What about no-triangulation?

→ Point-based Rendering (Just a Quick View…)

CS334

Daniel Aliaga

Point-Based Rendering

- Options:
  - Ray tracing
  - Polygon rendering
  - **Point-based rendering**

- Instead of drawing triangles, just draw lots of dots (or small circles, or something)
- What are the advantages of this?
- What problems do you need to solve?

(a)
Points

(b)
Polygons – same number of primitives as (a)
Same rendering time as (a)

(c)
Polygons – same number of vertices as (a)
Twice the rendering time of (a)

Some History

- The idea of splatting points onto the screen originated with Levoy and Whitted in 1985
- It found some use in volume rendering
  - Voxels were accumulated along rays and onto the screen
- A commonly cited commercial system is Animatek's *Caviar* player
  - Interactive frame rates for complex models aimed at games
- Image-based rendering can be viewed as point-based rendering

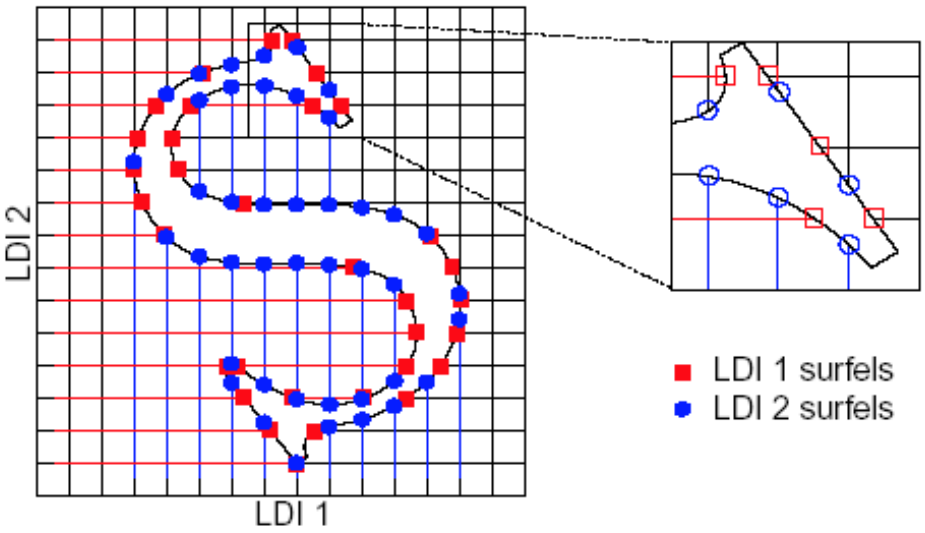
Surfels

(by Pfister, Zwicker, van Baar and Gross)

- We've seen pixels, voxels, texels, and now **surfels**
  - You can probably guess what it means
- This paper focuses on the issues of:
  - Sampling other representations into surfels
  - Storing surfels

Sampling Objects

- The final pixel resolution determines the sampling density
  - Want at least one sample per pixel
  - Typically go higher
- Cast rays through the object in an axis aligned direction on a regular grid spacing
  - Do this for each of three axis align directions
- Store pre-filtered texture colors at the points
  - Project the surfel into texture space and filter to get a single color



