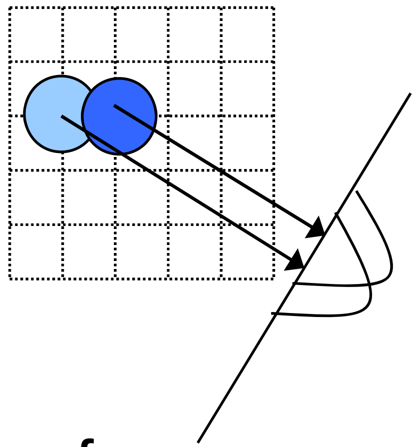
What we will cover

- Contour Tracking
- Surface Rendering
- Direct Volume Rendering
- Isosurface Rendering
- Optimizing DVR
- Pre-Integrated DVR
- Splatting
- Unstructured Volume Rendering
- GPU-based Volume Rendering

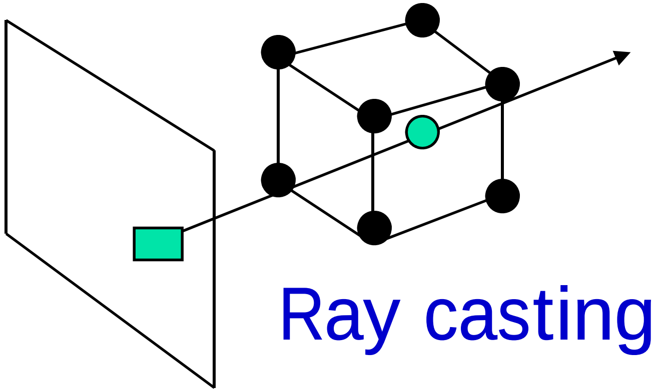
Splatting Algorithm

- Distributes volume data values across a region on the rendered image in terms of a *distribution function* -typically Gaussian functions
- Object-order algorithm
- Front-To-Back or Back-To-Front
- Original method is fast, but quality is poor.

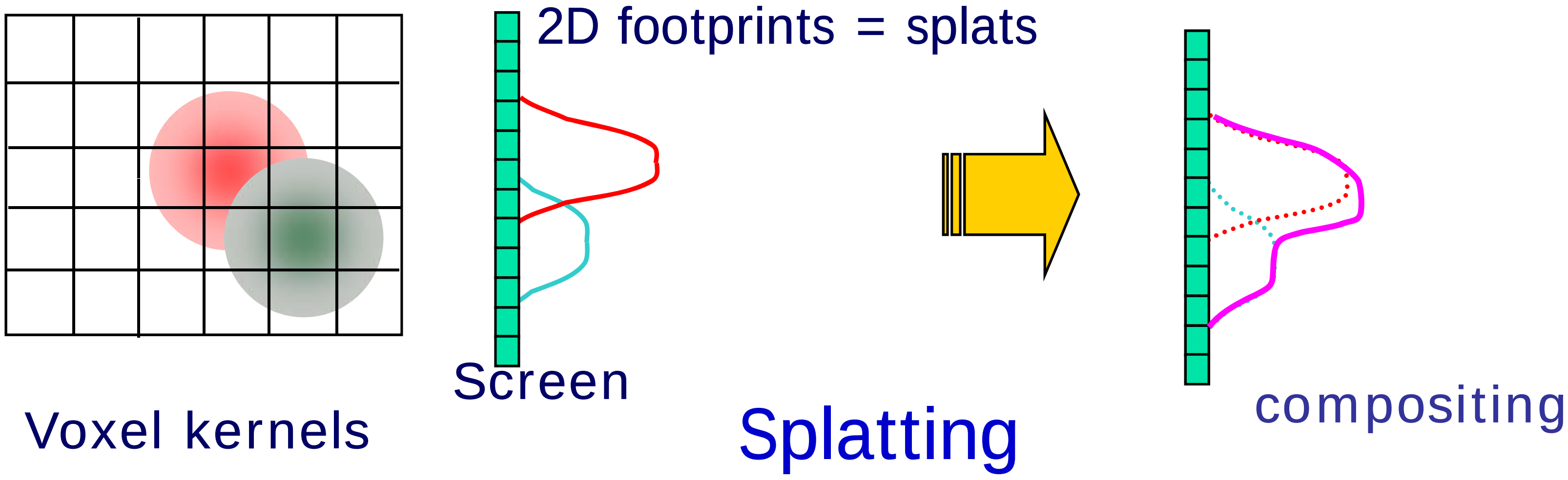
  Many improvements since first publication
- Reading: Lee Westover, “Footprint Evaluation for Volume Rendering”, Siggraph 1990



