State of the Art in Data Representation for Visualization:

Volumetric Points

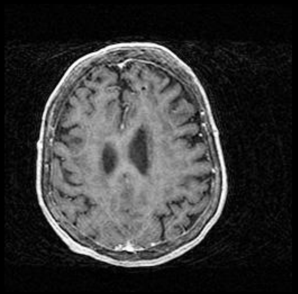
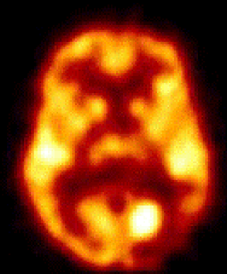
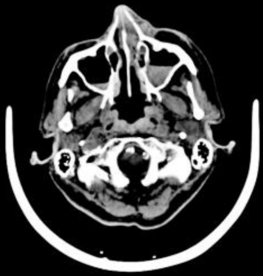
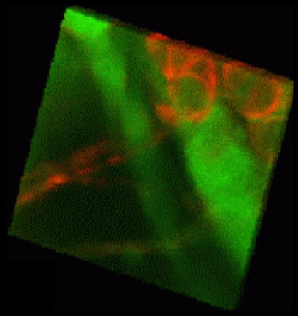
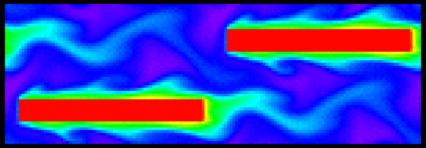
Klaus Mueller

Stony Brook University

Computer Science Center for Visual Computing

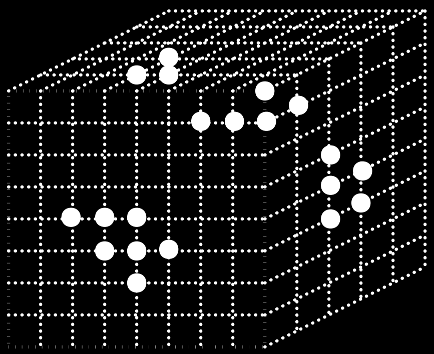
Data Sources

- Volumetric sampling modalities
  - Medical scanners (MRI, CT, PET, SPECT, fMRI)
  - Industrial and security (CT)
  - Biology (confocal and electron microscopy)
  - Computational science (CFD, FE, FD)
  - Seismic devices (oil, precious metals, earthquake)
  - Engineering and industrial design (CAD/CAM)

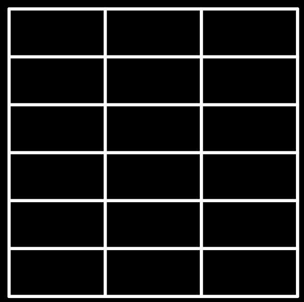
Fundamental Representation

- Volumetric objects are sampled into points, arranged in some 3D grid raster:

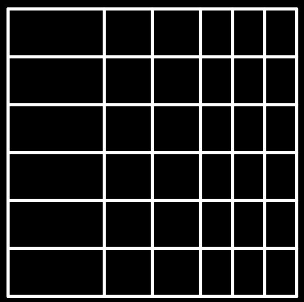


cubic grid

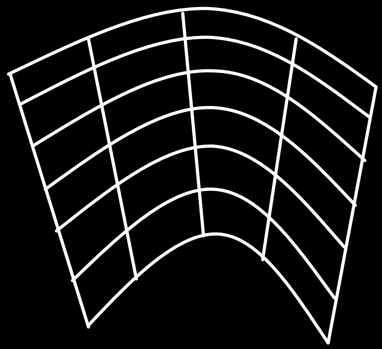
- Other common grids:



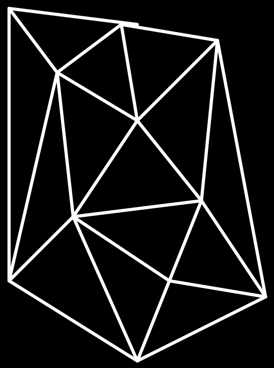
anisotropic rectilinear



rectilinear



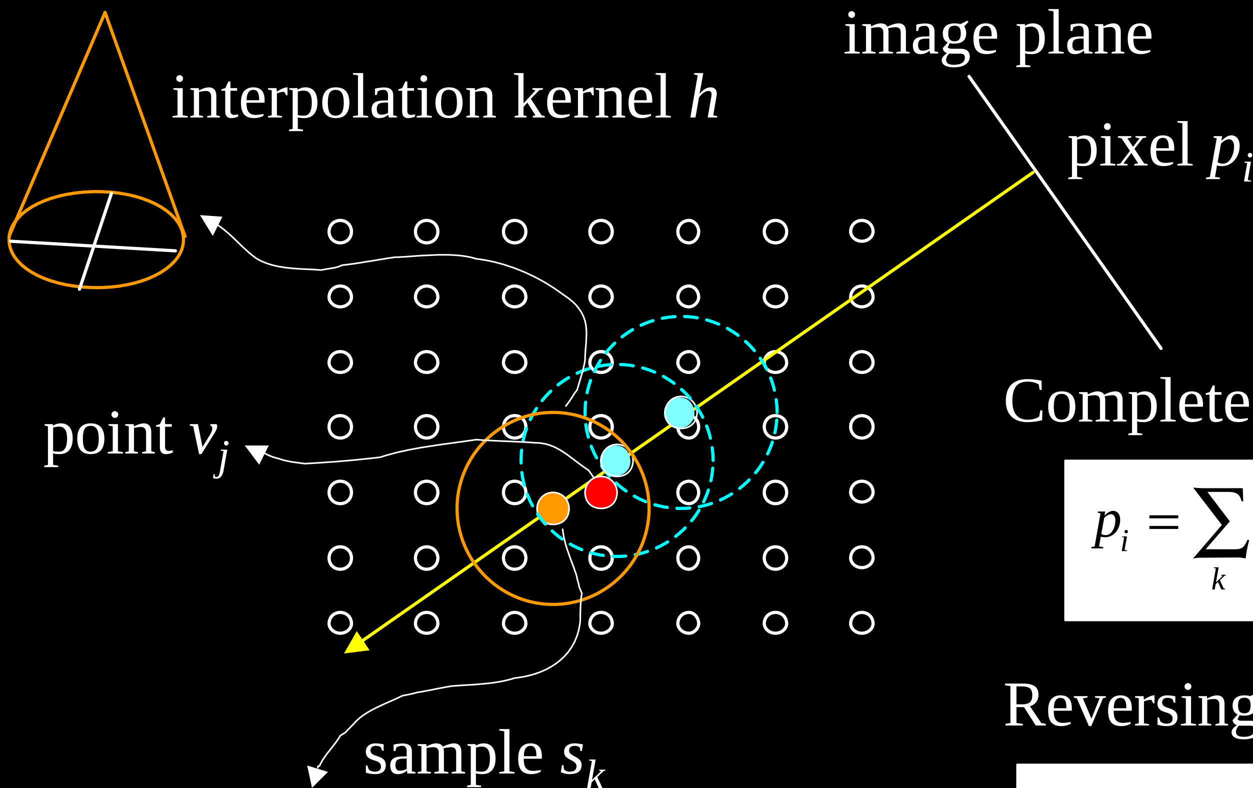
curvilinear



unstructured

X-Ray Rendering

- Estimate ray integral via discrete raycasting:



$$s_k = \sum_j v_j \cdot h(\boldsymbol{X}(s_k) - \boldsymbol{X}(v_j))$$

Complete discrete ray integral:

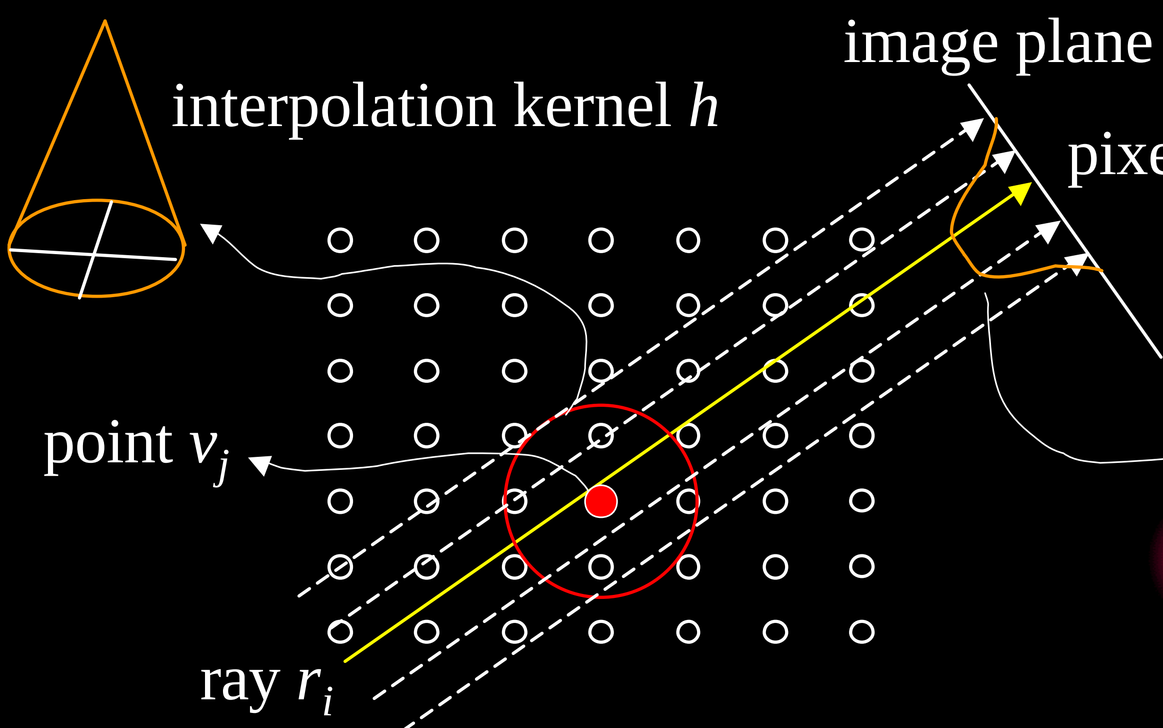
$$p_i = \sum_k \sum_j v_j \cdot h(\boldsymbol{X}(s_k) - \boldsymbol{X}(v_j))$$

Reversing the order of j and k:

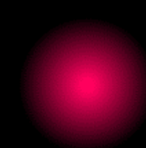
$$p_i = \sum_j v_j \sum_k h(\boldsymbol{X}(s_k) - \boldsymbol{X}(v_j))$$

X-Ray Rendering

- Estimate the ray integral via point projection:



Footprint (splat) of point $v_{j:}$



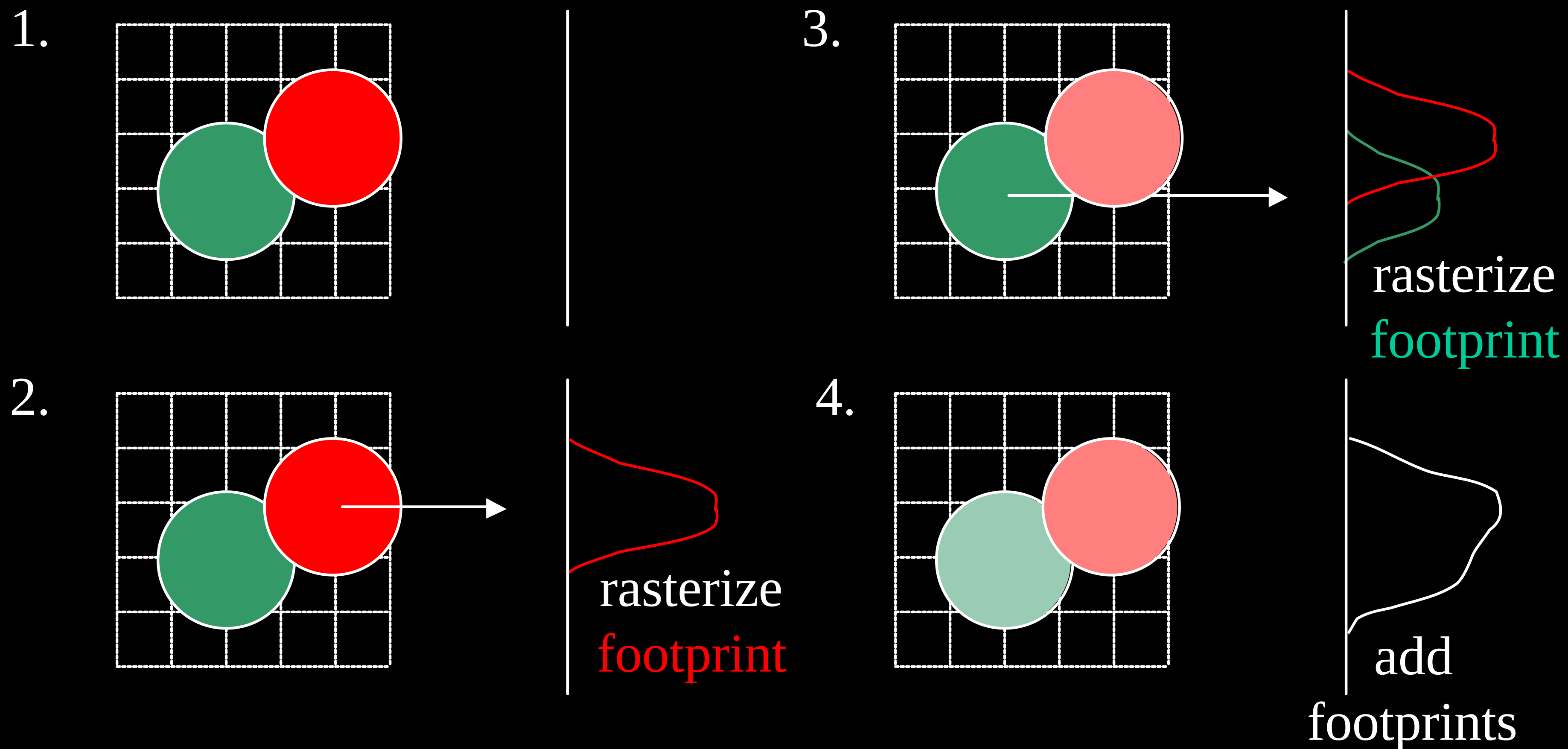
$$\bar{h}(r_i) = \int_{-ext}^{ext} h(r_i, s)\,ds$$