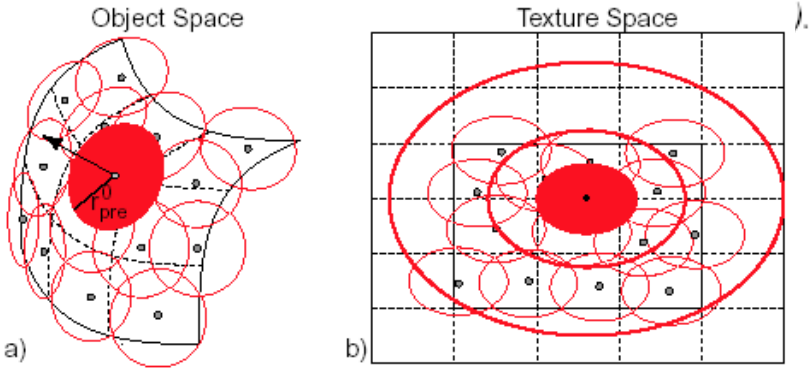
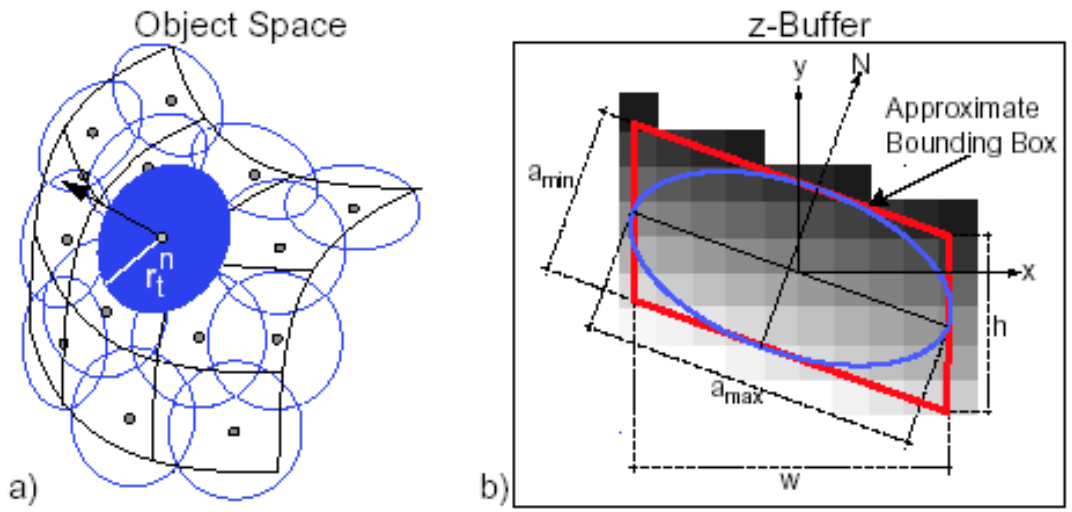


Figure 4: *Texture prefiltering with tangent disks.*

Storing Surfels

- Store 3 layered depth images (LDI), one for each axis-aligned orthographic viewing direction
  - Call the result a Layered Depth Cube (LDC)
  - A layered depth image (LDI) stores multiple depths per pixel, with color for each depth
- Build an octree hierarchy by down-sampling high-res LDIs
  - Nodes in the tree are called *blocks*
- Can also reduce a LDC to a single LDI (but with some error)