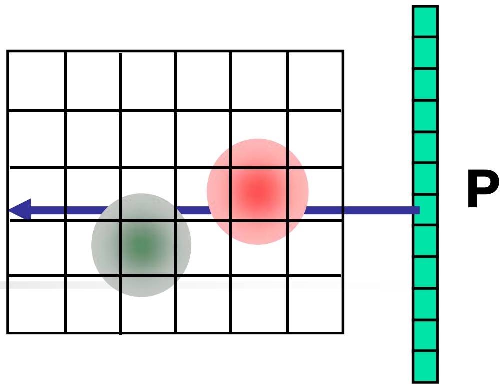
Ray Casting vs. Splatting



- Rendering is expensive (trilinear interpolation)
- Resampling might miss some voxels and thus cause errors



Splatting



- Ray color for pixel **p** of image is given by the integral along the ray

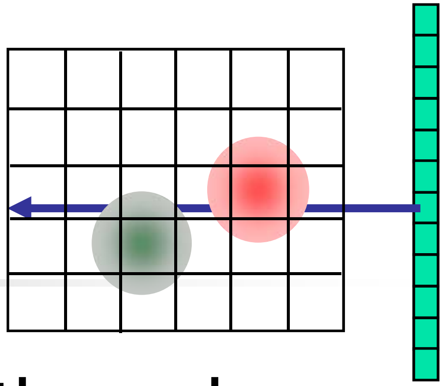
$$I(p) = \int f(p+s)ds$$

where $f(r) = f(p+s)$ is the density function,

s is a vector along the ray

- Since ***r*** can be anywhere in the 3D continuous space, *f(**r**)* is not known and must be reconstructed from discrete voxels.

Splatting



- The reconstruction can be done through

$$f(r) = \sum_k w(r - r_k)\, f(r_k)$$

Where $f(r_k)$ is the density value at sample point r_k

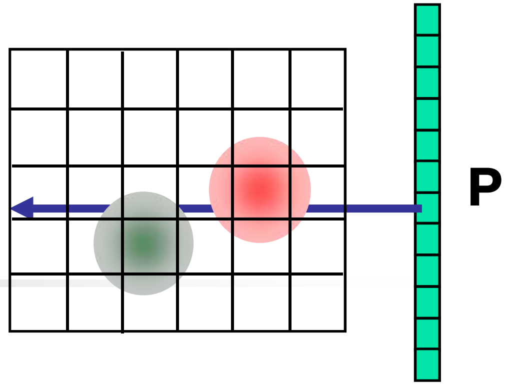
$w(r\text{-}r_k)$ is the reconstruction kernel,

such as a Gaussian functions

$$\text{Thus}\quad I(p) = \int f(r)\,ds$$

$$= \int \sum_k w(p + s - r_k) f(r_k)\,ds$$

Splatting



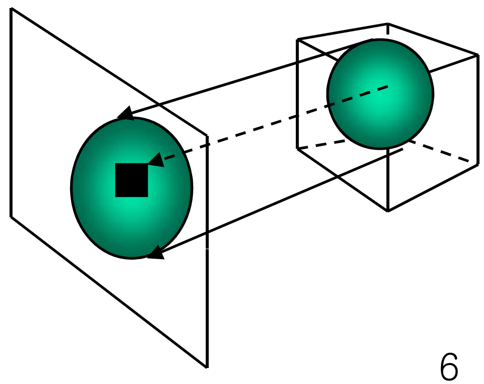
$$I(p) = \int \sum_k w(p + s - r_k) f(r_k) ds$$

$$= \sum_k f(r_k) \int w(p + s - r_k) ds$$

Splat footprint

In x, y, z coordinates (ray parallel to z-axis),
splat footprint is defined as:

$$\text{footprint}(x, y) = \int w(x, y, z) dz$$



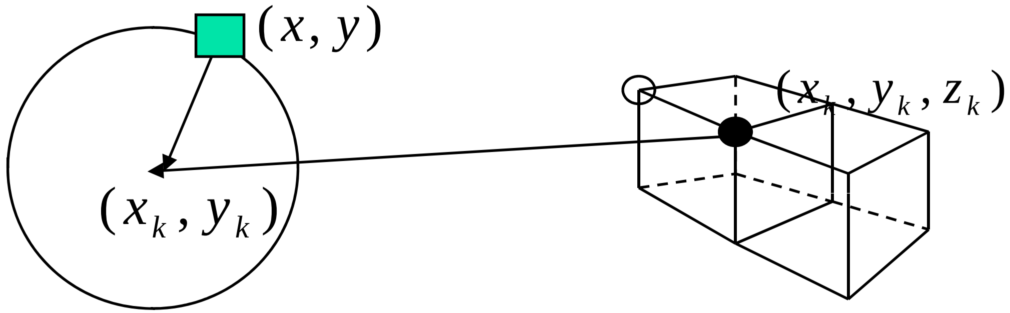
Splatting

The ray color is given by

$$I(x, y) = \sum_k f(r_k)\,\text{footprint}(x - x_k, y - y_k)$$

(x,y) is the pixel's location

(x_k, y_k) is the image plane location of the sample



The final value of pixel (x,y) will be a total sum of the contributions from its surrounding voxel projections