Compute continuous ray integral at p_i:

$$p_i = \sum_j v_j \cdot \bar{h}_j(r_i)$$

X-Ray Point Splatting

- Example: projecting a volume of two points



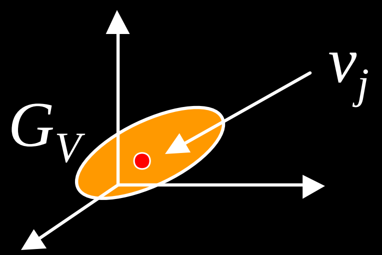
X-Ray Point Splatting

- Re-ordering was first recognized by Hanson and Wecksung for 2D CT (Hanson '85)
  - Later independently discovered by Westover for 3D volume rendering (Westover '89)
- Facilitates computation of the true ray integral
  - not just a discrete Riemann sum (raycasting)
- Pre-integrated footprint is stored into a table
  - Need a kernel function for which mappings into the footprint table can be defined for any orientation
  - The Gaussian is such a function

Point Projection

- Each point is represented by a 3D Gaussian G_V:

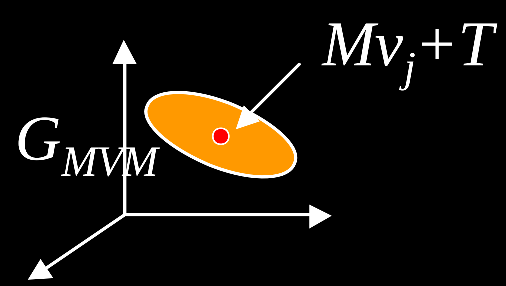
$$G_V = \frac{1}{2p\left|V\right|^{0.5}} e^{-0.5(x-v_j)^T V^{-1}(x-v_j)}$$



  - G_V is an ellipsoid to facilitate more general grids
  - It is a sphere for cubic grids

- A viewing matrix M transforms G_V into G_{MVM}:

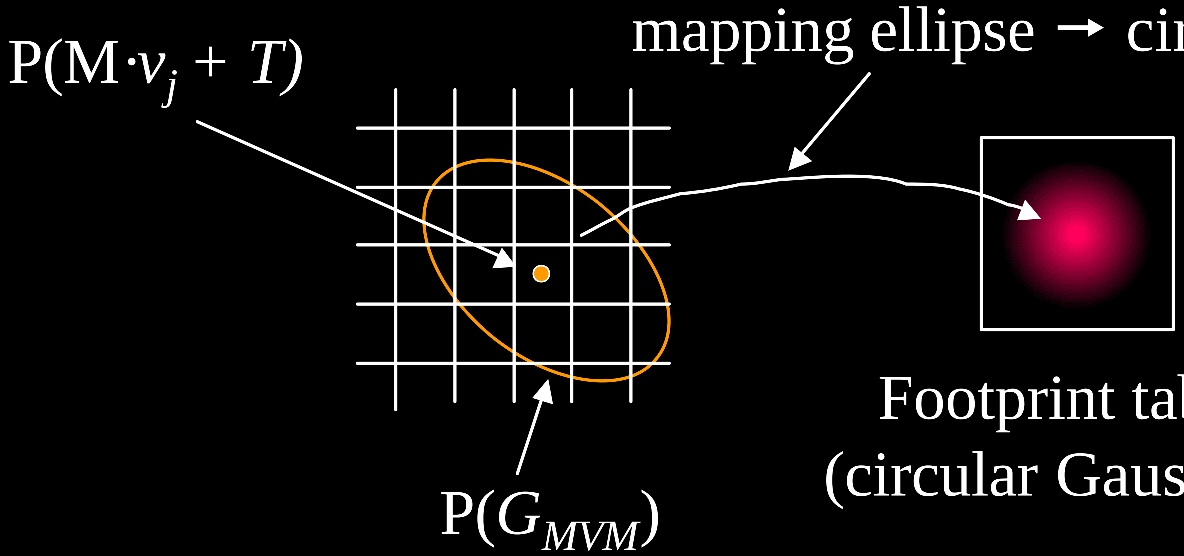
$$G_{MVM} = \frac{1}{\left|M^{-1}\right|} G_{MVM^T}(u - Mv_j - T)$$



(Heckbert ‘89, Zwicker ‘01)

Point Projection

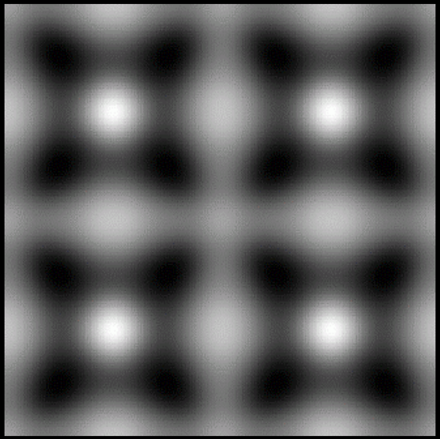
- Projection P of G_{MVM} is screen ellipse P(G_{MVM})
  - Find v_j's screen projection P(M·v_j + T)
  - Find linear mapping of P(G_{MVM}) into footprint table
  - Rasterize footprint table under P(G_{MVM}) at P(V·v_j)



Footprint table
(circular Gaussian)

Blending

- Note: Gaussian kernels do not blend perfectly
  - A small ripple always remains:



Typical range: (0.99845, 1.00249)
(assuming a function of unity)

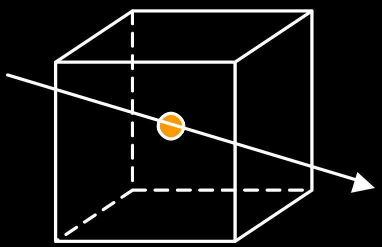
- The wider the Gaussians, the smaller the ripple
- In practice, a radius = 2.0 in volume space works well (given the appropriate Gaussian)
- See (Crawfis and Max, Vis '93) for an optimized kernel

Complexity

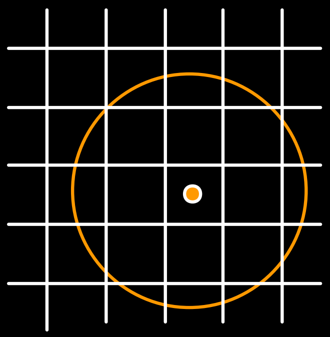
- Splatting seemingly reduces the interpolation complexity by one dimension:
  - Raycasting: interpolation of samples in 3D
  - Splatting: rasterization of footprints in 2D
- But…

Complexity

- Consider magnification = 1

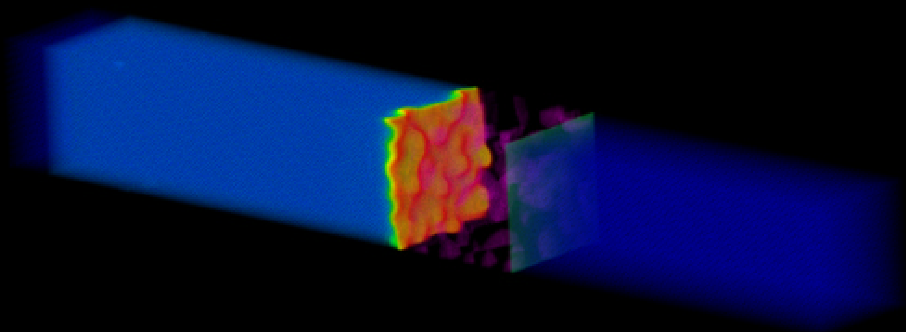


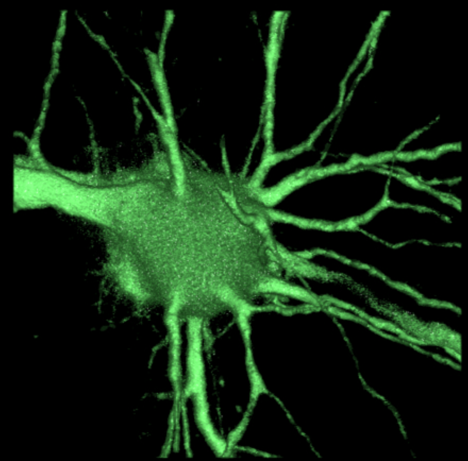
- Raycasting:
  - Commonly uses trilinear interpolation
  - Requires 8 points to calculate one ray sample point
  - Total complexity: $O(8 \cdot n^3)$
- Splatting:
  - Uses Gaussian kernel of radius=2
  - Footprint rasterization touches 16 pixels
  - Total complexity: $O(16 \cdot n^3)$



Complexity

- Does this mean that raycasting is more efficient than splatting?



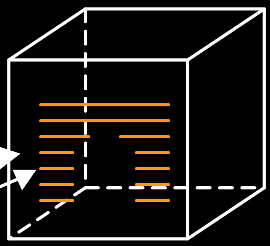


- It depends….
  - Spatially intricate objects are good candidates for point-based rendering (splatting)
  - But the simplicity of splatting has advantages even for less favorable objects

Storage Complexity

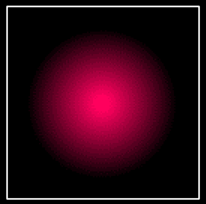
- Generally, only need to store relevant points
  - Non-air points, masked-out points, ROI-points
- Provides easy space-leaping for irregular objects
- Storage schemes (in increasing order of spatial coherence):
  - List of points, sorted by value (fast iso-contouring)
  - RLE list of points (fast transformations and sparse)
  - Octree with hierarchical bins of points

RLE: #E #F$v_1v_2v_3v_4$ #E #Fv_1v_2…
#E #F$v_1v_2v_3$ #E #F$v_1v_2v_3$…



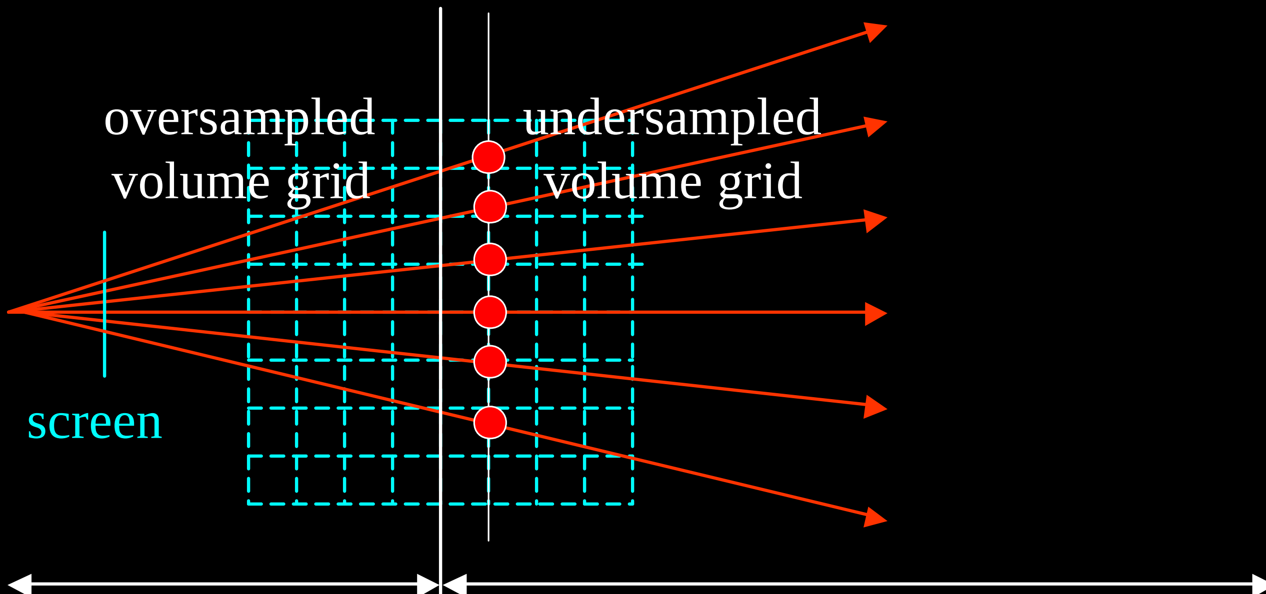
Rendering

- RLE list facilitates fast incremental arithmetic for point projection in software
- Texture mapping hardware can also be used
  - Texture map footprint onto a square polygon
  - Set GL blending functions, etc.
  - Warp polygon according to point’s screen space ellipse
  - Align the warped polygon with the screen
  - Project polygon to the screen