Splatting and Reconstruction



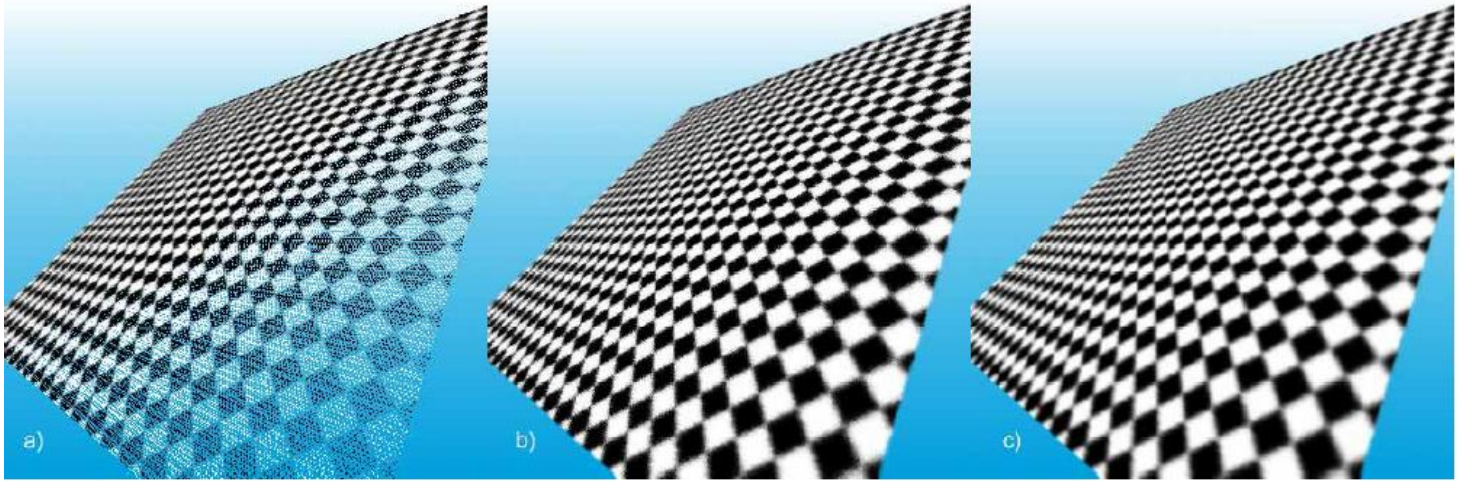


Figure 11: *Tilted checker plane. Reconstruction filter: a) Nearest neighbor. b) Gaussian filter. c) Supersampling.*

QSplat

(by Rusinkiewicz and Levoy)

- Primary goal is interactive rendering of very large point-data sets
- Built for the Digital Michelangelo Project

QSplat Sphere Tree

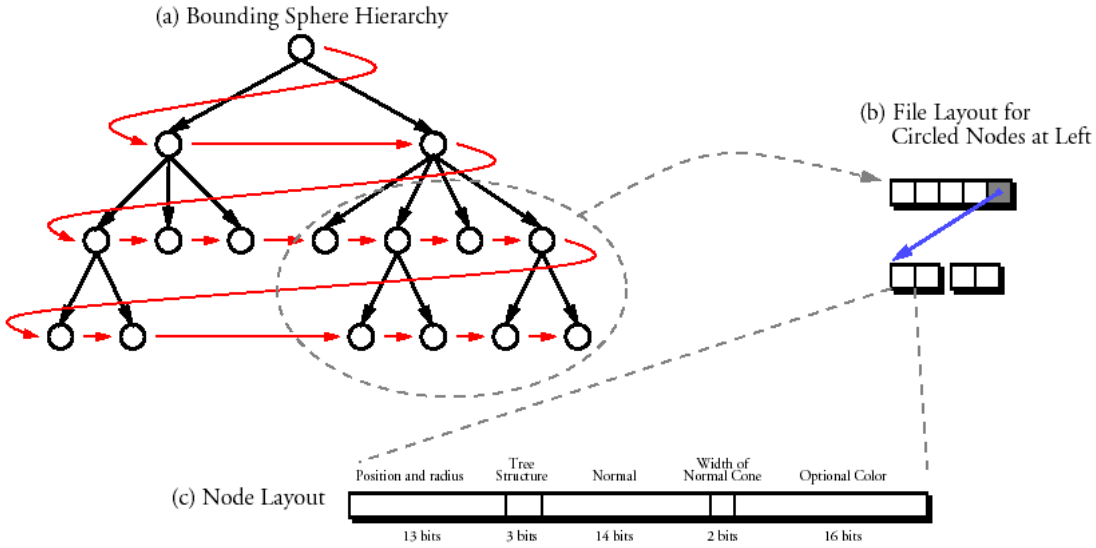


Figure 2: QSplat file and node layout. (a) The tree is stored in breadth-first order (i.e., the order given by the red arrows). (b) The link from parent to child nodes is established by a single pointer from a group of parents to the first child. The pointer is not present if all of the "parent" siblings are leaf nodes. All pointers are 32 bits. (c) A single quantized node occupies 48 bits (32 without color).

Splat Shape

- Several options
  - Square (OpenGL “point”)
  - Circle (triangle fan or texture mapped square)
  - Gaussian (have to do two-pass)
- Can squash splats depending on viewing angle
  - Sometimes causes holes at silhouettes, can be fixed by bounding squash factor