Splatting

$$I(x,y) = \sum_k f(r_k)\,\mathrm{footprint}(x - x_k, y - y_k)$$

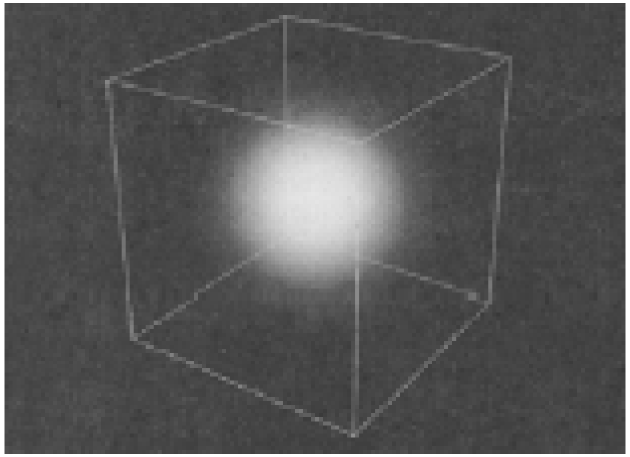
- $\mathrm{footprint}(x - x_k, y - y_k)$ defines the **weight** of a voxel's contribution to the rendered image
- Splat footprint, a function of x and y, can be pre-computed and stored
- Rendering just needs to traverse all voxels

Reconstruction Kernel

- Splat footprint is integral of reconstruction kernel *w*
- *For Gaussian reconstruction kernel*, splat footprint has elliptic shape
- In particular if Gaussian reconstruction kernel is spherical (a=b=c), splat footprint is isotropic

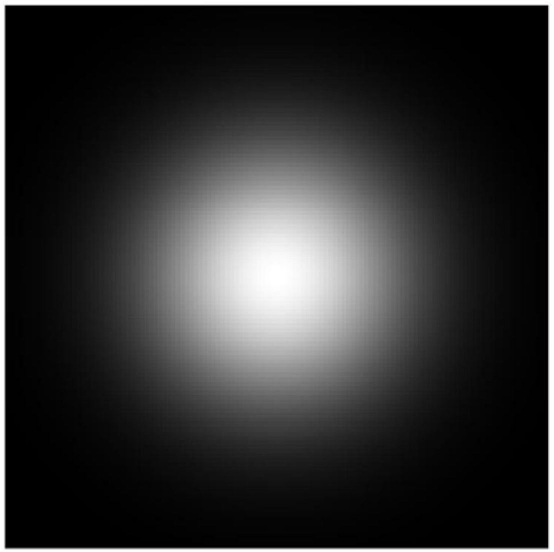
$$\text{footprint}(x, y) = \int w(x, y, z)\,dz$$

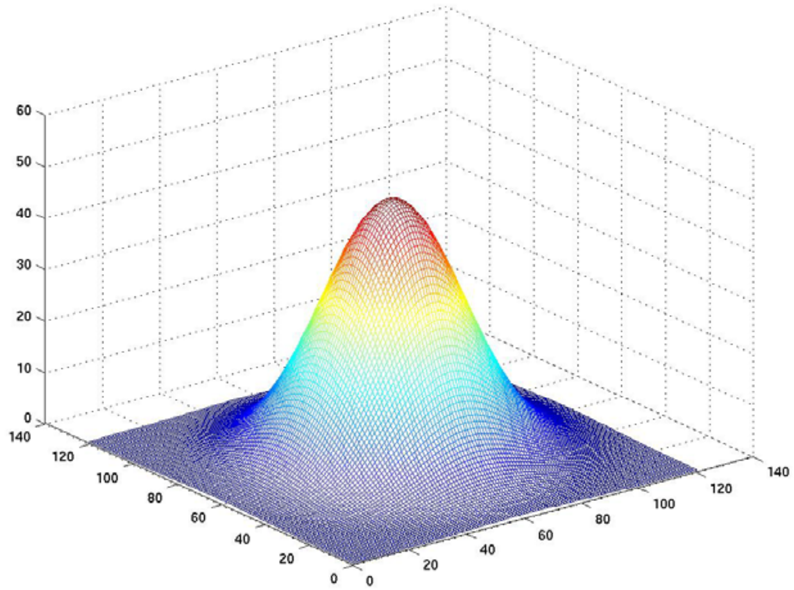
$$w(x, y, z) = d \exp\left(-\left(\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2}\right)\right)$$



Generic footprint function – Gaussian example

$$f(x, y) = \int e^{-0.5(x^2+y^2+z^2)} dz$$





Splatting

- Footprint table per view requires too much computation time
- For a regular grid (uniform sample intervals), in orthographic projections, the footprint of each sample is the same except for an image plane offset.