Aliasing

- In perspective or at low magnifications, some volume portions may be sampled below Nyquist

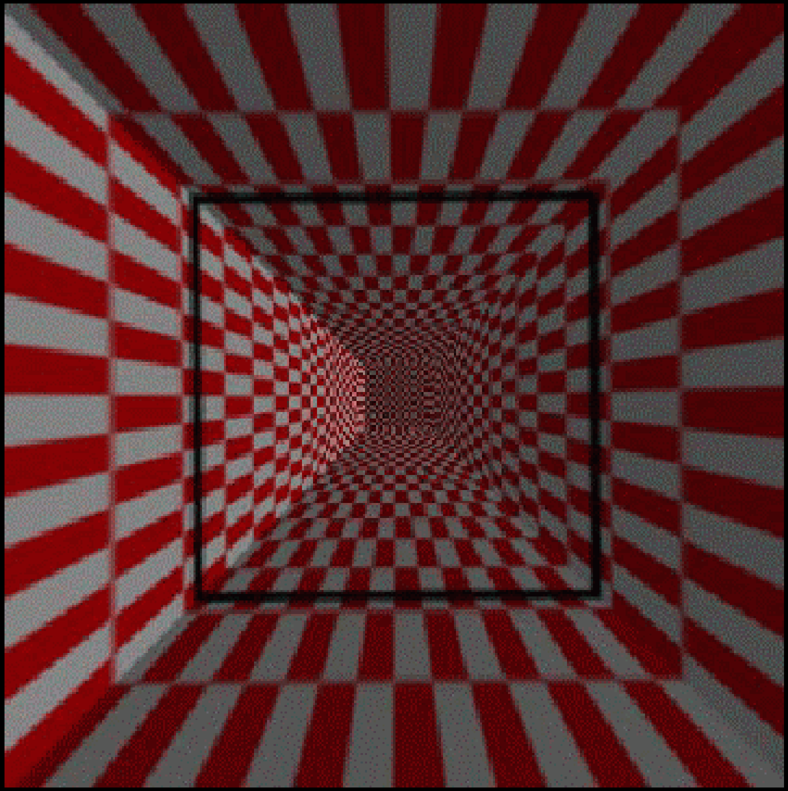


Ray grid sampling rate
> volume grid sampling rate
→ no aliasing

Ray grid sampling rate
£ volume grid sampling rate
→ aliasing

Aliasing

- Effects of aliasing

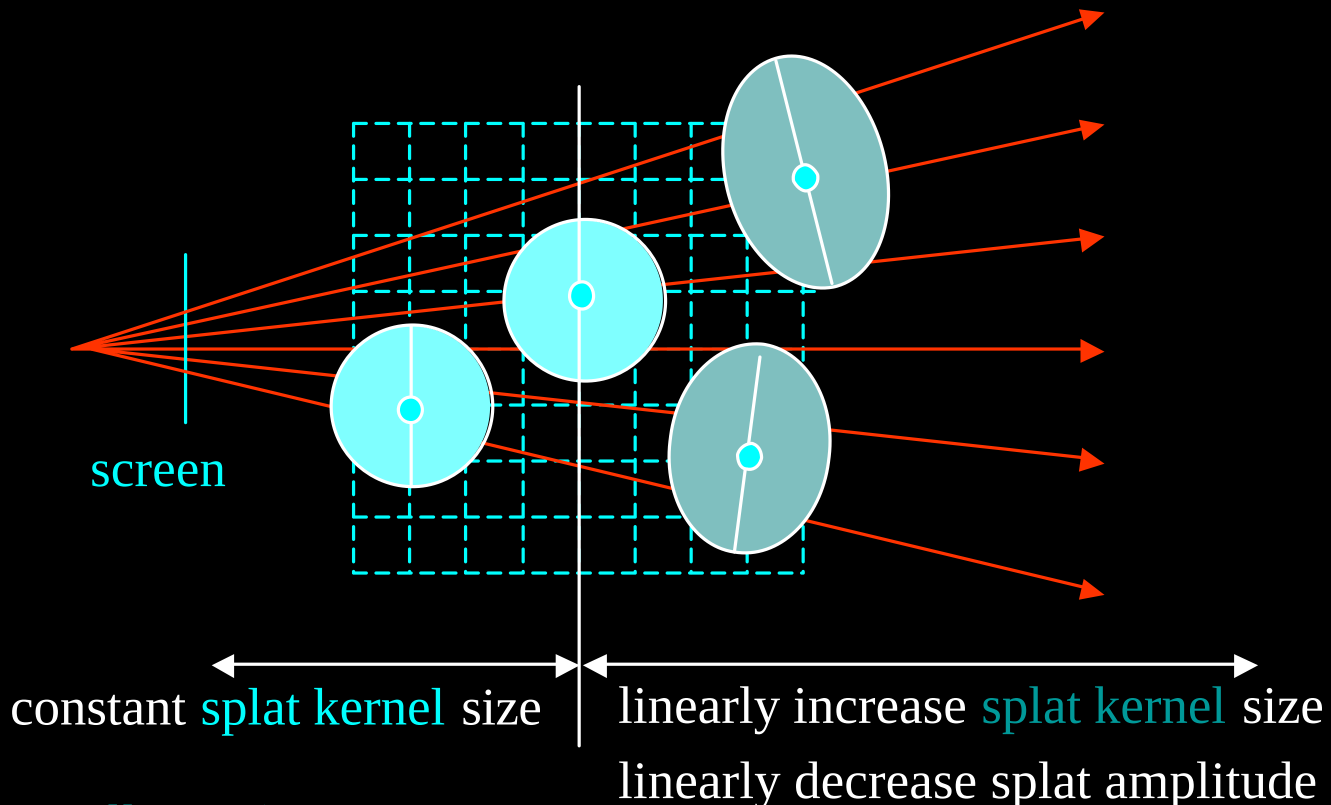


checkerboard tunnel

terrain

Anti-Aliasing

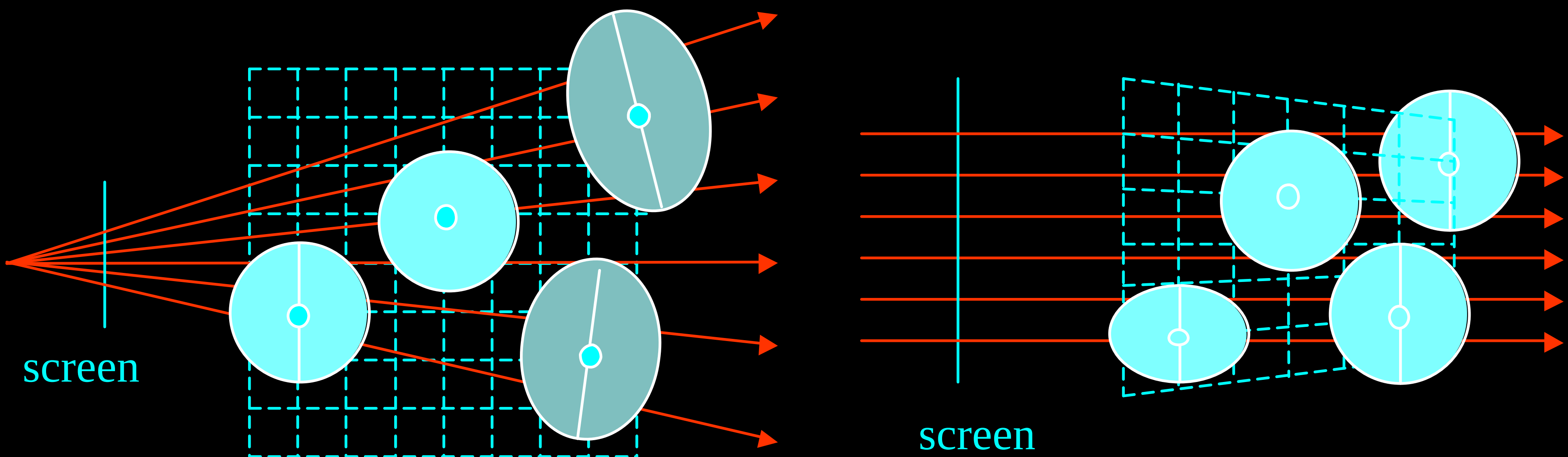
- Adapt kernel bandwidth for proper anti-aliasing
- Amounts to a stretch of the 3D kernel



(Swan '97, Mueller '98)

Anti-Aliasing

- Conveniently done in perspective (ray-) space



camera space

ray space

Anti-Aliasing

- Compute the Gaussian ellipsoid in ray space
  - Calculate the Jacobian J of the local perspective distortion (varies for each point)
  - Compute the ray space ellipsoid G_{JMV} using J

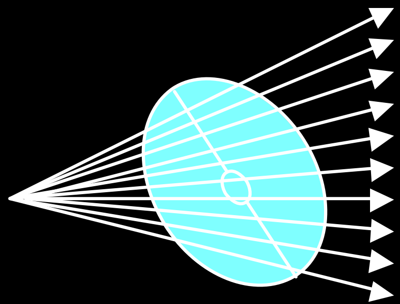
camera space

$$G_{MV} = \frac{1}{\left|M^{-1}\right|} G_{MVM^T}(u - Mv_j - T)$$

ray space

$$G_{JMV} = \frac{1}{\left|J^{-1}\right|} G_{JMVJ^T}(x - x_k)$$

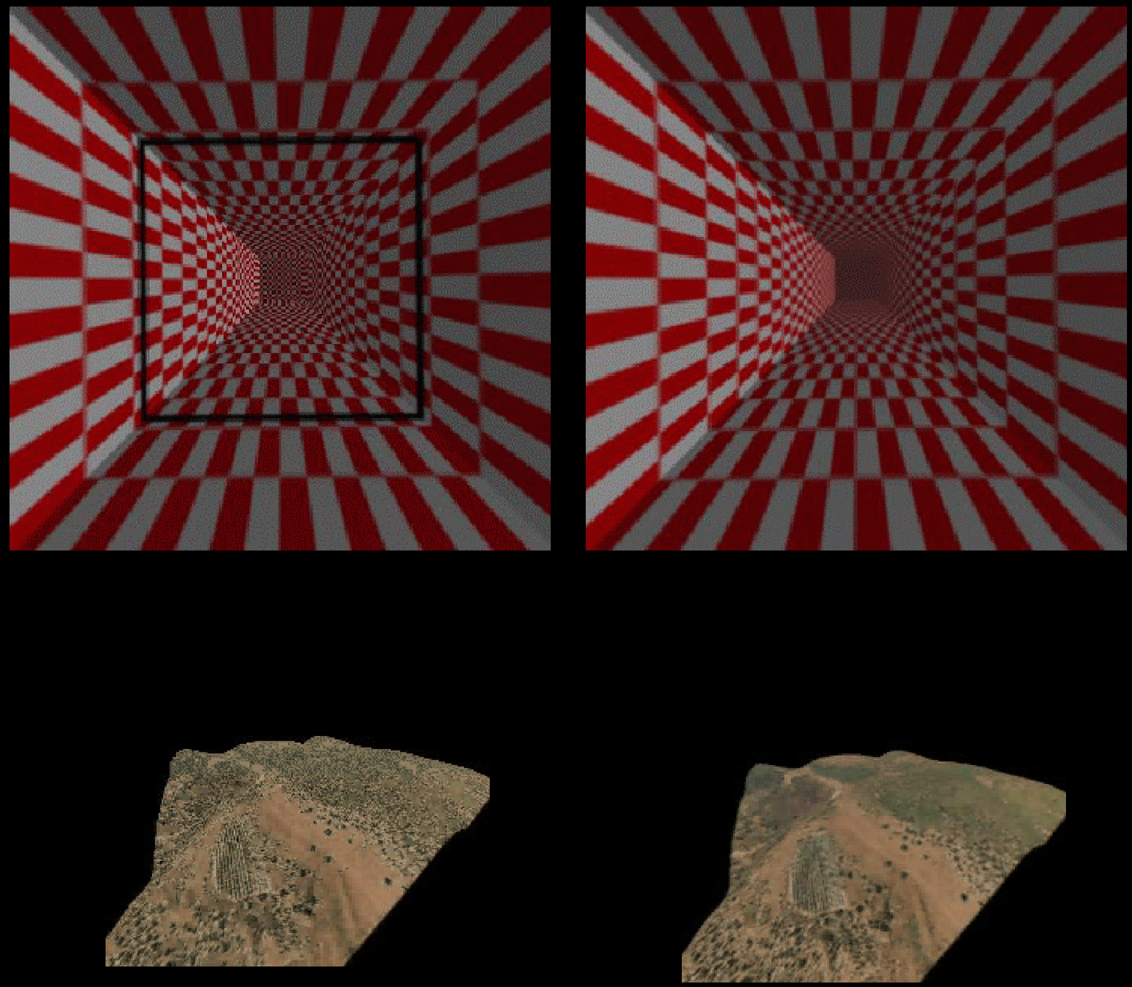
center of Gaussian in ray space



generalized Gaussian ellipsoid in camera space

(Zwicker ‘01)