Splat Shape

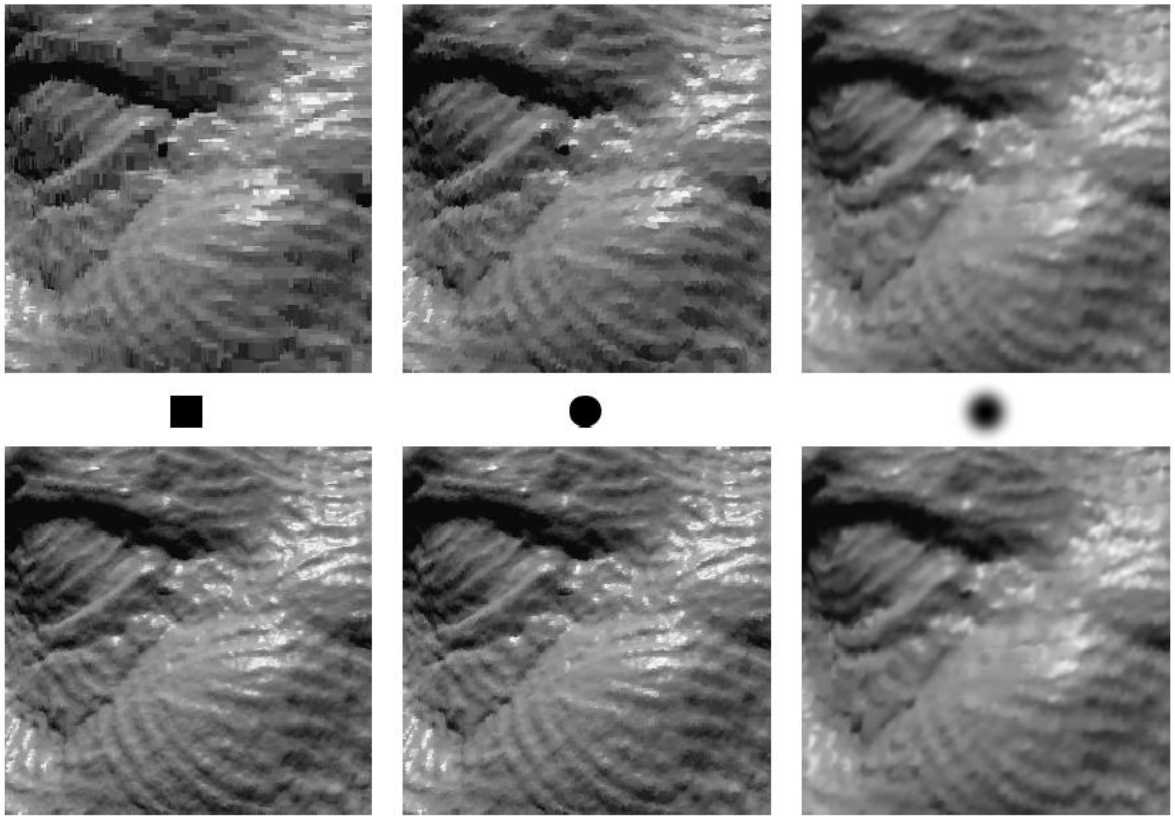