→ Use a generic footprint table
Generate a view-transformed footprint table

Spherical Kernels (generic footprint table)

For orthographic view

- projection of reconstruction kernel
- mapping to generic footprint table

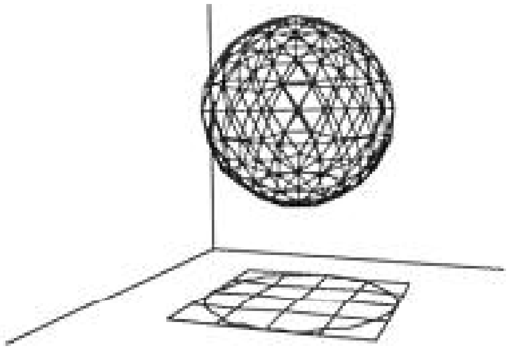
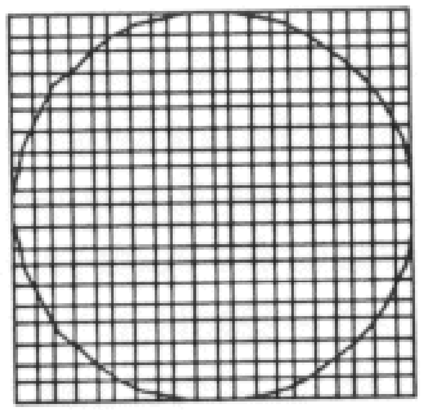
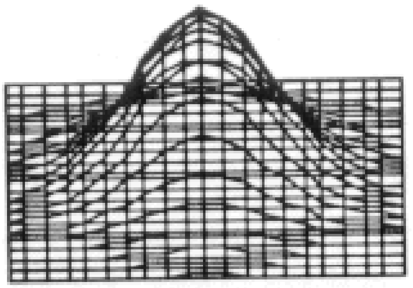


