State of the Art in Data Representation for Visualization:

Surface Points and Images

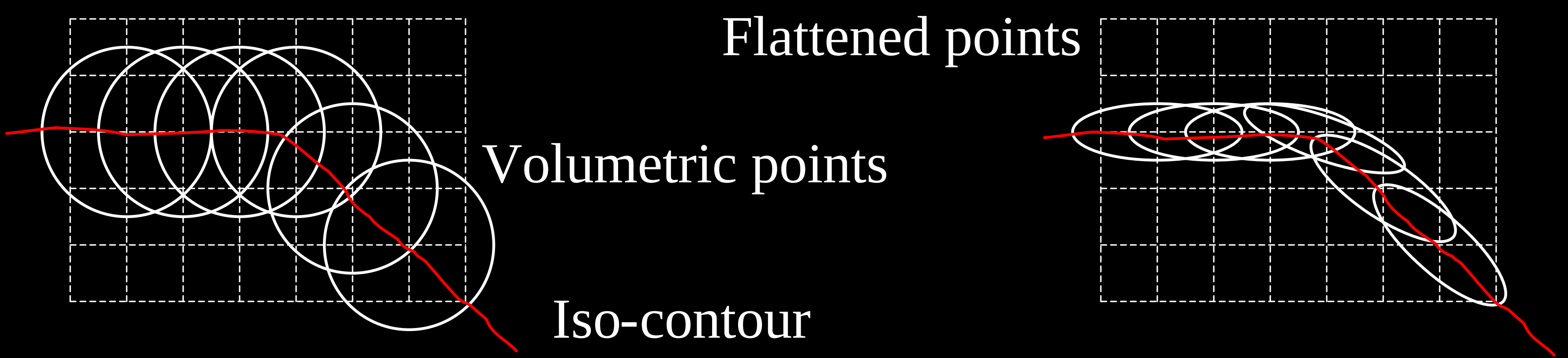
Klaus Mueller

Stony Brook University

Computer Science   Center for Visual Computing

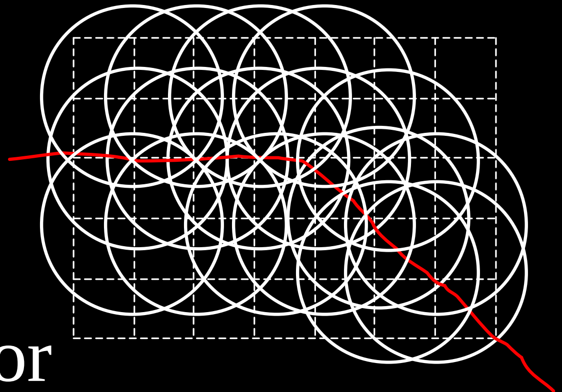
From Volume To Surface

- Iso-surface of a volumetric point-based object is represented by a hull of Gaussian kernels
- Flattening the points in direction of the surface normals yields a more exact representation
- In the limit get a surface composed of 2D Gaussians (aka surface points or surfels)

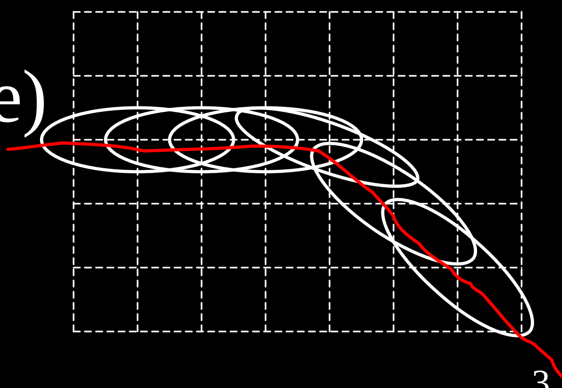


Volumetric vs. Surface Points

- Volumetric points:
  - Most often on a regular lattice
  - Represent both boundary and interior
  - Overlapping points reconstruct volumetric object
  - Different iso-surfaces, shapes, and compositions can be produced via transfer functions on the fly

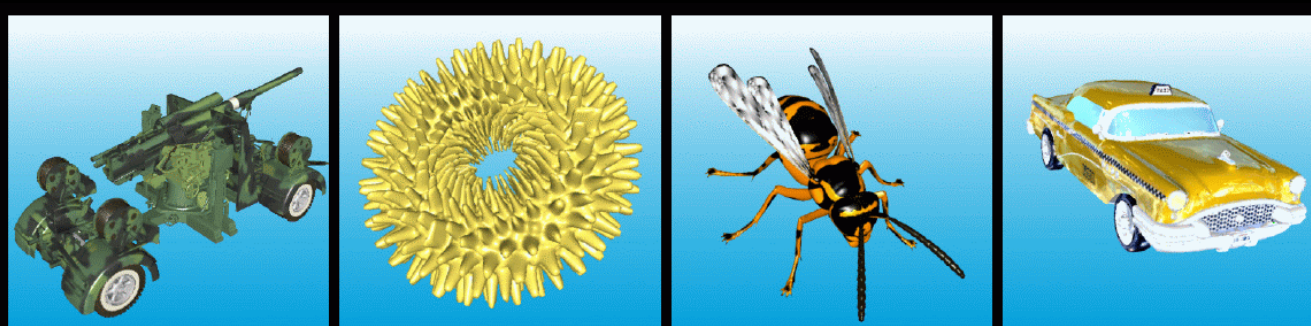


- Surface points:
  - Irregular distribution (on the surface)
  - Usually located only on boundaries