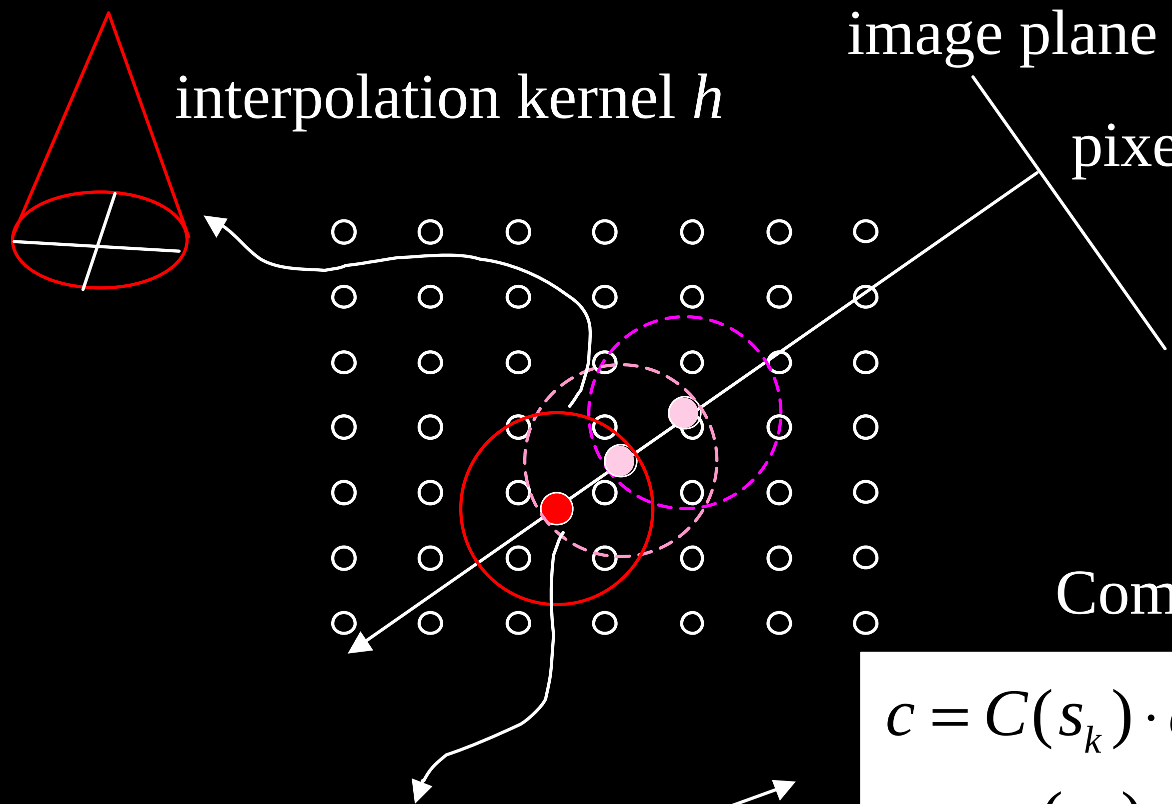
Anti-Aliasing - Results



aliased anti-aliased

Compositing - Raycasting

- Reconstruction followed by compositing



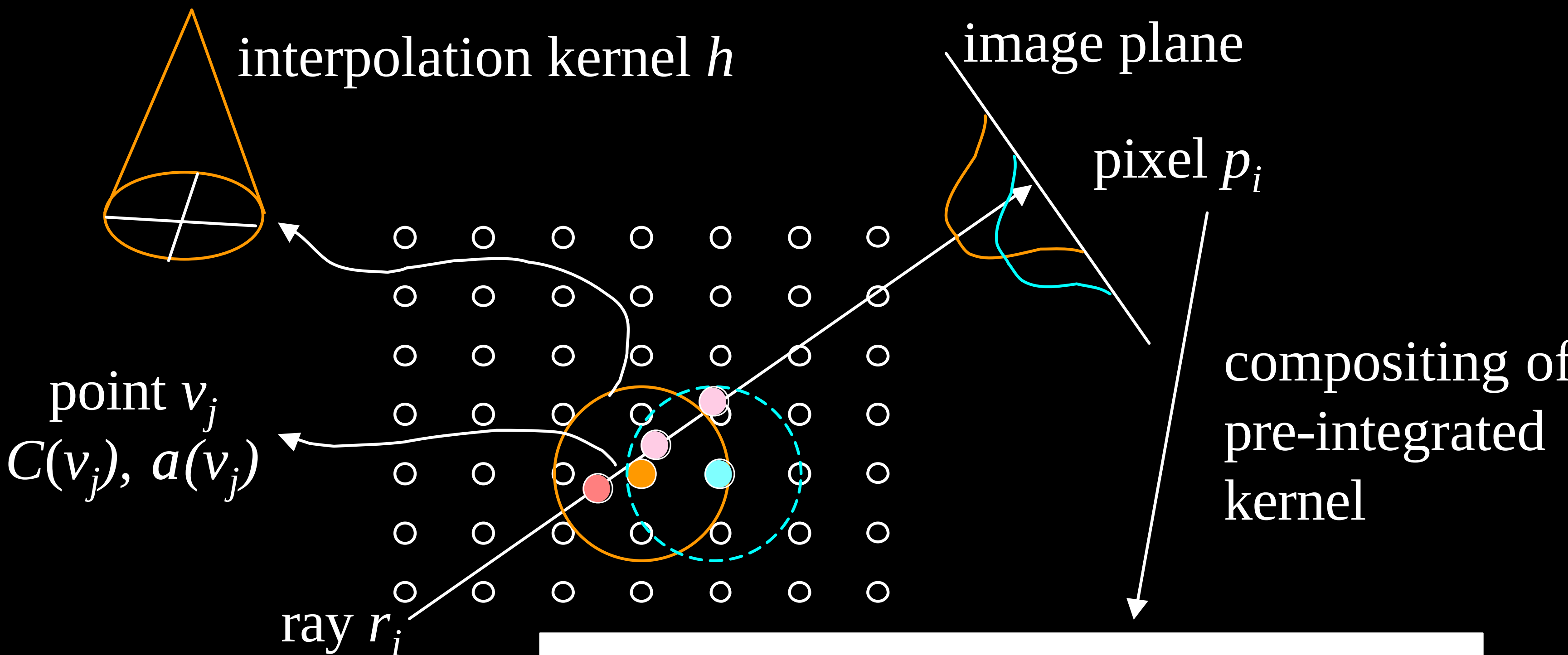
Compositing:

$$c = C(s_k) \cdot a(s_k) \cdot (1 - a) + c$$

$$a = a(s_k) \cdot (1 - a) + a$$

Compositing - Splatting

- Reconstruction not separable from compositing



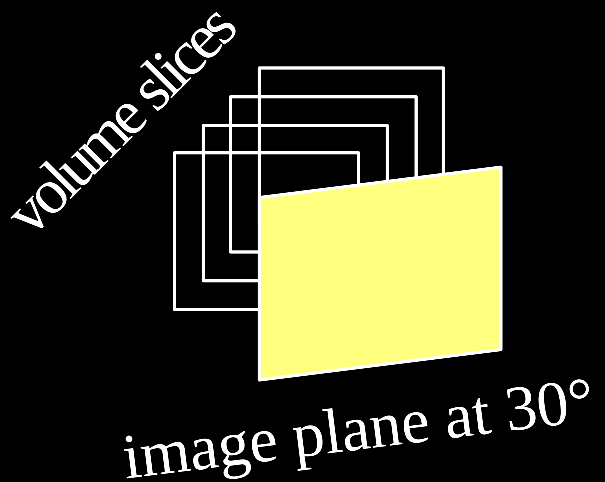
$$\bar{h}(r_i) = \int_{-ext}^{ext} h(r_i, s)\,ds$$

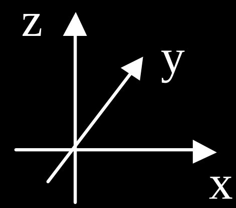
$$c = C(v_j)\cdot\bar{h}(r_i)\cdot a(v_j)\cdot\bar{h}(r_i)\cdot(1-a)+c$$
$$a = a(v_j)\cdot\bar{h}(r_i)\cdot(1-a)+a$$

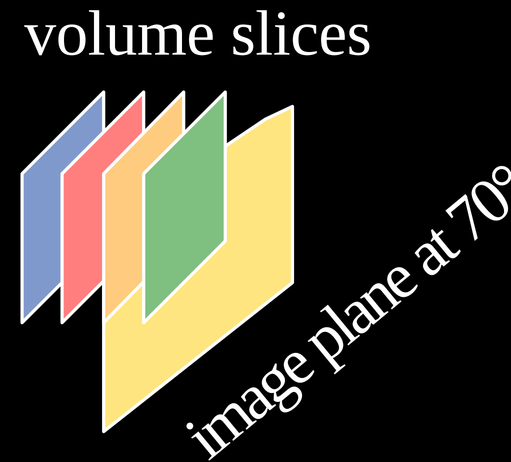
Compositing

- Two strategies devised by Westover (Westover ‘89, ‘90)
- Composite every point:
  - Shown in previous slide
  - Fast and simple
  - Leads to “sparkling” in animated viewing
- Axis-aligned sheet-buffers:
  - Add splats within sheets most parallel to image plane
  - Composite these sheets in depth-order
  - Leads to “popping” artifacts in animated viewing

Axis-Aligned Sheet-Buffers

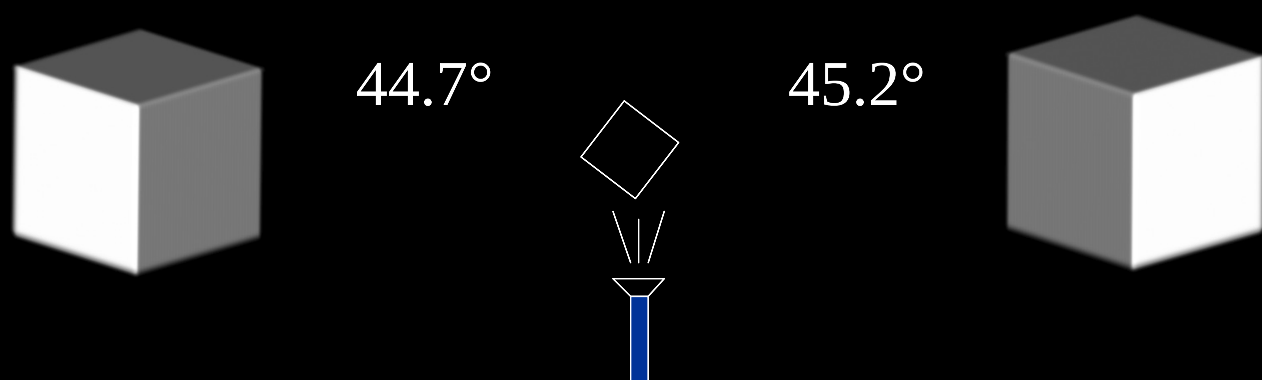






Switch compositing
axis at 45°

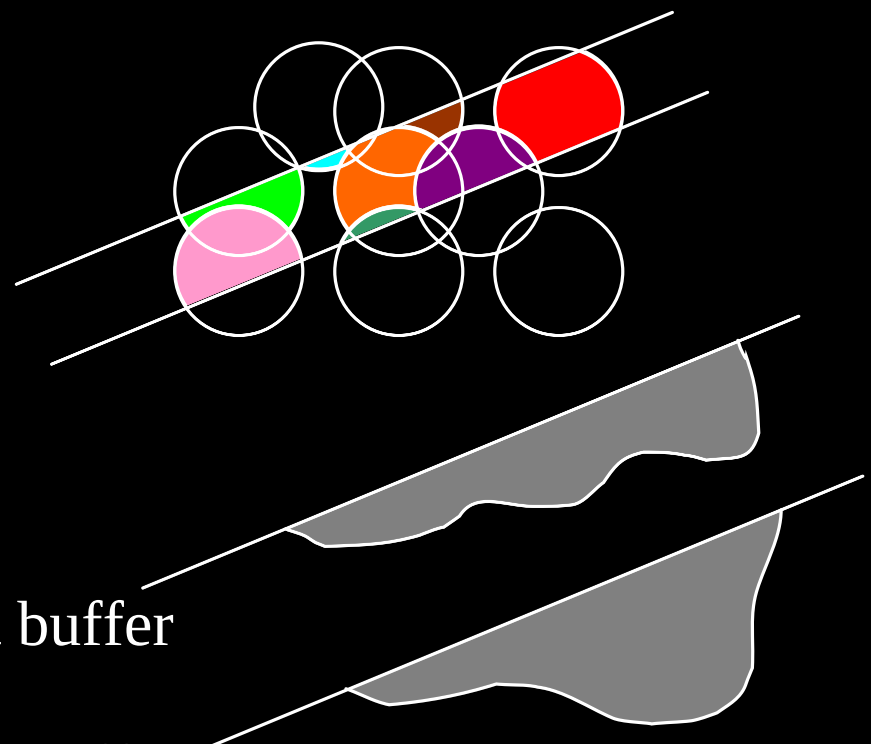
Popping occurs:



binary cube

Image-Aligned Sheet-Buffers

- Eliminates popping
  - Slicing slab cuts kernels into sections
  - Kernel sections are added into sheet-buffer
  - Sheet-buffers are composited