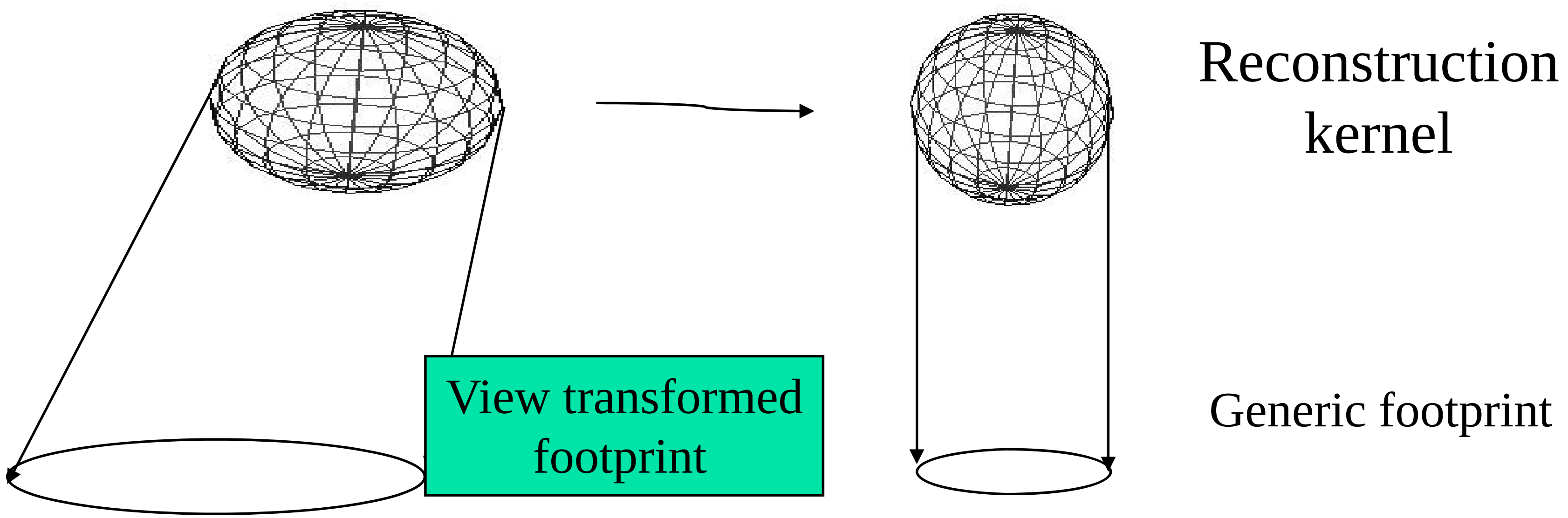
Figure 3. Spherical Kernel





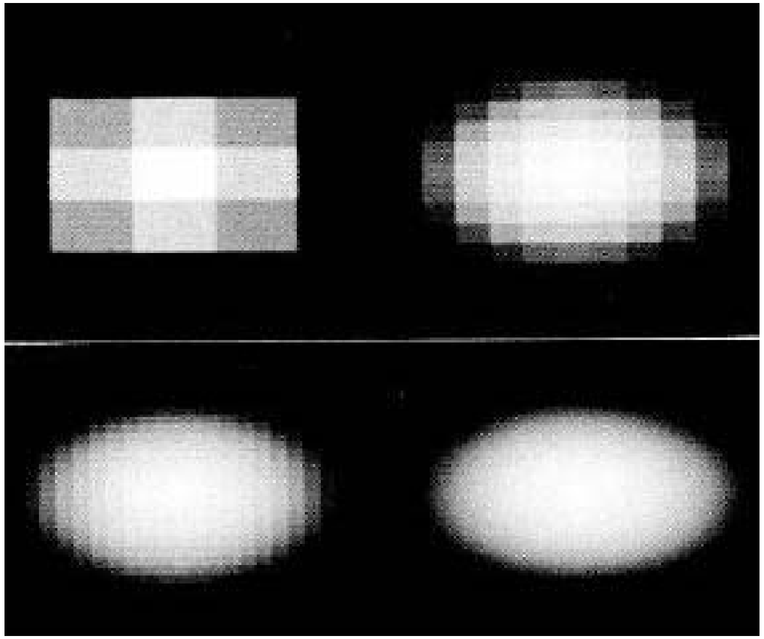
View Transformed Footprint

- A spherical reconstruction kernel would be transformed to an ellipsoid if there is a difference in the scaling factors between the axis. This ellipsoid would always mapped to an elliptic footprint function.
- The ellipsoid is more general then the spherical case.



Results

the size of the footprint tables and the resultant artifacts in the images.