Points vs. Triangles (1)

- Points are favorable when the geometric detail is less than the size of a pixel (triangle size < 1)
  - Intricate objects with great geometric detail
  - Minified objects when reduced detail reaches the resolution of the screen
- In these cases overhead for triangle rasterization and shading is overkill

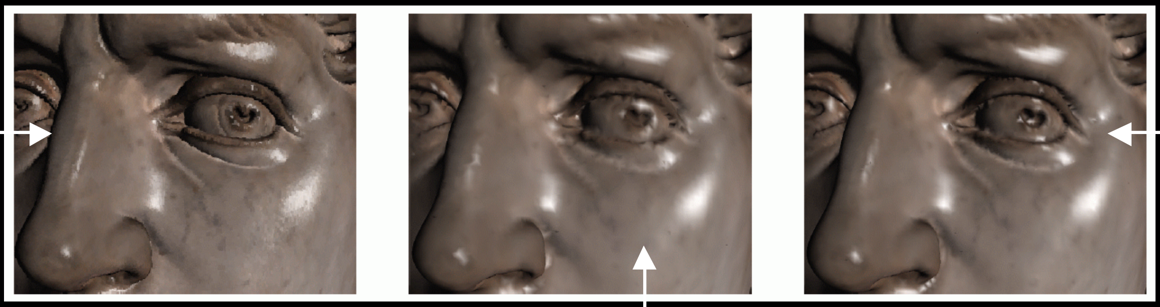


from Pfister '00

Points vs. Triangles (2)

- Polygons are better when the image resolution is greater than the projected geometry resolution
  - Flat or slightly curved surfaces
  - Magnified objects
- In these cases triangle rasterization is more efficient and yields better quality

Points



Same # of vertices
2x the rendering time

Same # of polygons,
same rendering time

from Rusinkiewicz ‘00

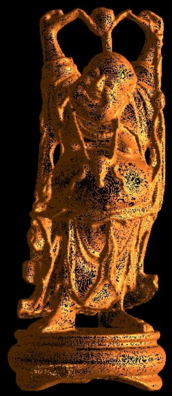
Points vs. Triangles (3)

- Hybrid solutions have been proposed (Chen ‘01, Cohen ‘01)
  - Use triangles for magnified and original detail
  - Use points for minified object portions

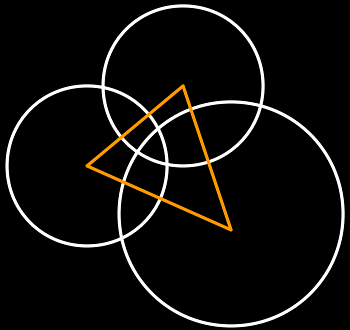
250,279 pts, 16 tris, 4.58 fps — 261,925 pts, 57,029 tris, 3.16 fps — 29,076 pts, 274,802 tris, 3.19 fps

from Chen ‘01

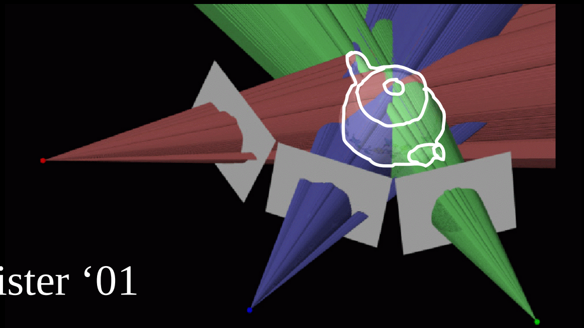
Surface Point Generation



- Point cloud output by laser range scanners
  - Natural source of surface points
  - No tedious geometry generation required



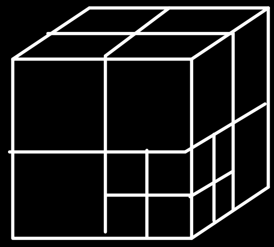
- Geometry-to-surfel conversion
  - Replace vertices by overlapping points
- Volume-to-surfel conversion
  - Go from Marching Cubes directly to surface points
- Image-Based Visual Hull
  - From object silhouettes to points



from Pfister '01

Surface Points Pioneers

- Geometric subdivision leads to points (Catmull ‘74)
- Particle systems (Reeves ‘83, ‘85, Szeliski ‘92)
- Points as a display primitive (Levoy ‘85)
- Dividing Cubes (Cline et al., ‘88)
  - Used to extract iso-surfaces from volume datasets
  - Create “surface points” instead of triangles
  - Trilinearly interpolate normals from grid points
  - Subdivide volume cells until extracted surface points are pixel-size