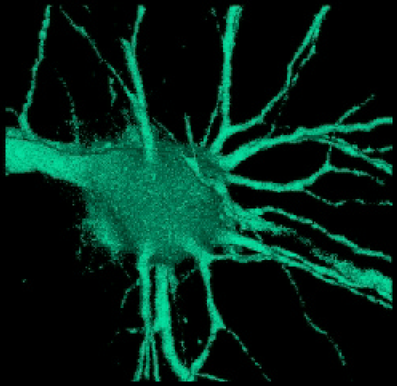


binary cube

(Mueller '98)

Image-Aligned Sheet-Buffers

- Footprint mapping as usual
  - Requires multiple footprint rasterizations per point



axis-aligned

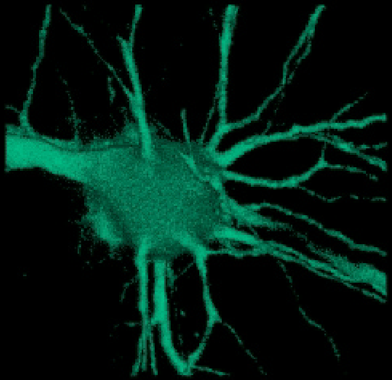
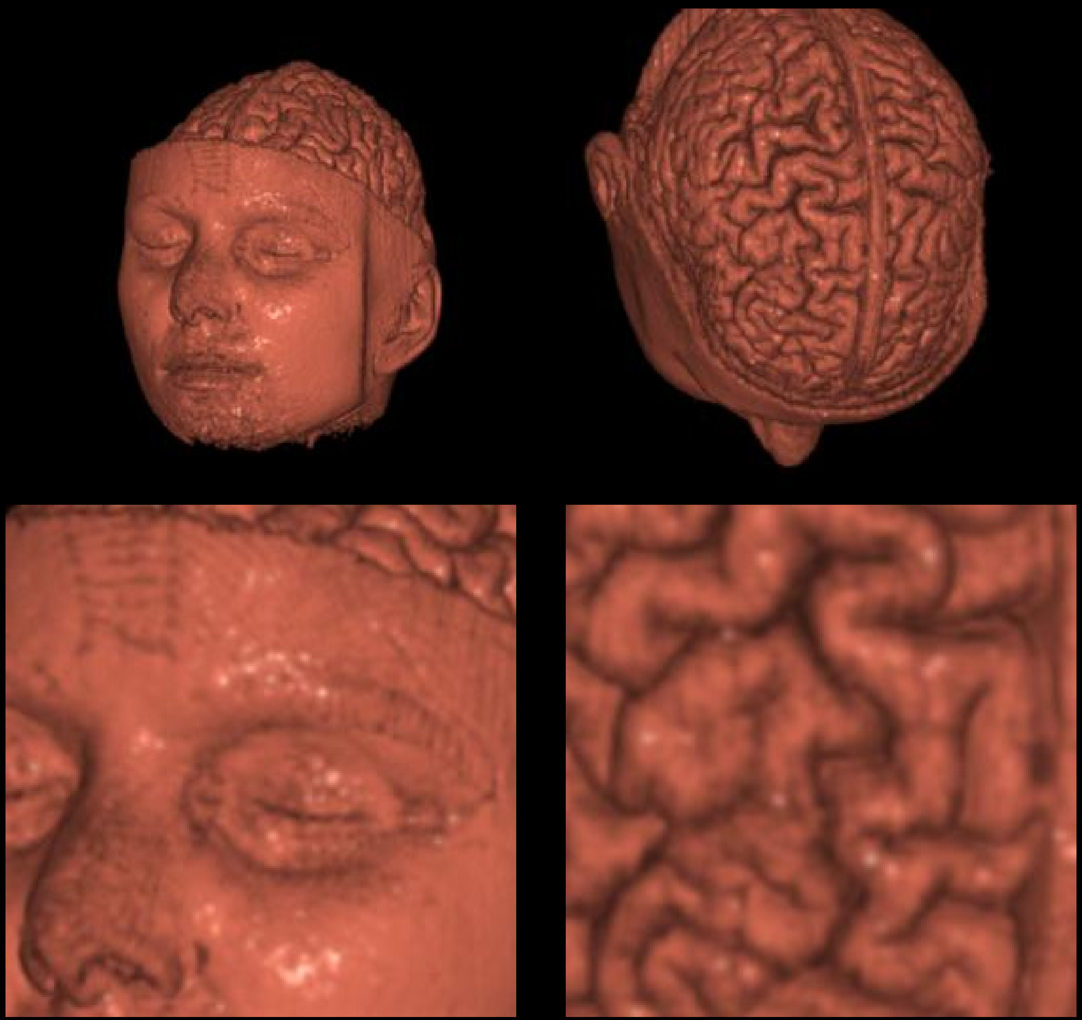


image-aligned

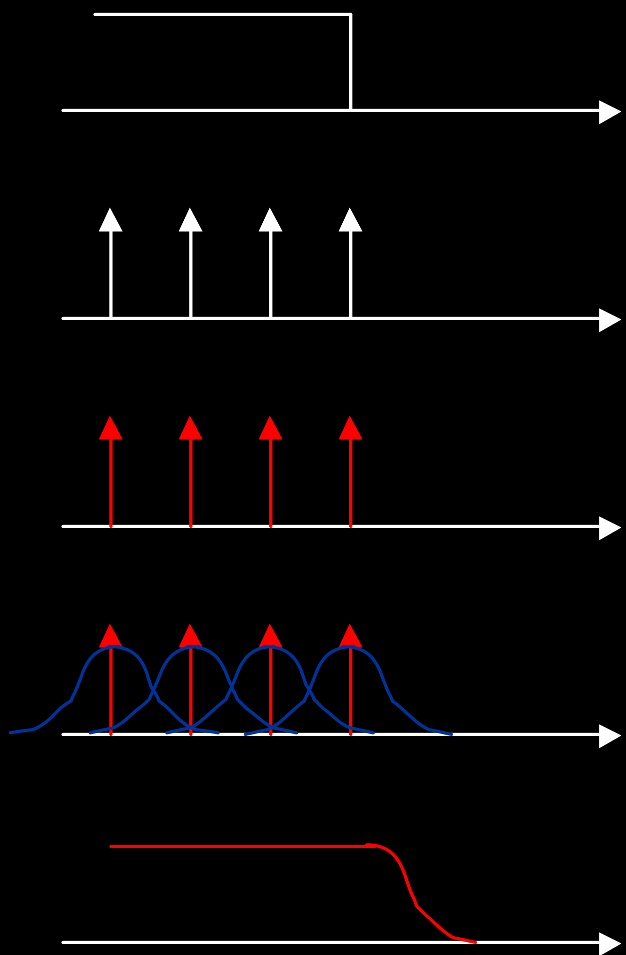
Pre-Classified Splatting



normal

blurred close-ups

Pre-Classified Splatting



Original edge

Sampled edge

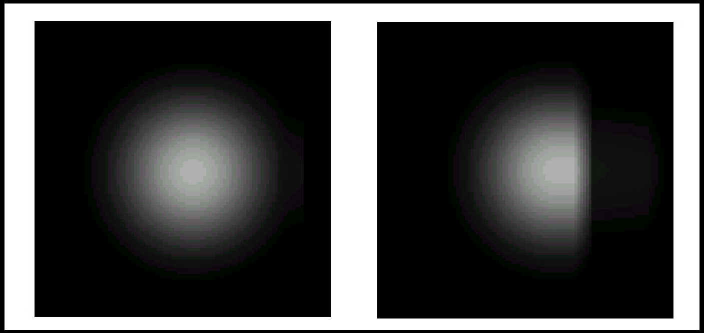
Classification and shading

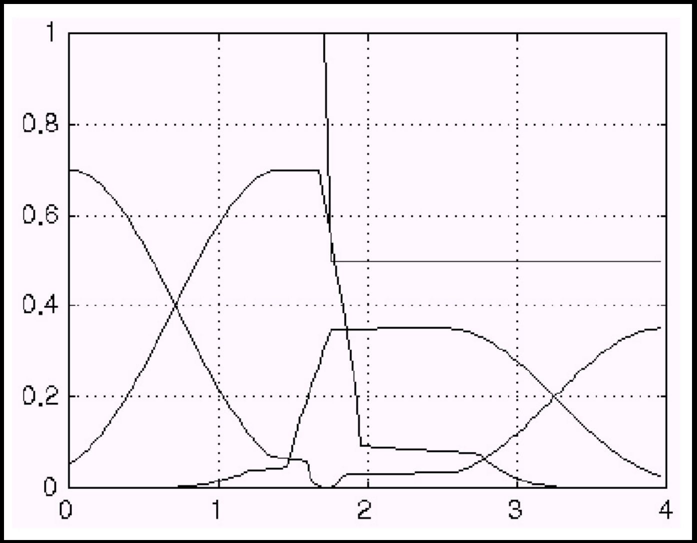
Splatted with Gaussian kernel

Reconstruction: blurred edge image

One Solution: Edge Splats

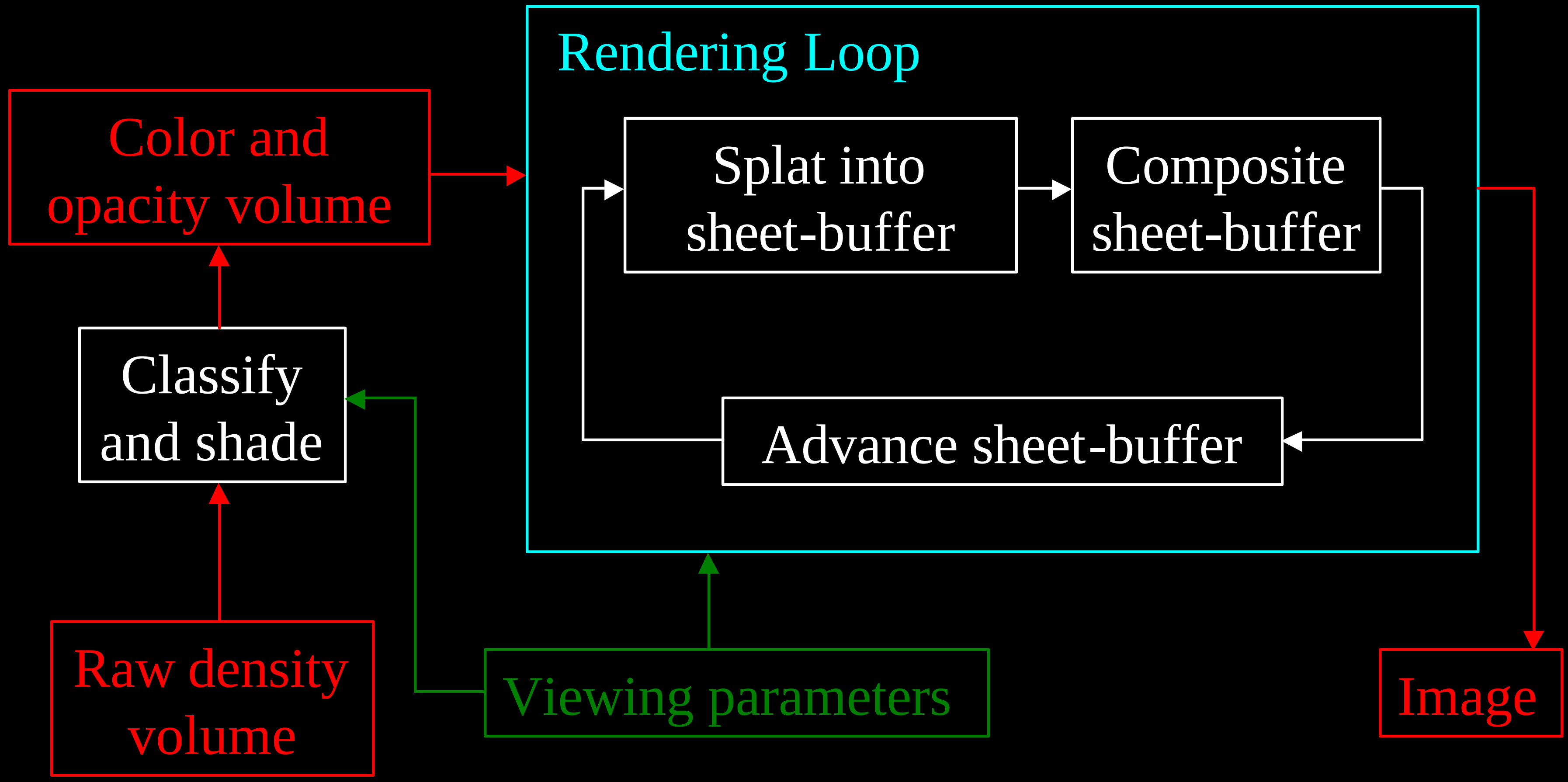
- Edge splats (Huang '98)
  - replace normal splat by special edge splat