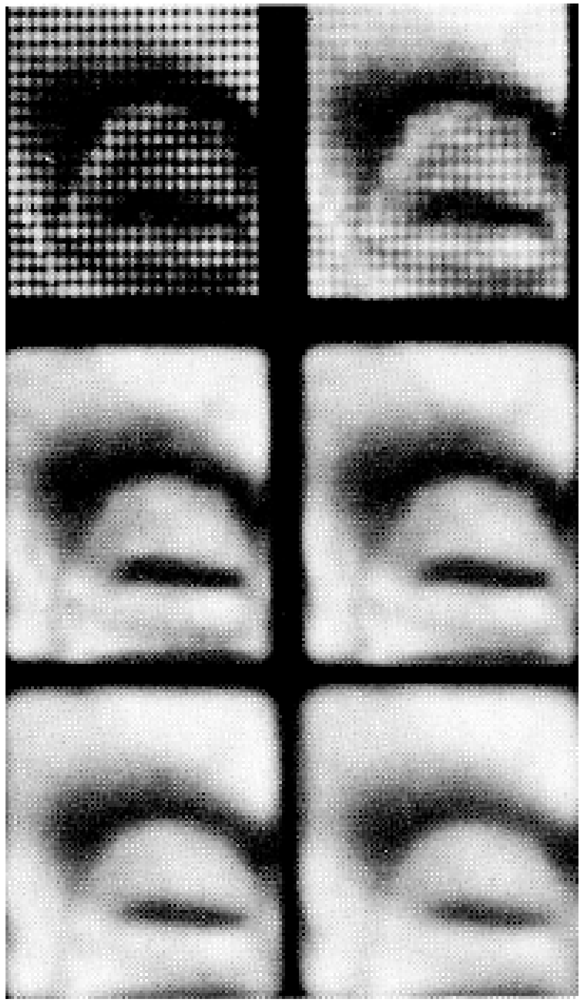


5×5 11×11

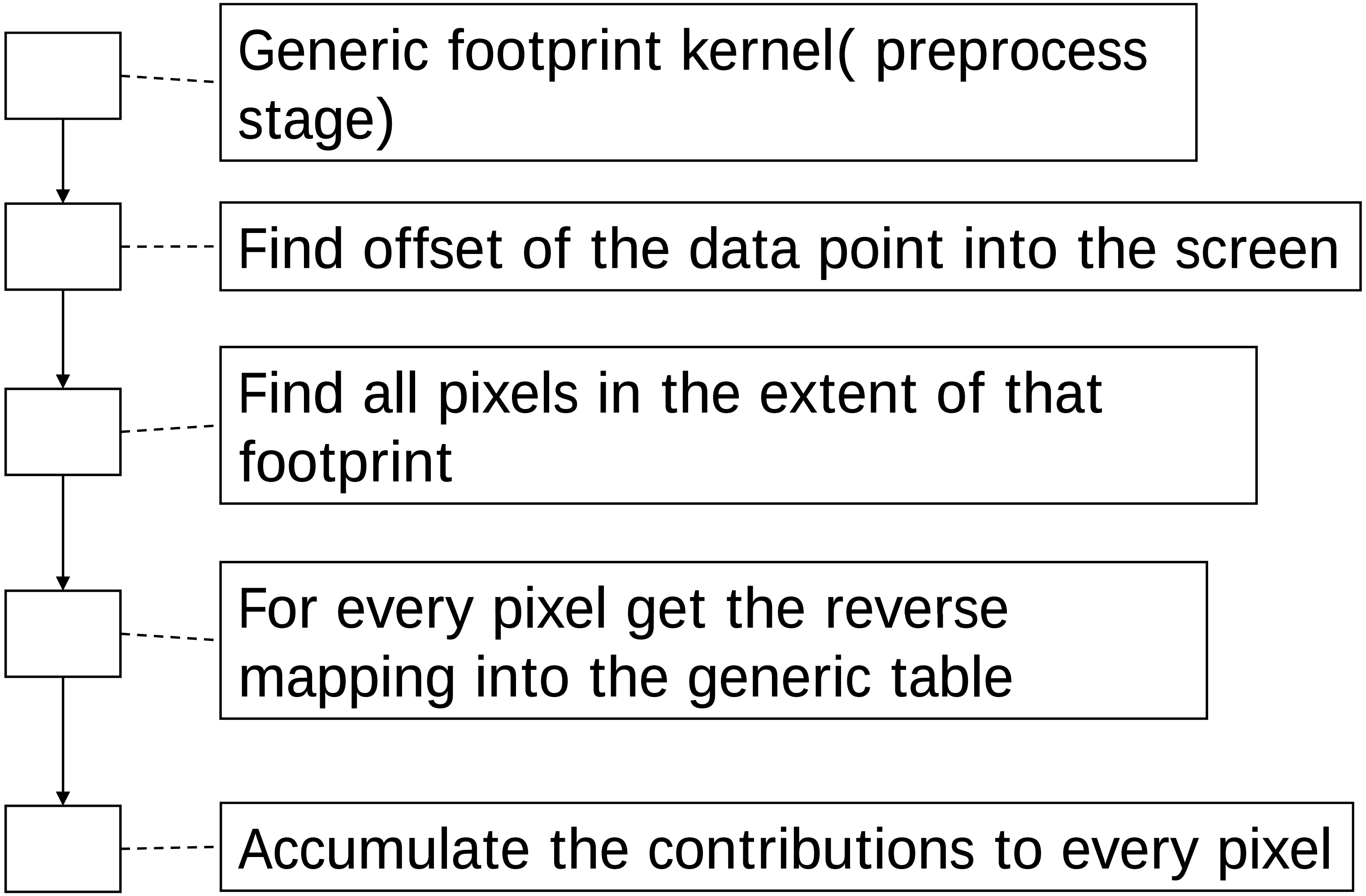
21×21 101×101

Effects of Kernel Size

- If the kernel size is too small, image has gaps
- Larger footprint ➔ larger spatial kernel extent ➔ lower frequency components ➔ more blurring

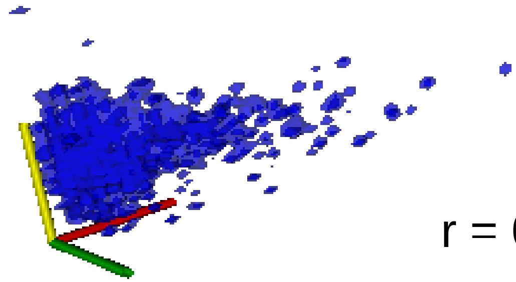


The Graphic pipeline

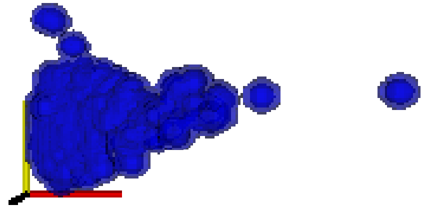


Examples

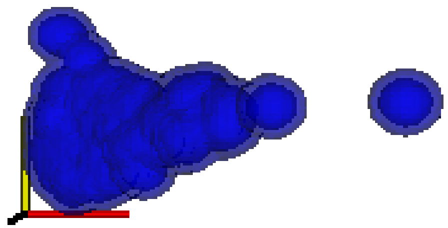
- The examples show the result of splatting algorithm with Gaussian reconstruction kernel , the radii of the kernel vary from 0.01 to 0.5.
- The kernel was rendered using coloring and opacity adjustment.