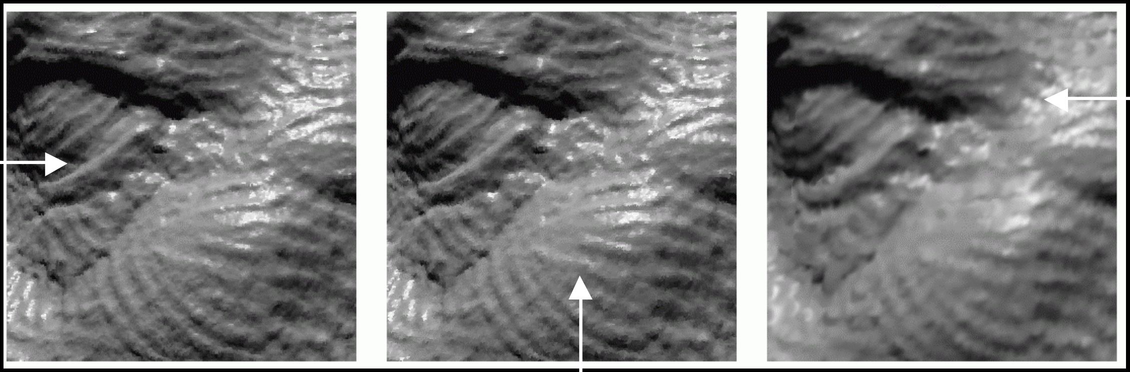


More Recent Work

- Layered Depth Images (Shade ‘98)
- Point-based rendering system (Grossman ‘98)
- Surfels (Pfister ‘00)
- Qsplat (Rusinkiewicz ‘00)
- Raytracing of point-based geometry (Schauffler ‘00)
- Point set surfaces (Alexa ‘01)
- Surface Splatting (Zwicker ‘01)
- Opacity Hulls (Matusik ‘02)

Issues

- What shape does a point have?
  - Small dot, square, rectangle, circle, ellipse,…



from Rusinkiewicz ‘00

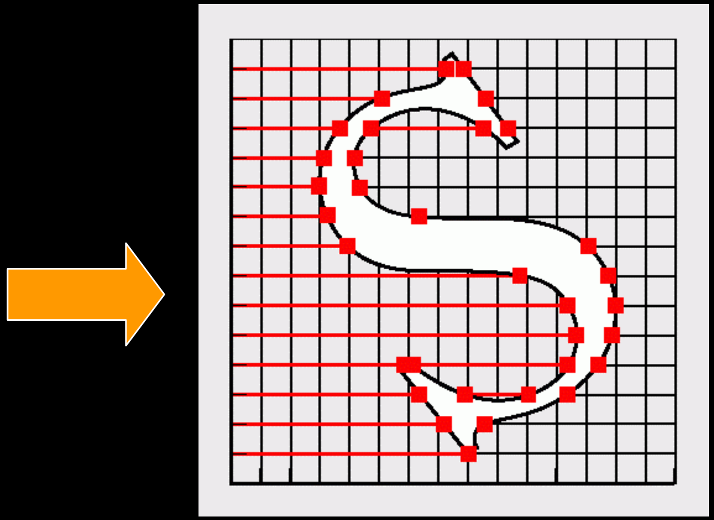
- How to deal with the lack of connectivity?
  - Blending, visibility and occlusion, holes,…
- How to adapt to local resolution?
  - Hierarchical representation for level-of-detail

Data Structure

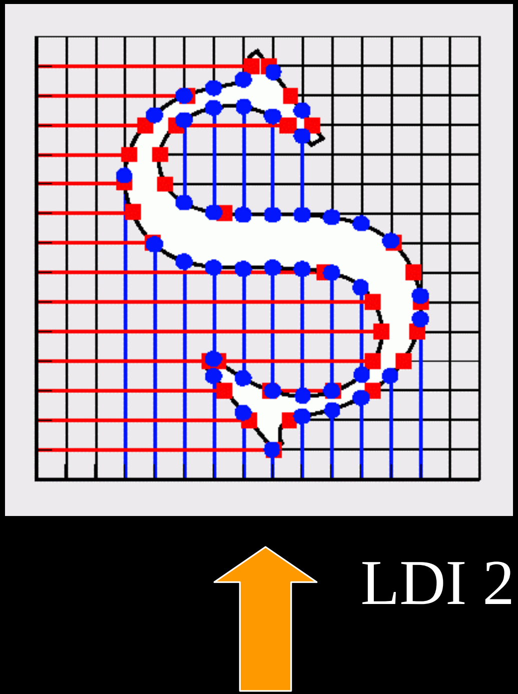
- A popular points data structure is the LDC tree
  - LDC tree = Layered Depth Cube tree
- The LDC tree is a hierarchical LDC
- An LDC consists of 3 orthogonal LDIs (Lischinski ‘98)
  - LDI = Layered Depth Image (Shade ‘96)
- Strategy:
  - Acquire a set of LDIs
  - Merge LDIs into one LDC
  - Calculate the LCD tree