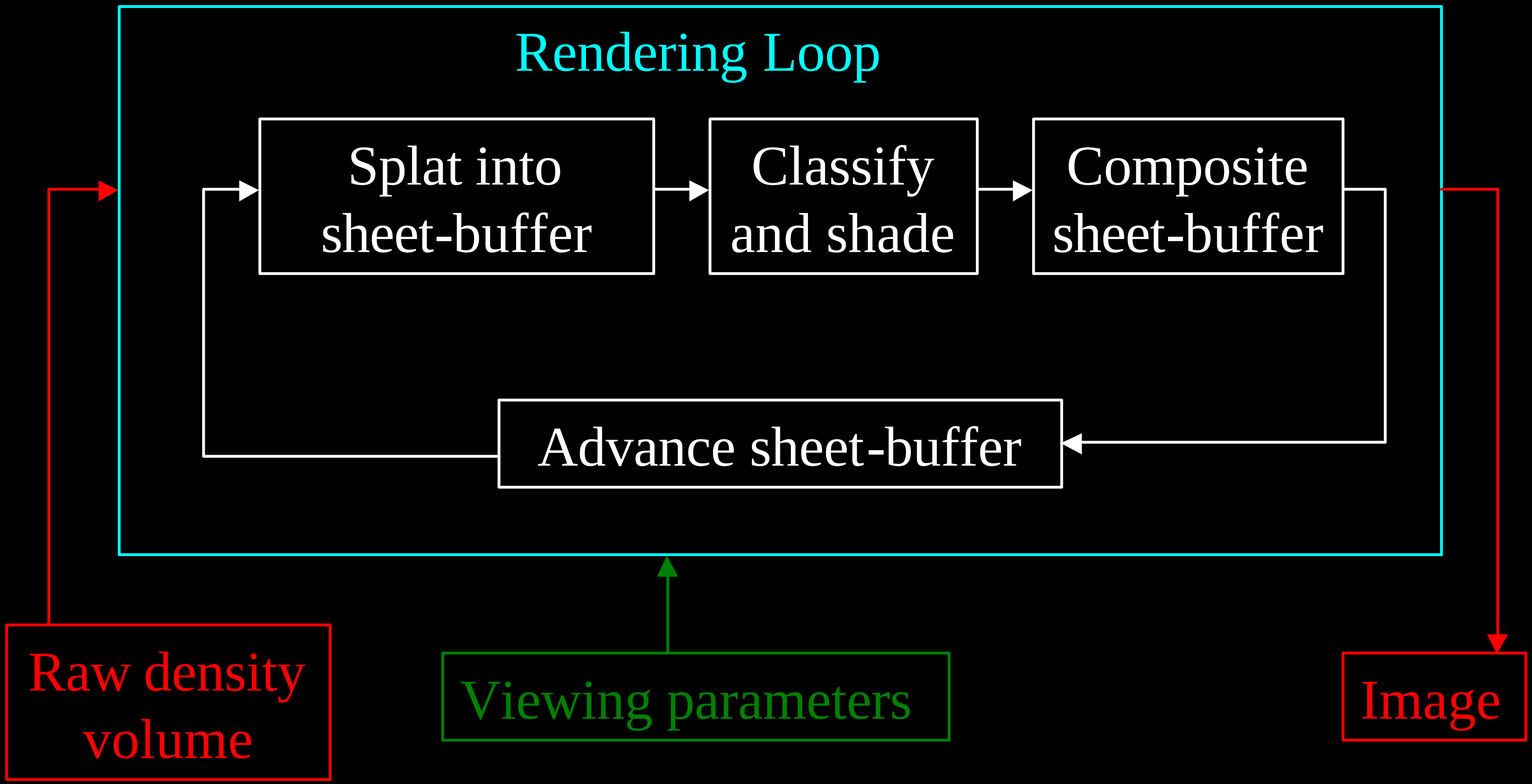
- Shortcomings:
  - pre-processing required
  - problems with discontinuities
  - "micro-edges" are hard to resolve

Pre-Classified Rendering



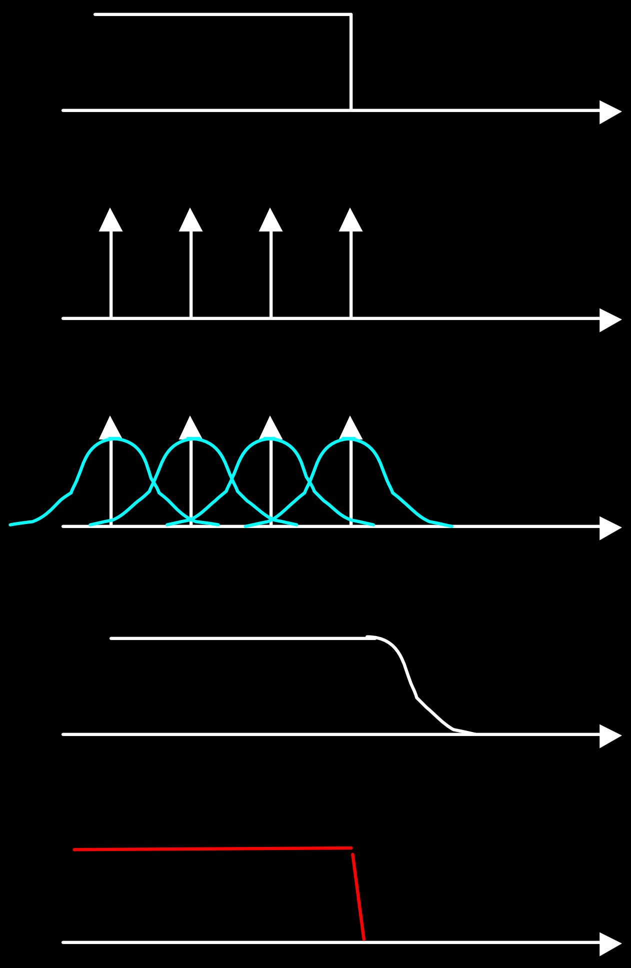
Post-Classified Rendering

(Mueller ‘99)