r = 0.01



r=0.1



r=0.5

Splatting Algorithm

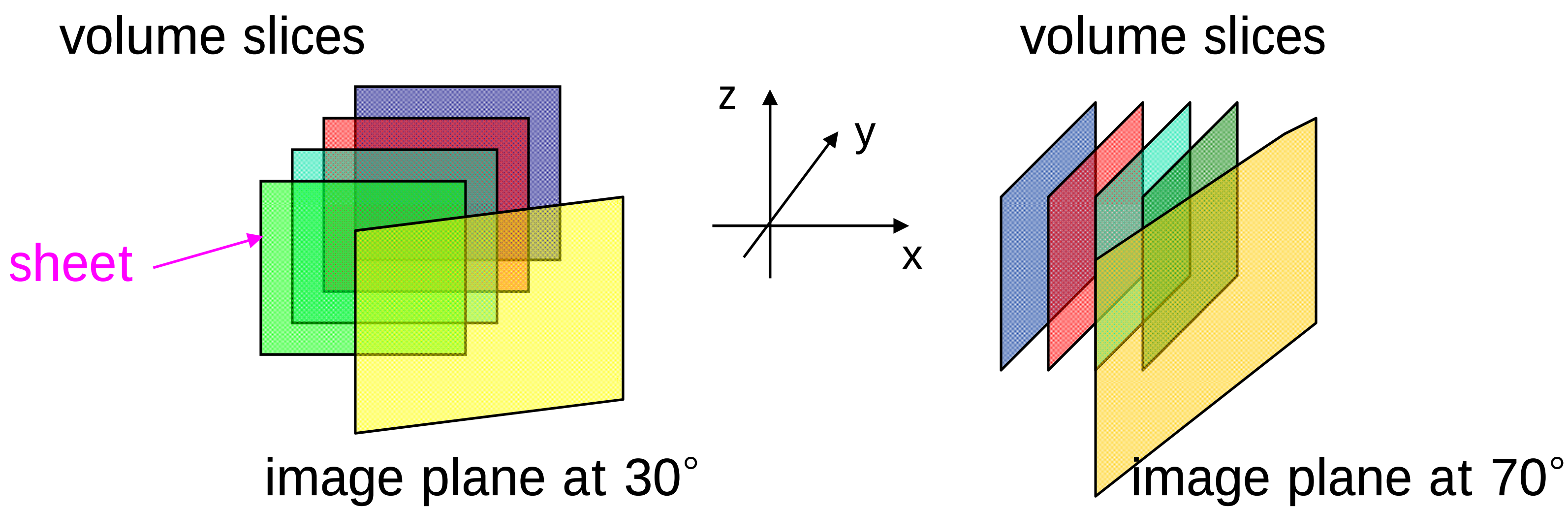
- Efficient by keeping only relevant voxels
- Rendering image quality is similar to ray casting but smoother
- Good for large volume
- Ray casting is faster than splatting for data sets with a high number of contributing samples. But rendering is expensive (particularly trilinear interpolation)
  - Interpolation task of splatting: $O(n^2)$ in image plane parallel space
  - Interpolation of ray casting: $O(n^3)$ in volume space
- Applicable to both regular and irregular datasets
- Good for parallel computing

Splatting Algorithm

- Disadvantage
  - The use of pre-integrated footprints
    - → visibility incorrect
    - → compositing not accurate
  - The use of Gaussian filter (large, symmetric)
    - → blurring effect
  - Perspective projection is slow
    - The splats must be scaled according to distance

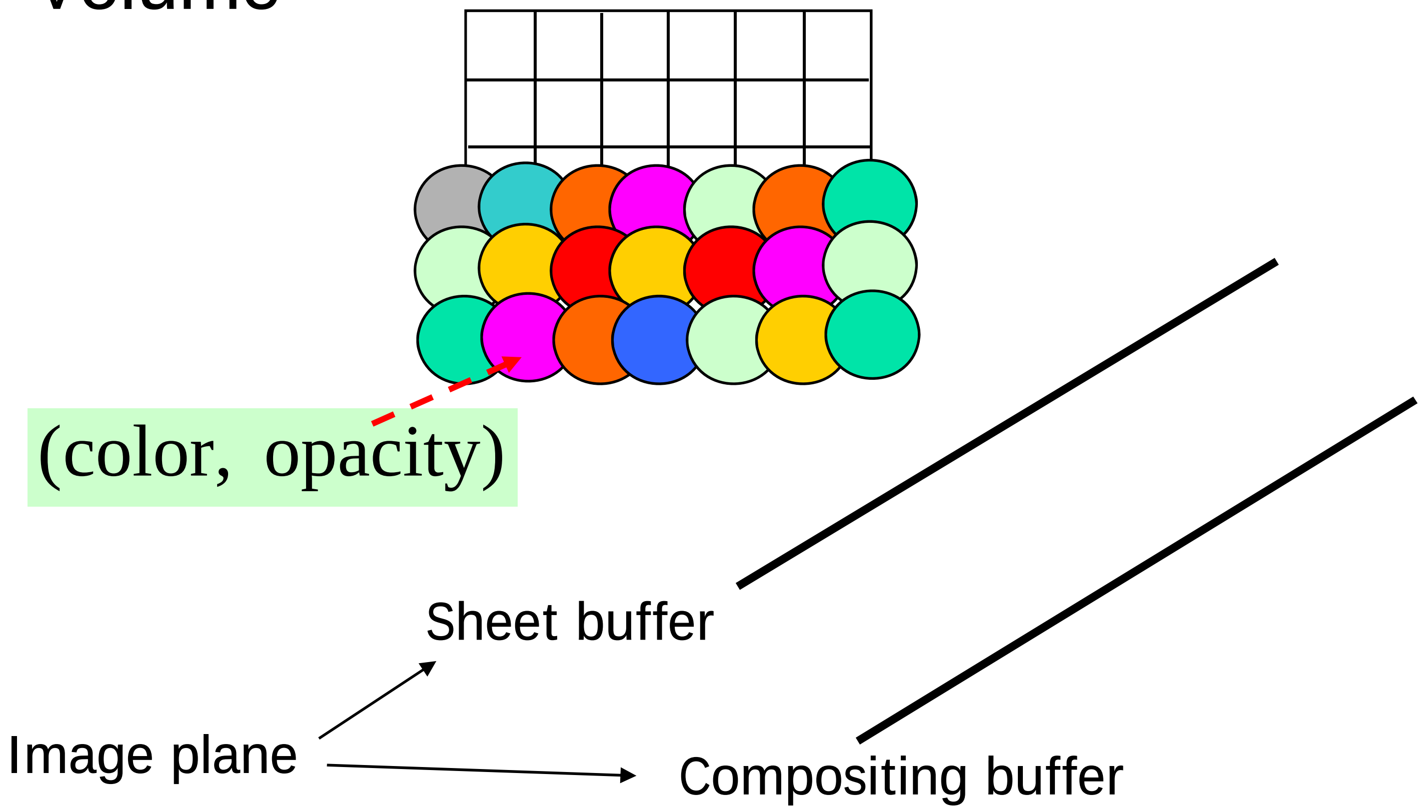
Early Implementation - Axis Aligned Splatting