LDI Acquisition

- For each LDI, cast a set of parallel rays
  - or use visual hull
- Calculate the ray-object intersections (depths)
- Store the depths into the LDI



LDI 1

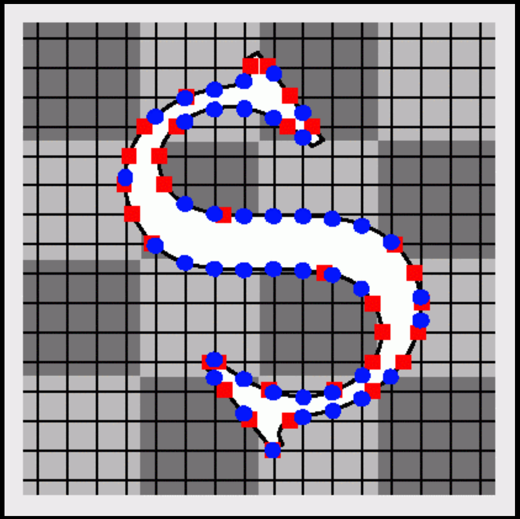


LDI 2

from Pfister ‘00

LDC Construction

- Partition the LDC into square blocks 16 LDI pixels large
- Merge all LDIs into the LDC

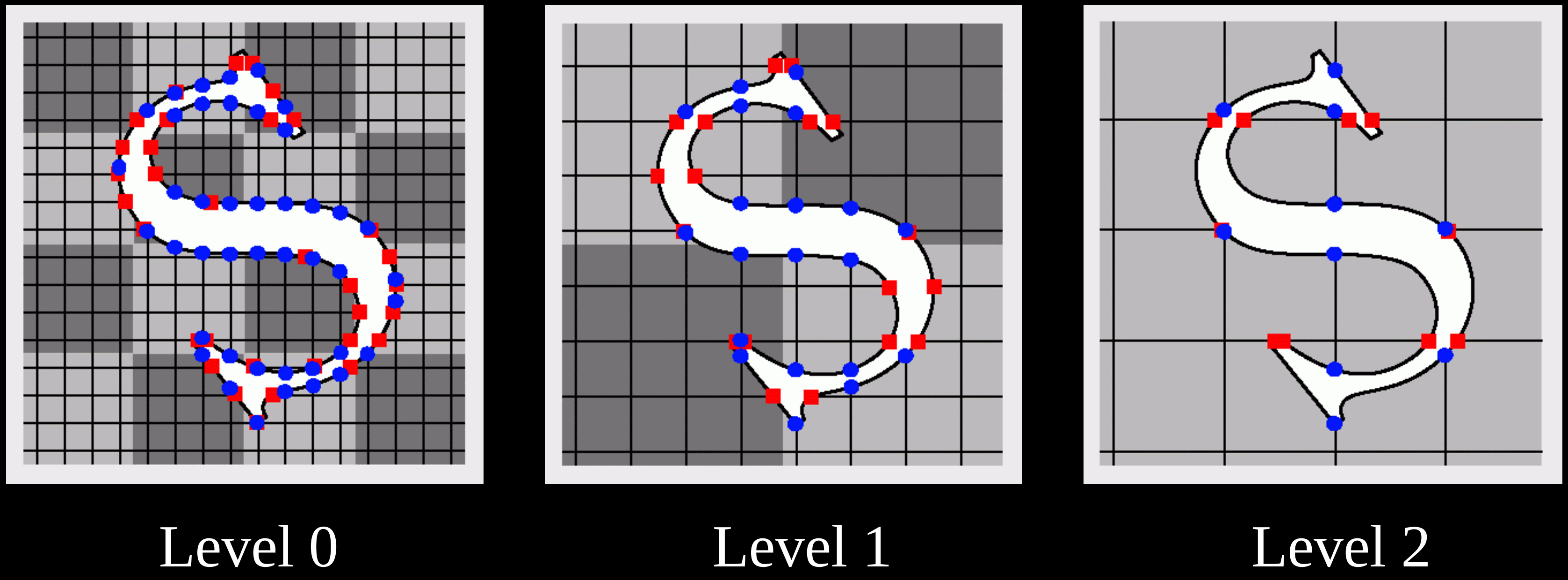


- This forms level 0 of the LDC tree

from Pfister '00

LDC Tree Construction

- Progressively sub-sample the level 0 LDC



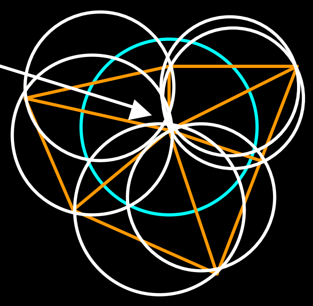
- Samples on upper level LDCs are also present in lower levels LDCs

from Pfister ‘00

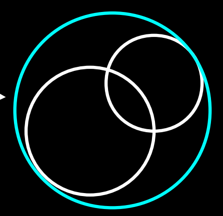
Alternative Data Structures

- Qsplat:

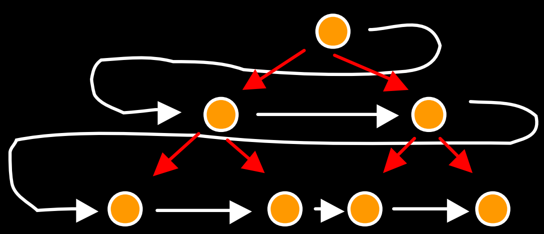
Sphere at vertex v = largest bounding sphere of the faces sharing v



  - Start from a triangular mesh and compute a sphere for each vertex
  - Ensure good overlap among neighboring spheres
  - Build a hierarchy of spheres with a recursive algorithm (two spheres merge into their bounding sphere)



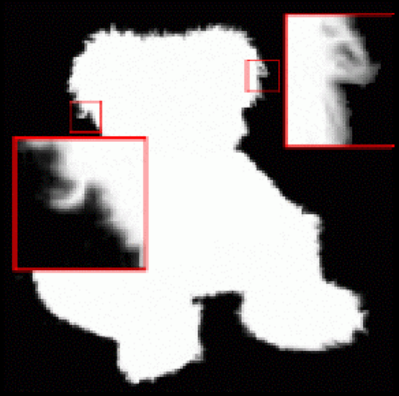
  - Quantize both radius and position of spheres
  - Store tree in breadth-first order to exploit locality of resolution



storage order

Associated Appearance Data

- Given in the form of images
  - Textures
  - Radiance images (object photographs)
  - Alpha (opacity) hulls (Matusik ‘02)

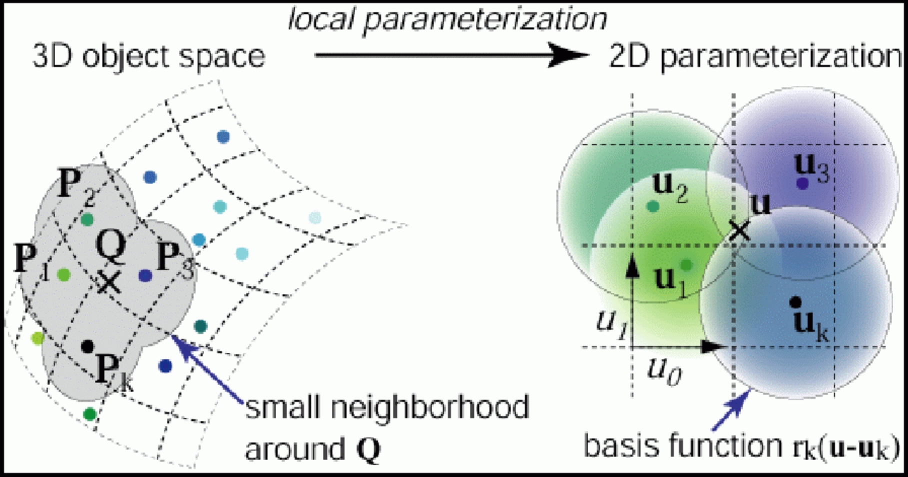


  - Reflectance field (for re-illumination) (Debevec ‘00)
- Distinguish
  - View-independent point coloring (texture mapping)
  - View-dependent point coloring (lumigraph, lightfield) (Levoy ‘96, Gortler, ‘96, Buehler ‘01)

View-Independent Coloring

- The texture function on the object surface given by:

$$f_c(u) = \sum_{k \in N} w_k r_k(u - u_k)$$



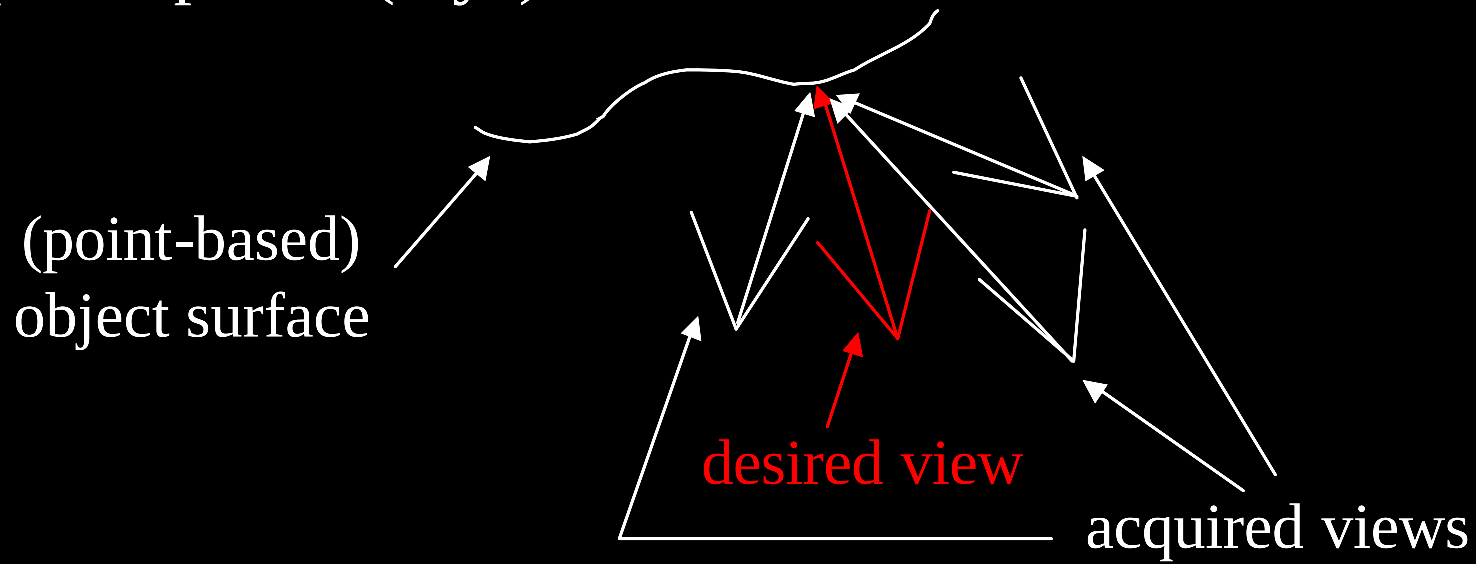
r_k = basis function (reconstruction kernel)

- Find the w_k via an optimization procedure:
  - Warp the (isotropic) texture reconstruction kernels from texture space into object surface space
  - Find the w_k through an error minimization procedure

from Zwicker '01

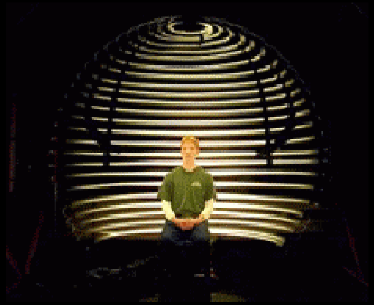
View-Dependent Coloring (1)

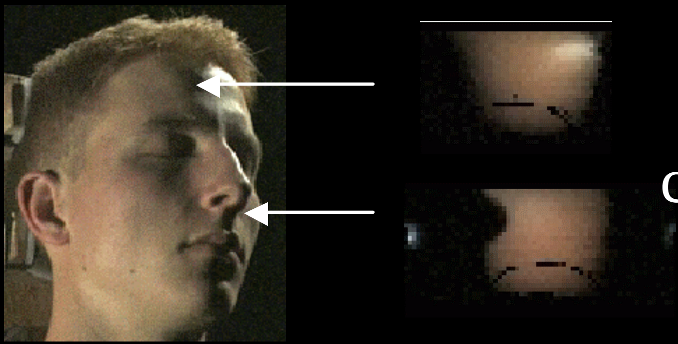
- Unstructured lumigraph:
  - Collection of radiance images taken from many different viewpoints
  - Point color = weighted sum of the *n* closest acquired pixels (rays)



View-Dependent Coloring (2)

- Reflectance fields (Debevec '00):
  - Collection of radiance images taken from
    - many different viewpoints *and*
    - under different illumination directions



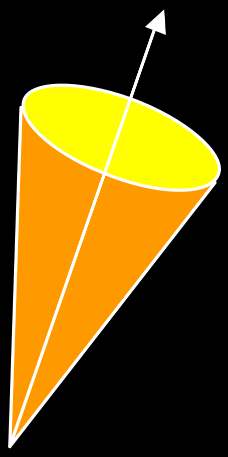


Two image pixels viewed from a constant location and lit from a number of different light source locations

- Use these "illumination basis functions" to re-light the object from a new set of light sources
  - This results in a new set of radiance images

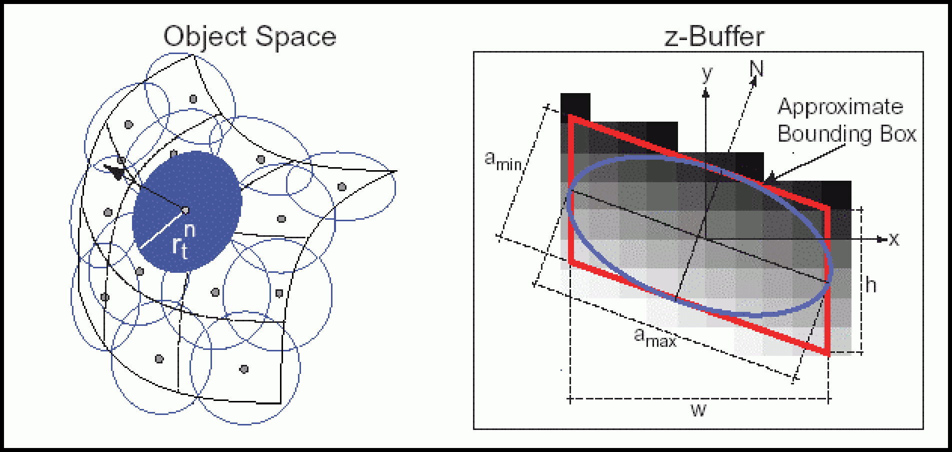
Point Rendering

- Traverse point hierarchy top to bottom
- Cull point blocks outside the viewing frustum
- Project points from object to screen space
  - Use fast incremental forward warping (Grossman '98)
  - Visibility cones to cull blocks with backfacing points
- Three-pass algorithm (Surfels, Qsplat):
  - 1. Project points into the z-buffer
  - 2. Blend colors of visible points, fill holes
  - 3. Shade in image space