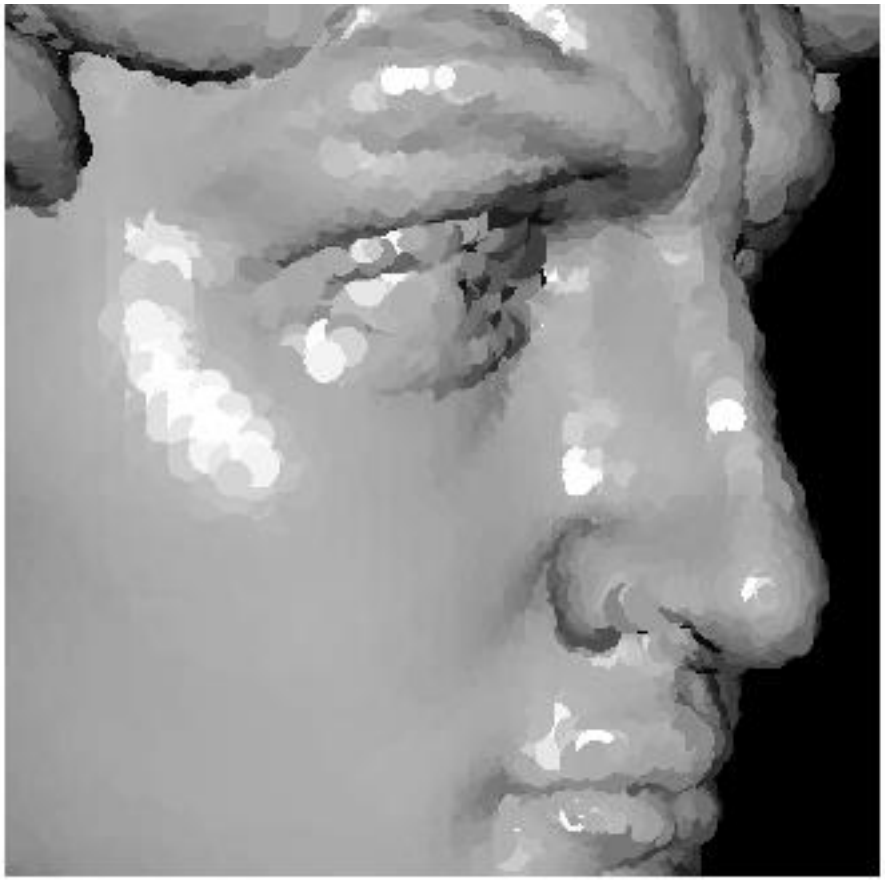
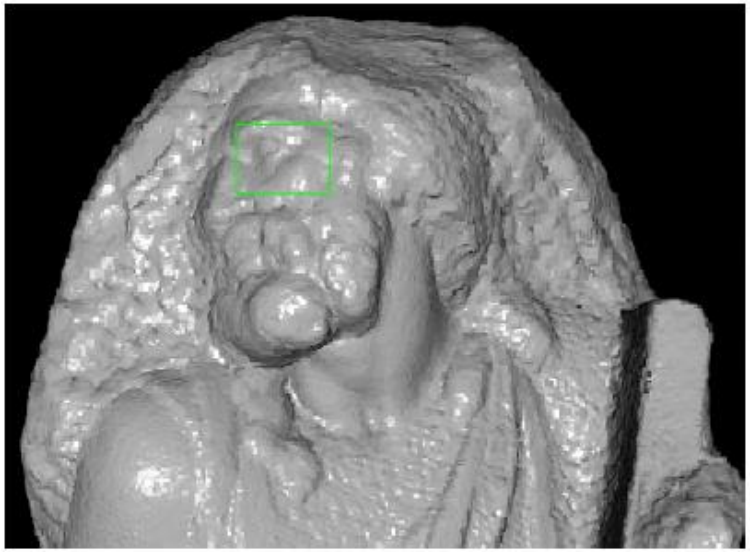


