Screen-Space Triangulation for Interactive Point Rendering

Reinhold Preiner

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Motivation

- High-quality point rendering mostly implies some kind of continuous surface reconstruction

- Using Point Properties for rendering
  - point normals (local surface orientation)
  - splat radii (connectivity)
Motivation

- Huge point clouds: time-consuming Preprocessing

- St. Stephans Cathedral
  460 Million Points
  Normal Est.: \(~ 17h\)

- Domitilla Catacomb
  1.9 Billion Points
  Normal Est.: \(~ 21h\)
Motivation

- Can we achieve comparable quality on rendering without precomputed attributes?

- **Our Approach:** Reconstruct normal and connectivity info on-the-fly during rendering in screen-space on the GPU

Advantages

- No time-consuming preprocessing
- Saves memory for storing attributes (normals, radii)
- Rendering/Reconstruction independent from data layout (Hierarchical, Out-Of-Core, …)

Possible Applications

- Fast on-site preview of scanned point clouds
- Instant rendering of 4D point streams
Overview of Our Approach

- **Input:** Point data projected to screen
  - Position
  - Color (optional)

- **Output:** Reconstructed frame buffers
  - Depth
  - Normal
  - Color (optional)

- Use for further deferred shading, illumination, …
Surface reconstruction - FAQ

- Given a surface sample → which neighbors to use for reconstruction?
  - KNN, FDN, …
  - robust statistics, LMS …
Surface reconstruction - FAQ

- Given a surface sample → which neighbors to use for reconstruction?
  - KNN, FDN, …
  - robust statistics, LMS …

- Given a local neighborhood → how to reconstruct surface?
  - surface fitting, forward search, …
Surface reconstruction - Our Approach

- Given a surface sample → which neighbors to use for reconstruction?
  - KNN, FDN, …
  - robust statistics, LMS …
  → Screen-Space Nearest Neighbor Search

- Given a local neighborhood → how to reconstruct surface?
  - surface fitting, forward search, …
  → Normal Estimation & Triangulation
Input:
- projected point buffer
- Initial search radius $r$

How to quickly **find** and **store** $k$ nearest neighbors of each point $Q$ in the input buffer?

Divide screen space region around $Q$ in **8 segments**

Storing nearest neighbor of each segment in 2 RGBA Textures
Screen-Space Nearest Neighbor Search

- Pass 1: for each \( P_i \), render search splat of radius \( r \)
  - Store min. world space distances \( d_{\text{min}} \) at pixel \( Q \)
- Pass 2: Render Search Splats \( P_i \) again
  - Compare distance \( P_i Q \) with saved \( d_{\text{min}} \)
Normal Estimation

- Lookup the neighbor points and calculate normal

![Diagram of neighbor points and buffers](image)

- Neighbor Buffers
- Point Position Buffer
- Point Normal Buffer
Triangulation in Geometry Shader

- **Sparse Input Buffers**
  - Depth
  - Normal
  - Color

- **Neighbor Buffers**

- **Final Buffers**
Temporal Coherence Depth Culling

Input Points

Depth Buffer

REPROJECT

Depth Culled Input Points

\[ i \]

\[ i+1 \]
Search Radii

- Maintain a search radius buffer
- Adapt radii over time
- Start with initial search radius $r_0$
- Define increase factor $\alpha > 1$

Frame i:

if #neighbors too small (e.g. < 3)

$$r_{i+1} = r_i \times \alpha$$

else

$$r_{i+1} = \max(\text{distance}(\text{neighbor}_k)), k = 1 \ldots 8$$
Algorithm - Summary

1) Project points to screen
   - Depth cull with depth buffer from previous frame
2) Update search radii
3) Perform neighbor search
4) Normal estimation
5) Triangulation
Results

Comparison to point splatting

Box Splatting  Gauss Splatting  SST
Results

Gauss Splats (Precomputed)  SST
Results

Normalestimation only locally $\rightarrow$ noise sensitive

precomputed Normals

SST Normals
Conclusion / Outlook

- Interactive rendering without precomputation
- Quality comparable to Gauss splats

Drawbacks
- Temporal Coherence Artifacts, Flickering
- Some degrees of freedom ($r_0$, $\alpha$)
- Normal estimation only local

Future Work
- Introduce denoising of normals by geometry-aware filter
- Estimate absolute radii per frame (get rid of TC, $r_0$ and $\alpha$)
- $\rightarrow$ instantly estimate good splat radii $\rightarrow$ Gauss splatting?