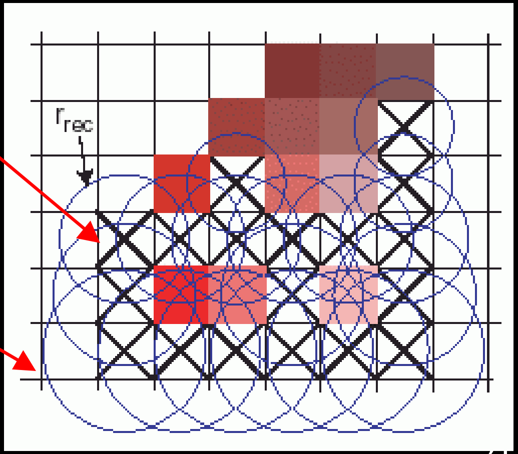
Surfels and QSplat

- Traverse point hierarchy from top to bottom
  - For each block of points, find level where the local resolution of the projected point set matches the screen resolution (oversample for better quality)
- Splat the selected points into the z-Buffer
- Blend visible points, fill holes, shade



hole

reconstr. filter



from Zwicker '01

Surface Splatting - Concepts

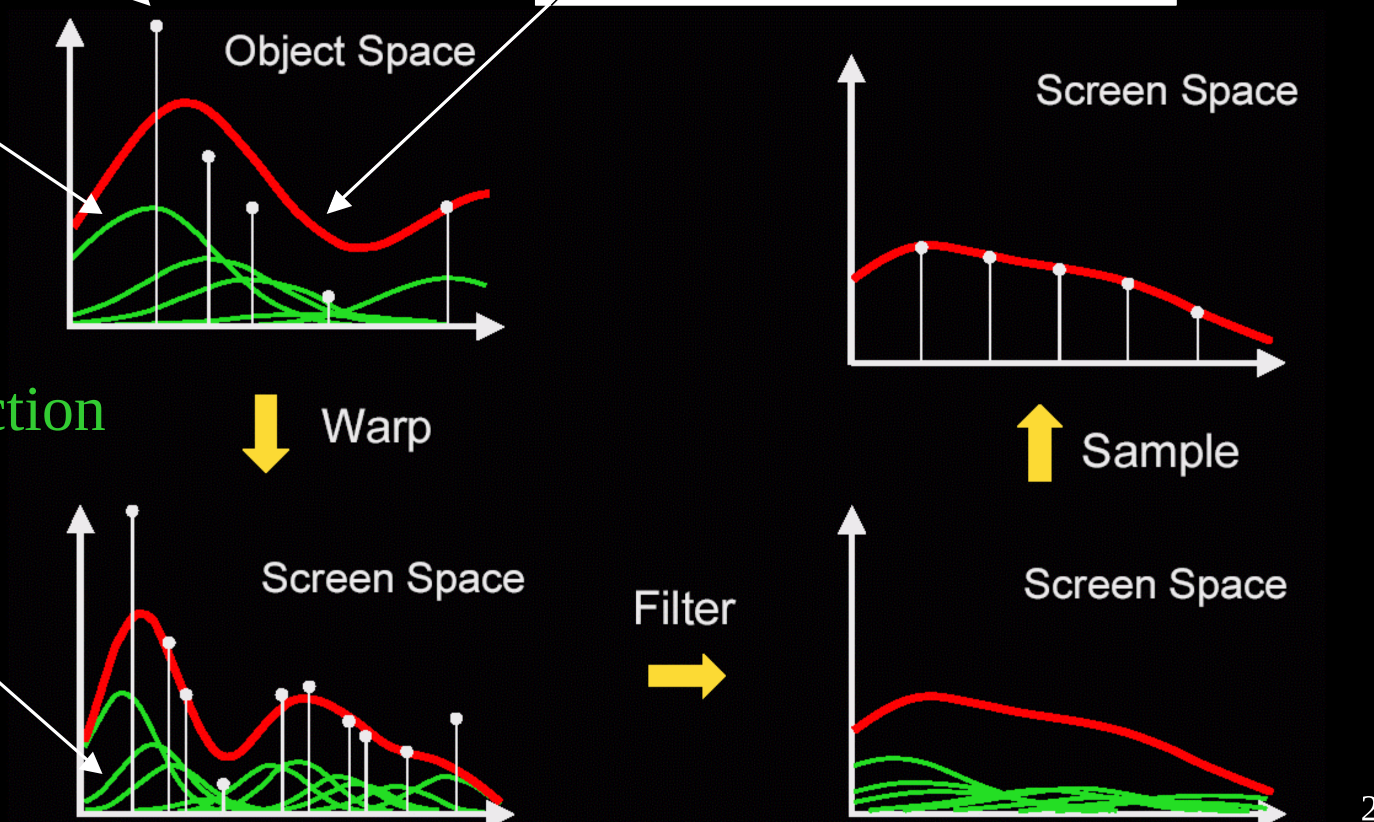
- High quality texture reconstruction in screen space