Figure 3: Choices for splat shape. We show a scene rendered using squares, circles, and Gaussians as splat kernels. In the top row, each image uses the same recursion threshold of 20 pixels. Relative to squares, circles take roughly twice as long to render, and Gaussians take approximately four times as long. The Gaussians, however, exhibit significantly less aliasing. In the bottom row, the threshold for each image is adjusted to produce the same rendering time in each case. According to this criterion, the square kernels appear to offer the highest quality.

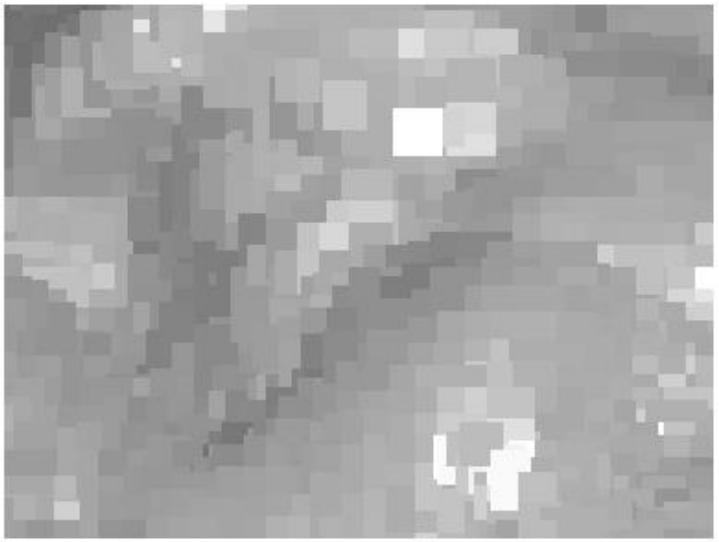
Splat Silhouettes

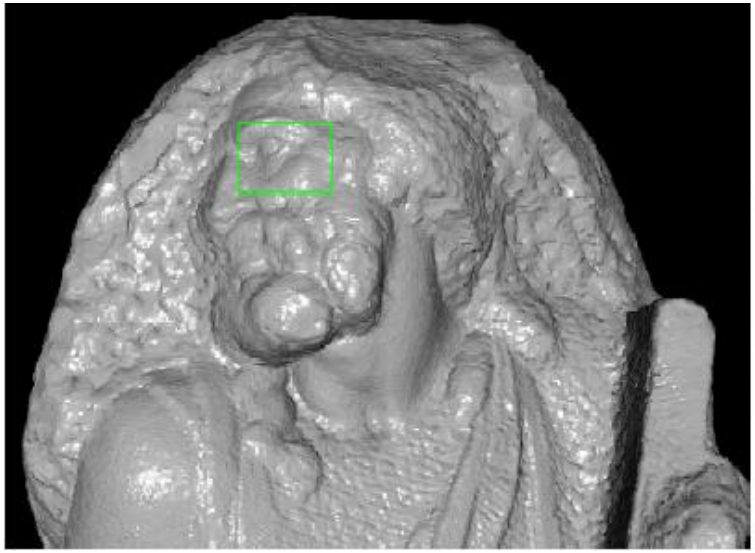


Few Splats

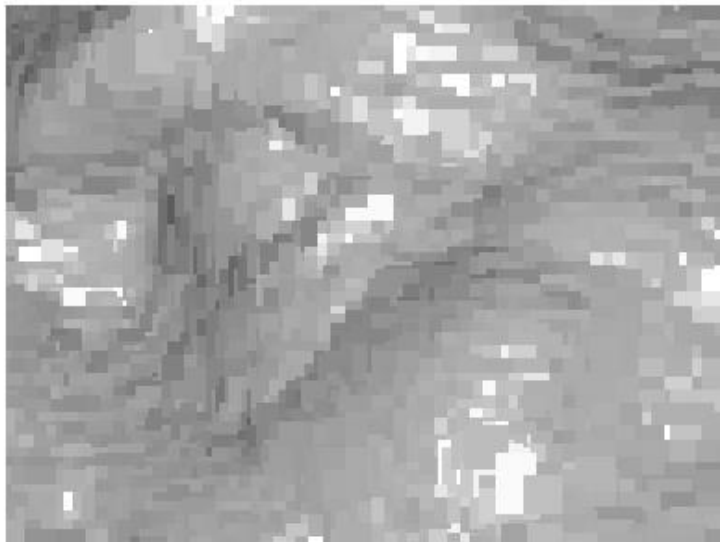


15-pixel cutoff
130,712 points
132 ms

