{bmatrix} \right)
\]

computed in the vertex program

2 MAD
1 projective 2D texture access
fragment program
EWA filtering

- Object-space filter only
  \[ \phi'(x) \]
  warped reconstruction kernel

- EWA filtering
  \[ (\phi' \otimes h)(x) \]
  reconstruction kernel
  low-pass filter

- OK for affine mapping only
EWA filtering approximations

<table>
<thead>
<tr>
<th>magnification</th>
<th>minification</th>
<th>magnification + minification</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="magnification" /></td>
<td><img src="image2" alt="minification" /></td>
<td><img src="image3" alt="magnification + minification" /></td>
</tr>
</tbody>
</table>

- **reconstruction kernel**
- **low-pass pre-filter**

**EWA resampling filter**

approximation used in [BHZH05]

$$\max(\phi'(x), h(x))$$
### EWA filtering approximations

<table>
<thead>
<tr>
<th></th>
<th>magnification</th>
<th>minification</th>
<th>magnification + minification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>reconstruction kernel</strong></td>
<td><img src="image1" alt="magnification" /></td>
<td><img src="image2" alt="minification" /></td>
<td><img src="image3" alt="magnification+minification" /></td>
</tr>
<tr>
<td><strong>low-pass pre-filter</strong></td>
<td><img src="image4" alt="low-pass" /></td>
<td><img src="image5" alt="low-pass" /></td>
<td><img src="image6" alt="low-pass" /></td>
</tr>
</tbody>
</table>

**Basic idea:**
Adjust the object space tangent vectors such that the warped reconstruction kernel can contain the screen space low-pass filter.
EWA filtering approximation

- Tangent vectors adjustment
  - only check along the tangent vector directions
EWA filtering approximation

• Provides the expected result if and only if the projected tangent vectors are orthogonal

  • => on the fly re-parametrization ?
    • too much expensive

  • => efficient heuristic for isotropic splats (disks):
    • \( s = p \times n \)
    • \( t = n \times s \)
    • exact at the screen center
    • exact for splats parallel to the screen plane
    • “good” worst case
About depth values and EWA filtering

Our approach: constant depth values

Previous approaches: may generate arbitrary depth values!
EWA filtering approximation
(results)

[Botsch et al. 05]

our new approximation
Performances

<table>
<thead>
<tr>
<th></th>
<th>Perspective splatting</th>
<th>raycasting [BHZK05]</th>
</tr>
</thead>
<tbody>
<tr>
<td>#instr.</td>
<td>46/3</td>
<td>34/9</td>
</tr>
<tr>
<td>visibility pass</td>
<td>58/3</td>
<td>35/13</td>
</tr>
<tr>
<td>attribute pass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>154k</td>
<td>50 (7.7)</td>
<td>33 (5)</td>
</tr>
<tr>
<td>460k</td>
<td>40 (18.4)</td>
<td>26 (12)</td>
</tr>
<tr>
<td>1.4M</td>
<td>22 (31)</td>
<td>13 (18.2)</td>
</tr>
<tr>
<td>2.5M</td>
<td>15 (37.5)</td>
<td>9.2 (23)</td>
</tr>
<tr>
<td>5M</td>
<td>6.5 (32.5)</td>
<td>5 (25)</td>
</tr>
</tbody>
</table>

screen resolution: 1024x1024
Hybrid rendering
Hybrid rendering (motivations)

- Flat surface or large zoom
  - points are inefficient (both in speed and quality)
    \[\Rightarrow\] **hybrid rendering**
    points and polygons are complementary

use **triangles** when **points** become less efficient

- What about the transitions?
Hybrid rendering (transition smoothing)

- Key idea: use the sum of weights coming from the splatting to blend the representations.

- straightforward!
- no additional rendering cost!

| standard splats & polygons rendering | splatting + \( \sum \)weights | hybrid rendering with alpha-blending |
Hybrid rendering
(transition smoothing)

- Too much straightforward?
- Best quality => uniform sampling of the “transition edges”
Hybrid rendering
(implementation example)

- Multi-resolution hierarchy of points
  - Leaves store both points and polygons
- At the sampling time:
  - explicitly sample the edges shared by two faces stored in two different leaves
Hybrid rendering (implementation example)

- Hybrid rendering rules:
  - render the polygons (instead of the splats) of all visible and not dense enough leaf node.
  - render the transition splats shared by at least one leaf rendered as a set of splats
Conclusion

• Summary:
  • Approximate depth-peeling for efficient transparency
  • Perspectively correct splat rasterization
    • efficient on current GPU
    • allows efficient dedicated implementation (incremental computation)
  • EWA filtering approximation
    • same quality as full EWA filtering
    • only for isotropic splats
  • Splat/polygon transitions smoothing
• Ray-casting -> splatting -> EWA splatting
• splat rasterization, 2 class of approaches:
  • perspective approx
    • match the center or the contour (better)
    • allow EWA filtering (by an analytic convolution)
    • expensive splat setup
    • suitable for incr. rasterization
  • ray casting
    • simple to implement
    • perspective correct
    • simple splat setup (all the computation are performed at the fragment level)
    • expensive rasterization shader
EWA filtering approximation

- Basic idea:
  - Adjust the tangent vectors $s$ and $t$ such that the warped reconstruction kernel can contain the screen space low-pass filter.
  - Adjust the tangent vectors $s$ and $t$ such that their screen space length are greater than the radius of the screen space low-pass filter.
  - OK if and only if the tangent vector are still orthogonal in the screen space and the low pass filter is radially symmetric.