Note: this can only be done with image-aligned sheet buffers

Post-Classified Splatting



Original edge

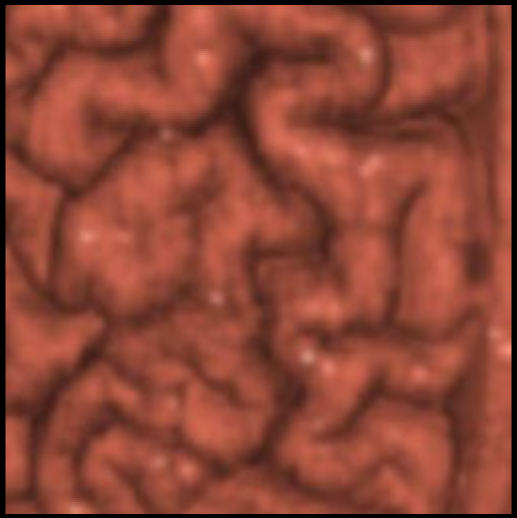
Sampled edge

Splatted with Gaussian kernel

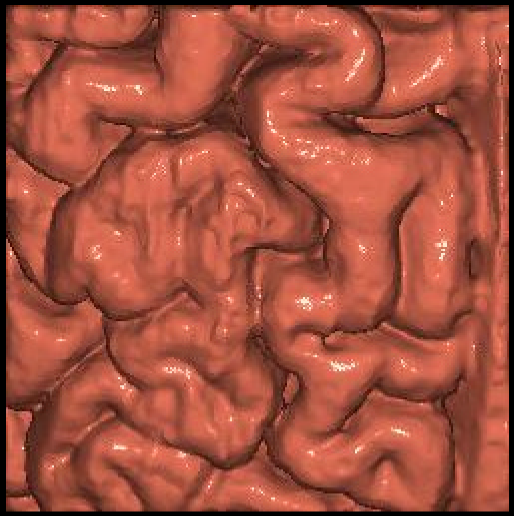
Reconstruction: blurred edge

Classification: crisp edge image

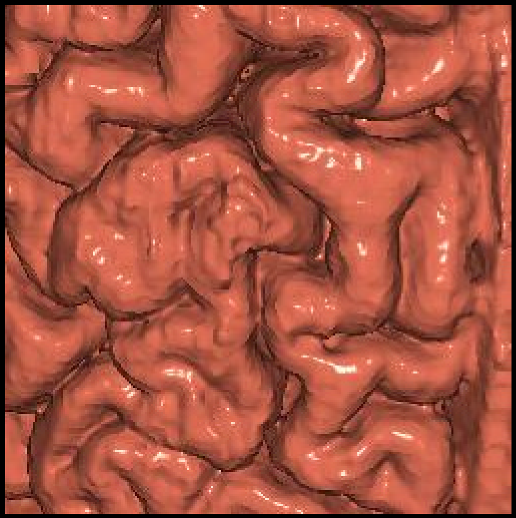
Post-Classified Splatting



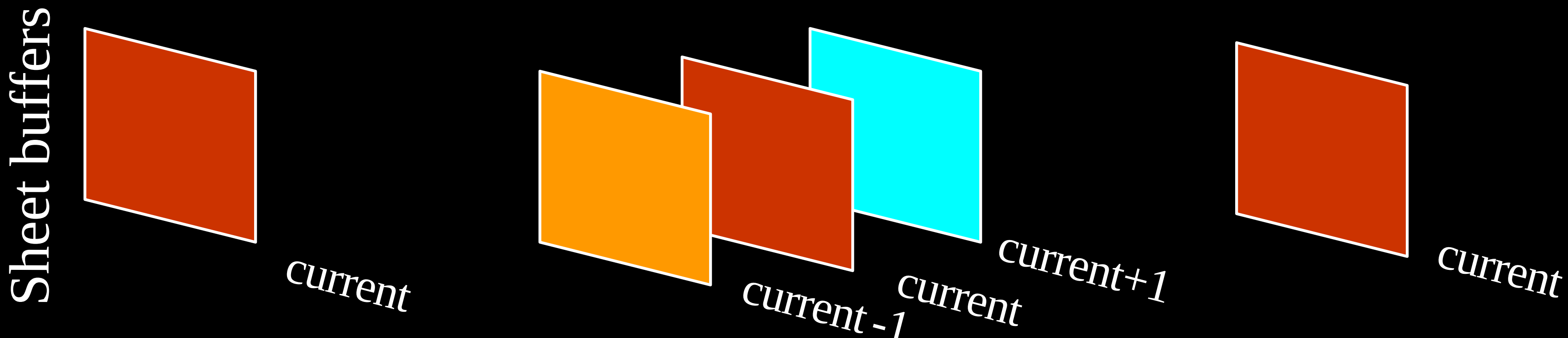
pre-shaded



post-shaded,
central difference

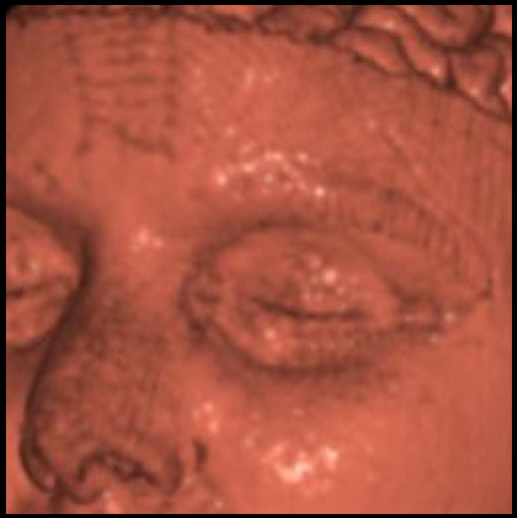


post-shaded,
gradient splats

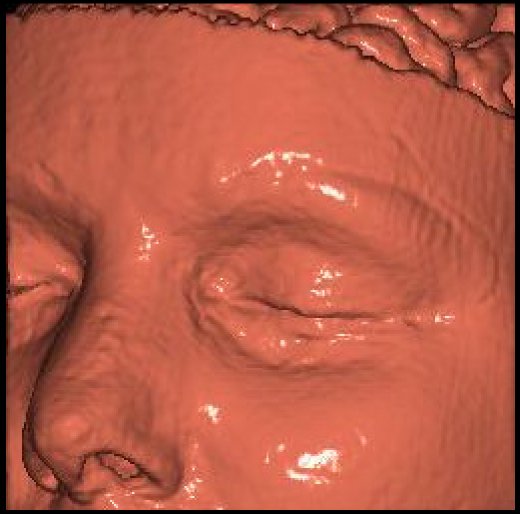


Post-Classified Splatting

pre-shaded

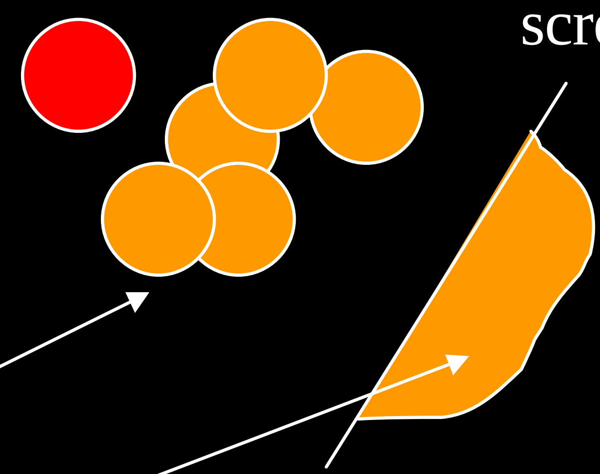


post-shaded



Occlusion Culling

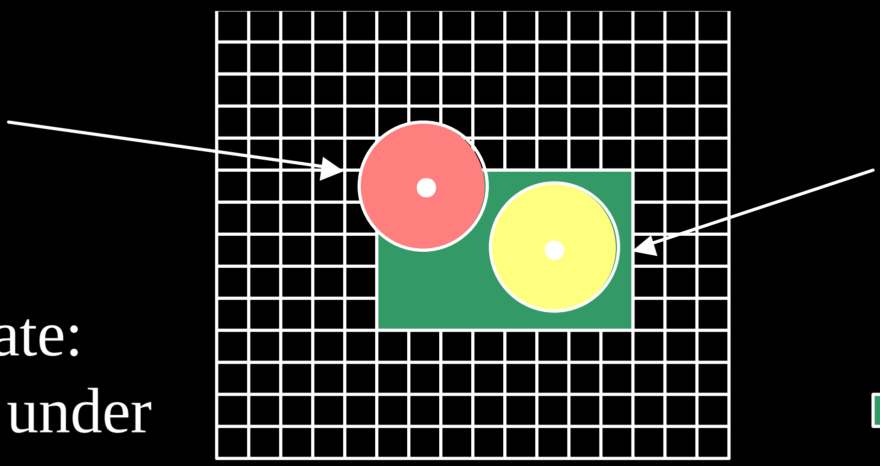
- Culling occluded points saves lots of time
- A point is only visible if the volume material in front of its footprint is not opaque



Require front-to-back rendering

Occlusion Culling

- Requirements for point visibiliy test:
  - Fast, efficient, simple
  - Hierarchical: quickly cull entire blocks of points
  - Accurate: the entire footprint must be occluded

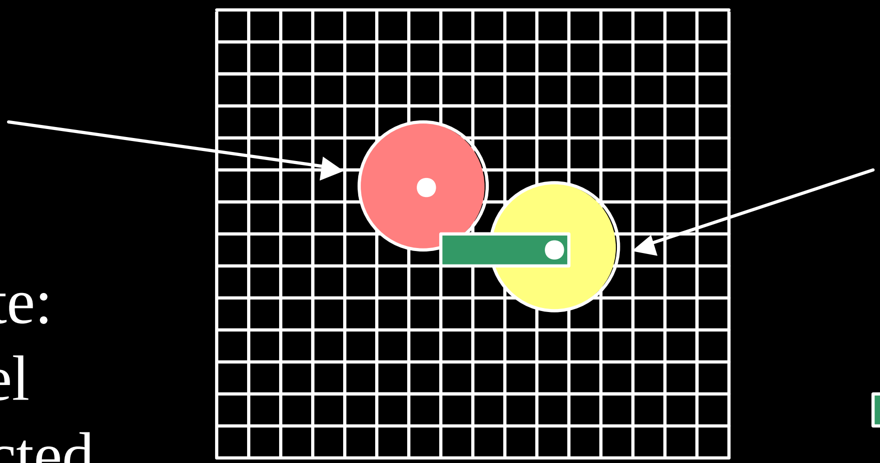


Slow, but accurate:
check all pixels under
the footprint first

Occlusion Culling

- Better method (Mueller '00):
  - After compositing, convolve opacity image with a box filter (size = projected footprint)
  - Then, when a pixel value > threshold, the entire footprint neighborhood > threshold

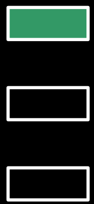
project



do not project

Fast and accurate:
Only check pixel nearest to projected point center

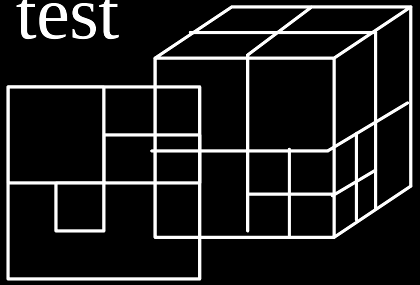
occlusion map



Occlusion Culling

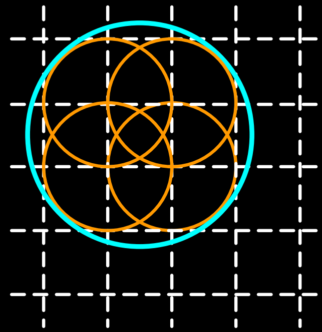
- Hierarchical occlusion maps (Lee '00):
  - Keep points in an octree
  - Maintain visibility map in form of a quadtree
  - Check projection of an octree node with corresponding level of the visibiliy map quadtree
  - Cull occluded octree nodes
  - Subdivide octree node if not occluded
  - Rasterize points that fail the occlusion test
  - Update visibility map

visibility map



Multi-Resolution Points

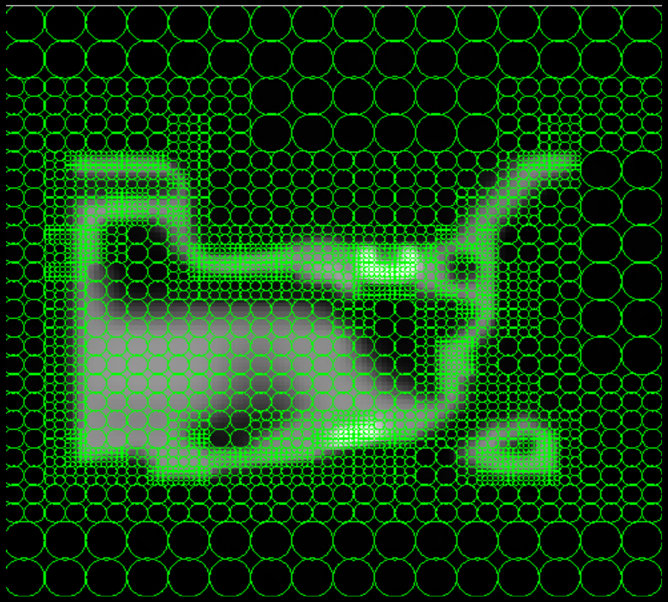
- Render one large point in place of many small points
  - Less rasterization cost (overlap areas)
  - Less storage required

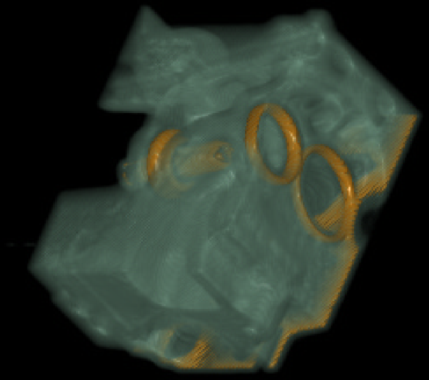


- Control point size by volume content
  - Organize points into a tree
  - Use a local error metric to decide on point size
  - Laur and Hanrahan use RMS error (Laur '91)
  - User sets an error threshold to control tree traversal

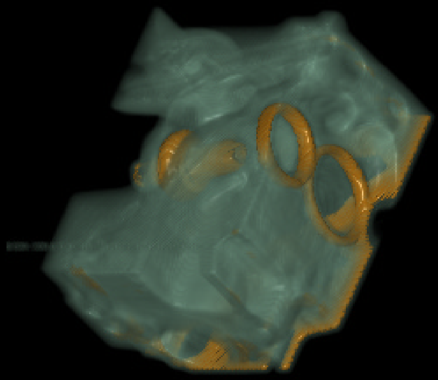
Multi-Resolution Points

- Preliminary results:
  - Use a frequency-space metric to control error and determine the size of the splatted point





Original resolution:
240k points



Multi-resolution:
90k points

(Welsh '02)

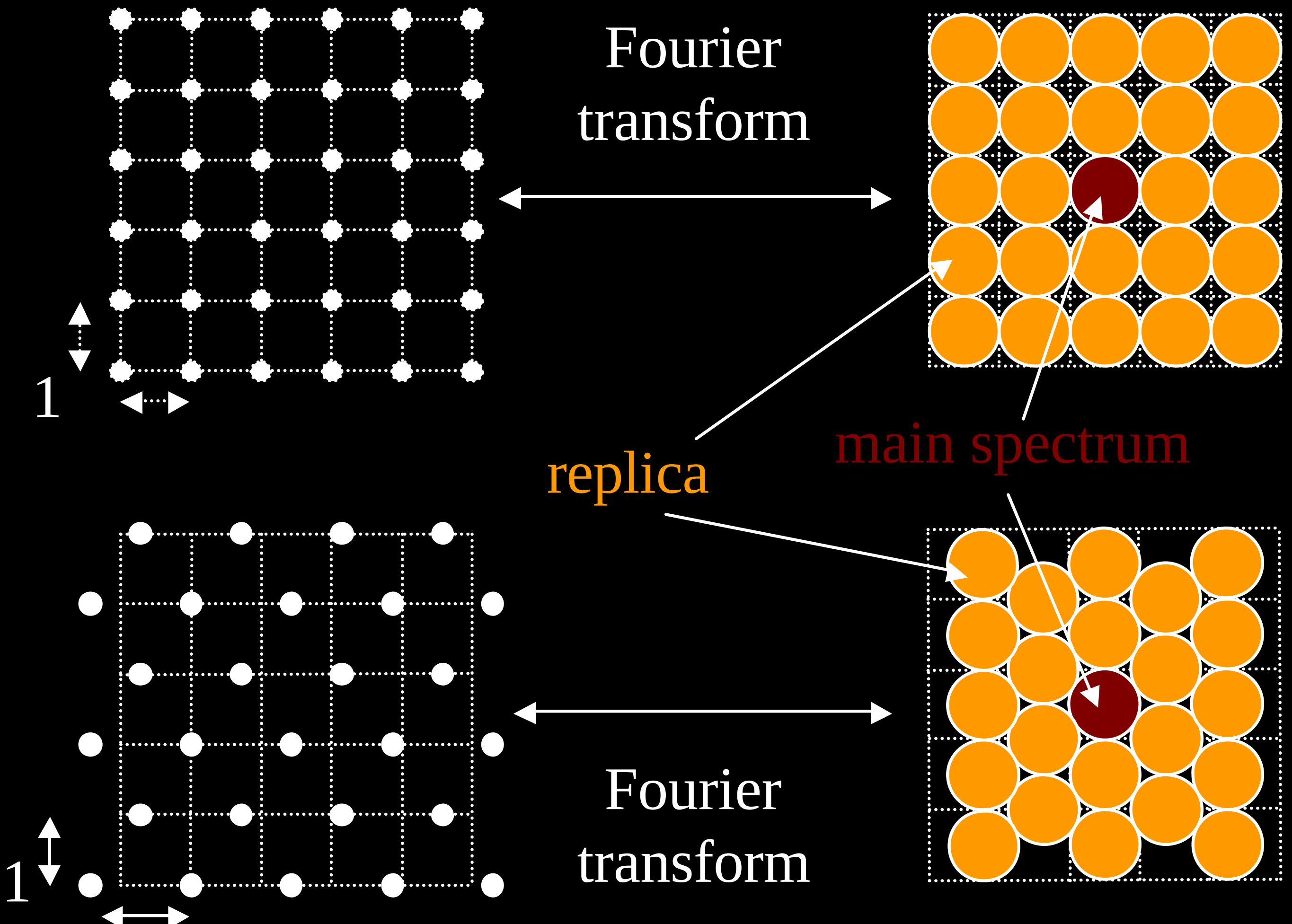
Compression

- Points provide a lossless data compression by retaining only a list of relevant points
- Are there further lossless compression opportunities?
  - Assume we deal with regular grids
  - Are there more efficient regular grids than the cubic cartesian grid?
- The answer comes from the theory on sphere packings and lattices (Conway ‘93)