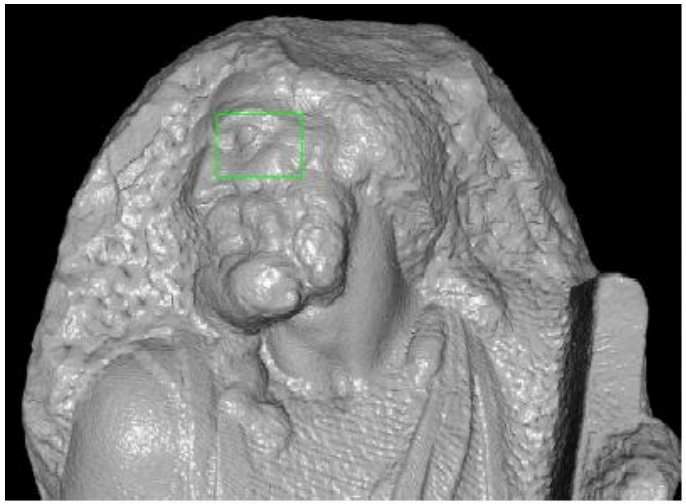




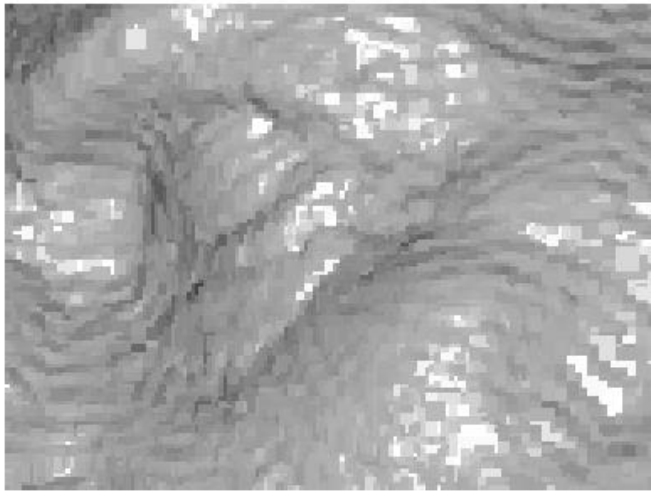
10-pixel cutoff
259,975 points
215 ms

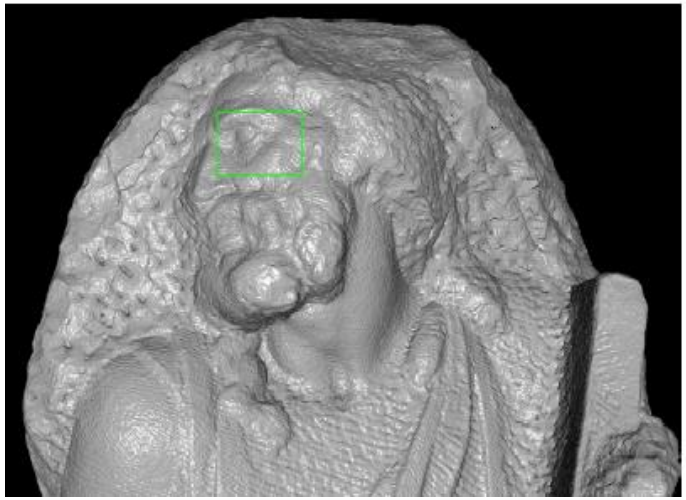


Many Splats

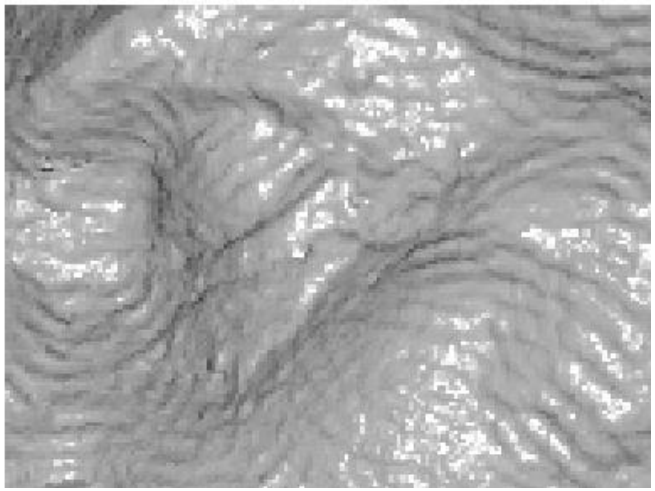


5-pixel cutoff
1,017,149 points
722 ms





1-pixel cutoff
14,835,967 points
8308 ms



Frameless Rendering

- Continuously update pixels in randomized order
- Reconstruct image with filtering based on recent pixels
- Many guises: frameless rendering, the render cache, radiosity interpolants
- Think raytracing:
  - As things change from frame to frame, cast as many rays as you have time for, and keep other rays from previous frame
- Hard parts are knowing which samples to keep/discard, and filtering
- “Adaptive Frameless Rendering”
  - Dayal et al. 2005
  - Video