$$f_c(u) = \sum_{k \in N} w_k r_k (u - u_k)$$

point color w_k

basis function r_k

warped basis function
= screen footprint



Surface Splatting - Algorithm

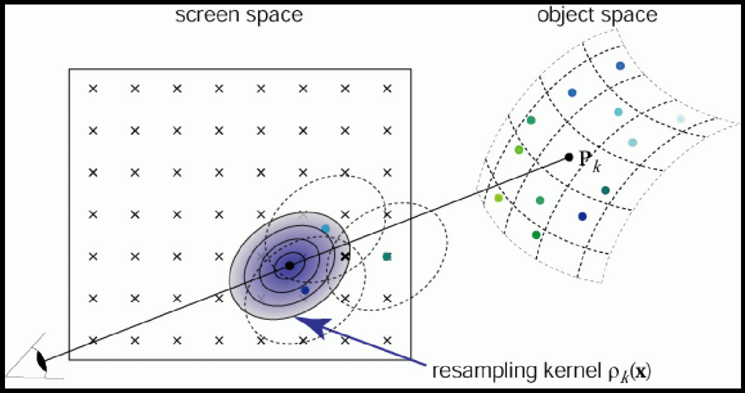
- Enable z-buffer, determine front-facing points
- For each point:
  - Determine resampling kernel =
    screen footprint ⊗ screen lowpass filter
  - Rasterize resampling kernel to screen
  - Perform z-buffer test on each fragment
  - Accumulate normals and texture colors
- Deferred shading
  - Shade pixels after all points have been projected
  - Use filtered normals and blended texture colors

Surface Splatting - EWA Filter

- The resampling kernel is called EWA filter:
  - It combines the Gaussian footprint with a screen space Gaussian low-pass filter
  - Analytical formulation:

footprint   screen space filter

$$g(x) = (q \otimes h) = \frac{1}{\left|W^{-1}\right|} G\left(W^T V^q W + V^h\right)(x - C)$$



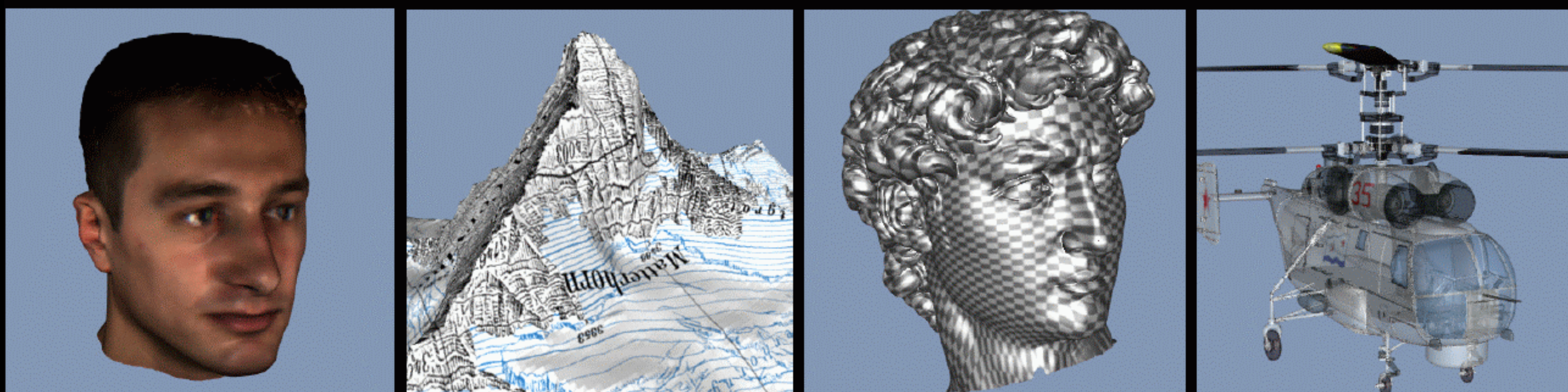
W: Warp and projection matrix

V^q: 3D reconstruction kernel

from Zwicker ‘01

Results

- Surface Splatting:
  - Transparencies via a layered z-buffer technique



- Surface Splatting with opacity hulls:
  - Composite semi-transparent points
  - Use for fuzzy objects, fur and feathers

from Zwicker ‘01, Matusik ‘02

Questions?