Alternative Grids

cubic Cartesian (CC)

hexagonal grid (13.4% less samples)

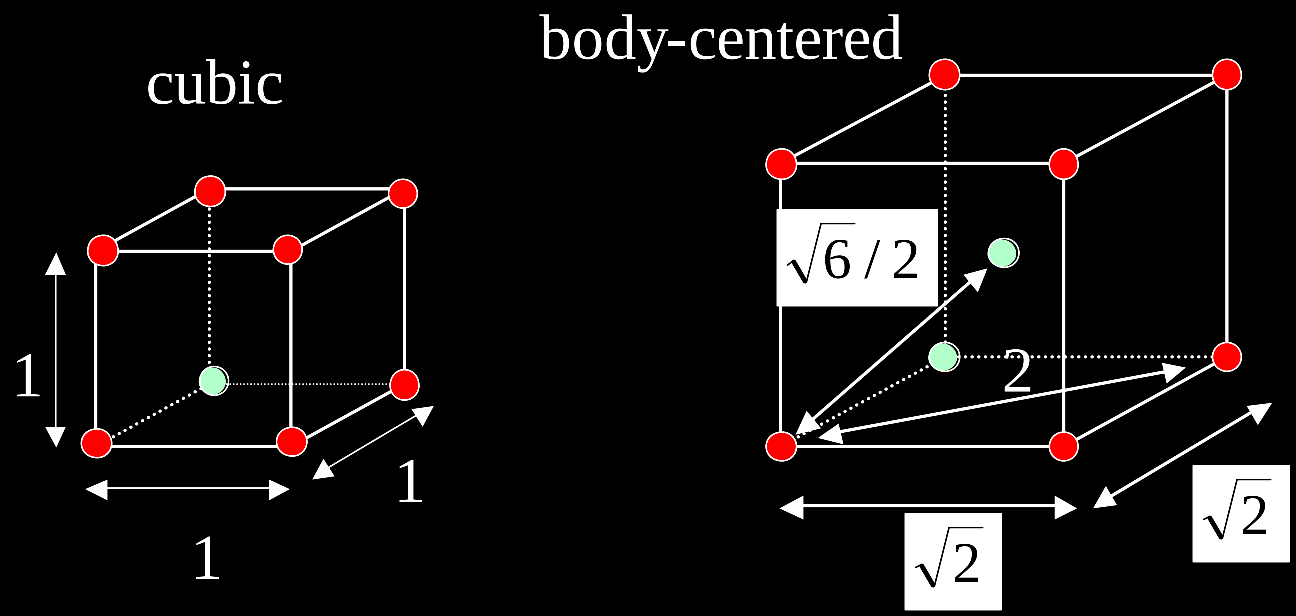


spatial domain

frequency domain

Alternative Grids

- Body-centered cartesian (BCC) grid:
  - Reduces # of required point samples to 70.3%

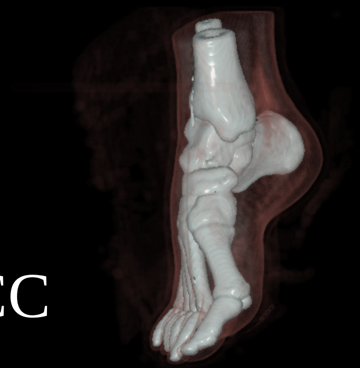


  - 4D BCC grid requires only 50% of the equivalent 4D cubic grid samples

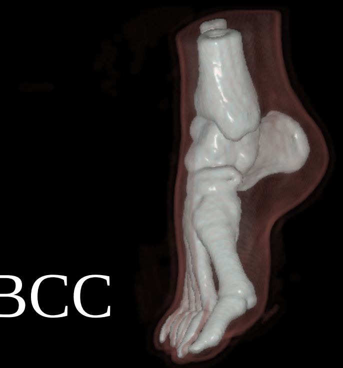
Alternative Grids

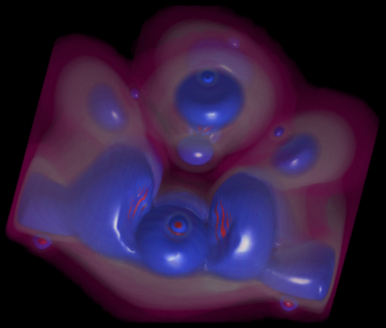
- Notes:
  - BCC grids assume spherically bandlimited signal
  - Under that assumption compression is lossless
- Rendering (Theussl '01):
  - All usual point rendering methods are applicable
  - Need to shift slices by $1/\sqrt{2}$

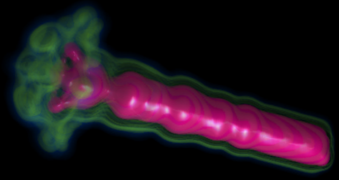
CC



BCC

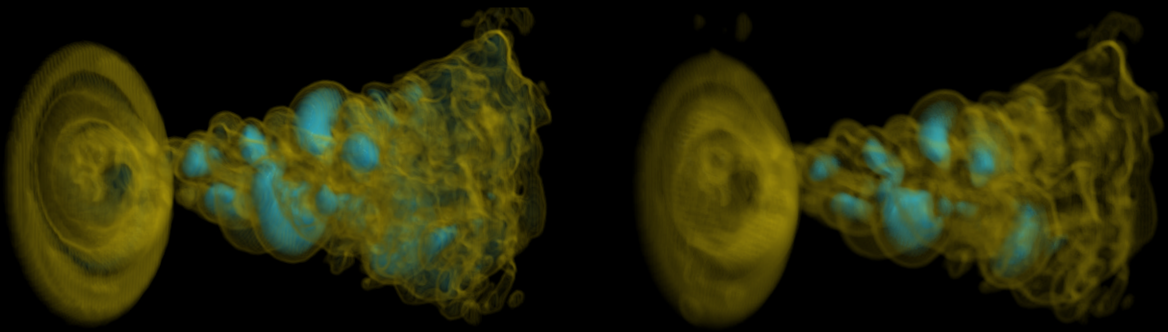






BCC

Alternative Grids

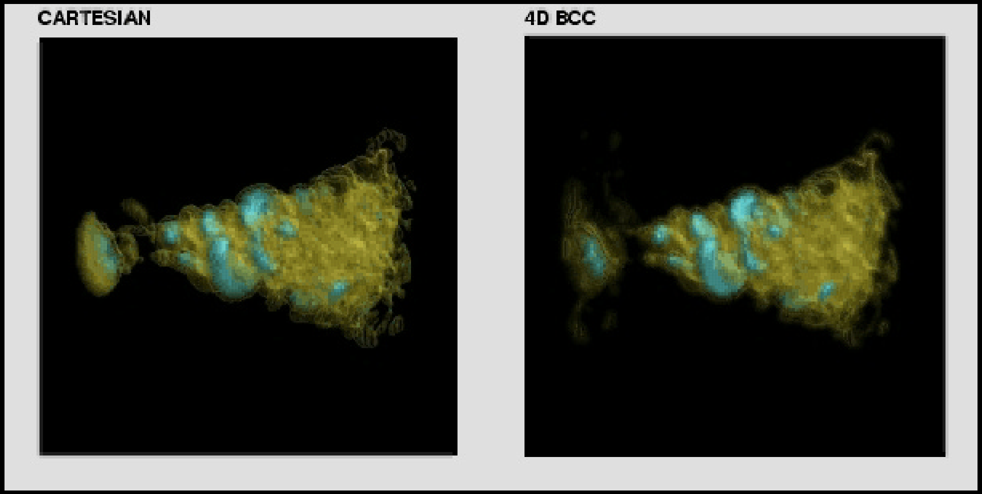


- Turbulent Jet 4D CC
  - 99 time steps (168M)
  - Relevant voxels: 9.4M
  - 3D extracted: 127k
  - Size RLE list: 146k
  - Render time: 1.23s
- Turbulent Jet 4D BCC
  - 138 time steps (87M)
  - Relevant voxels: 7.4M
  - 3D extracted: 107k
  - Size RLE list: 146k
  - Render time: 1.01s (71%)

(Neophytou '02)

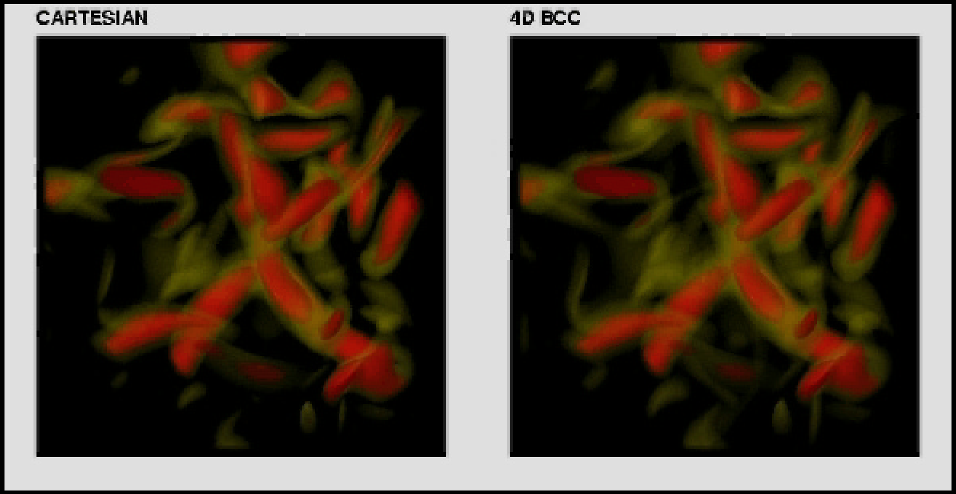
Alternative Grids

- Animations of time-varying datasets:



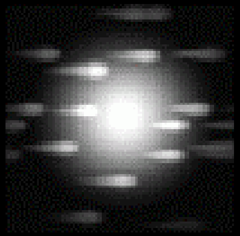
turbulent jet

turbulent flow

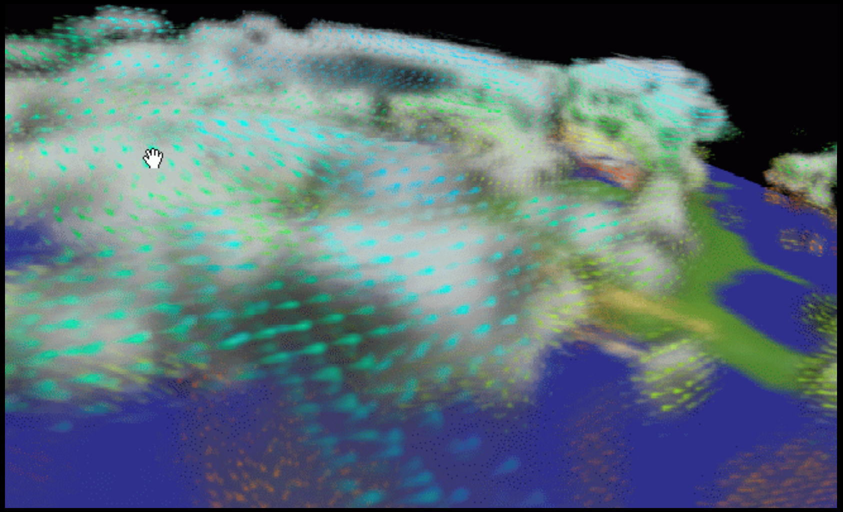


Detail Modeling

- Footprints do not have to serve interpolation alone (via the pre-integrated kernel function)
- They can be used to add additional detail or information between the sample points
- The Gaussian footprint provides the blending



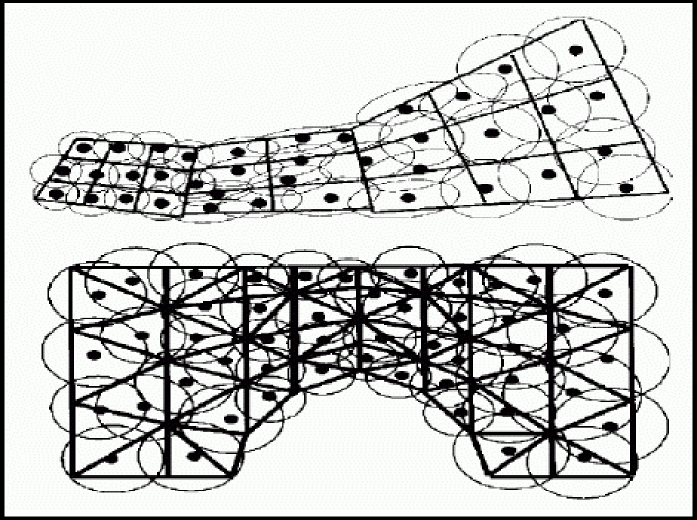
vector field splat



(Crawfis/Max '93)

Space-Filling Points

- Points can also be used to “stuff” empty space
- Example:
  - One may fill cells of an irregular grid with Poisson distributed points
  - Perform projection via point-based rendering



(Mao ’95)

Questions?