- Sheets are volume slices most parallel to image plane
- Voxel kernels are added within sheets
- Sheets are composited front-to-back
- Reduce color bleeding artifacts



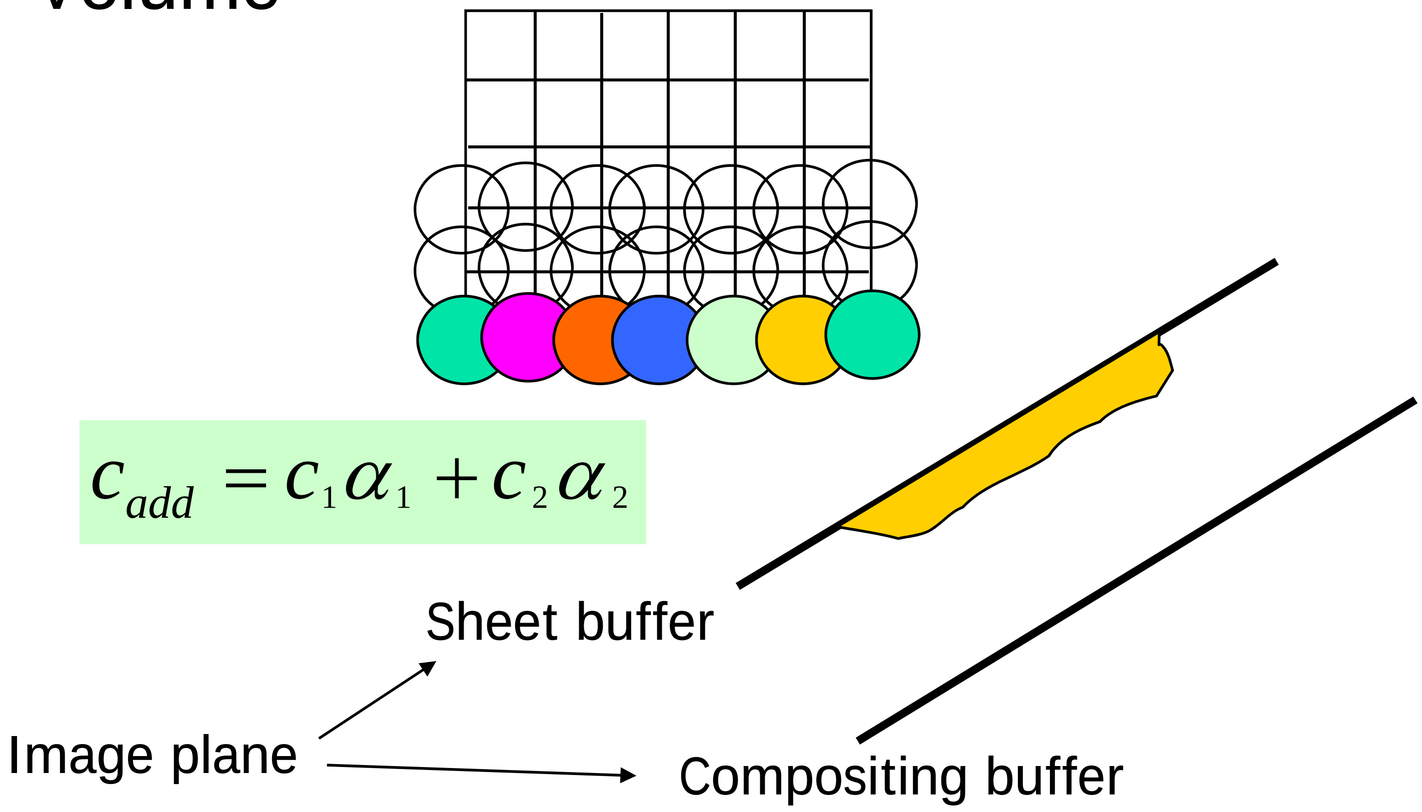
Axis Aligned Splatting

- Volume



Axis Aligned Splatting

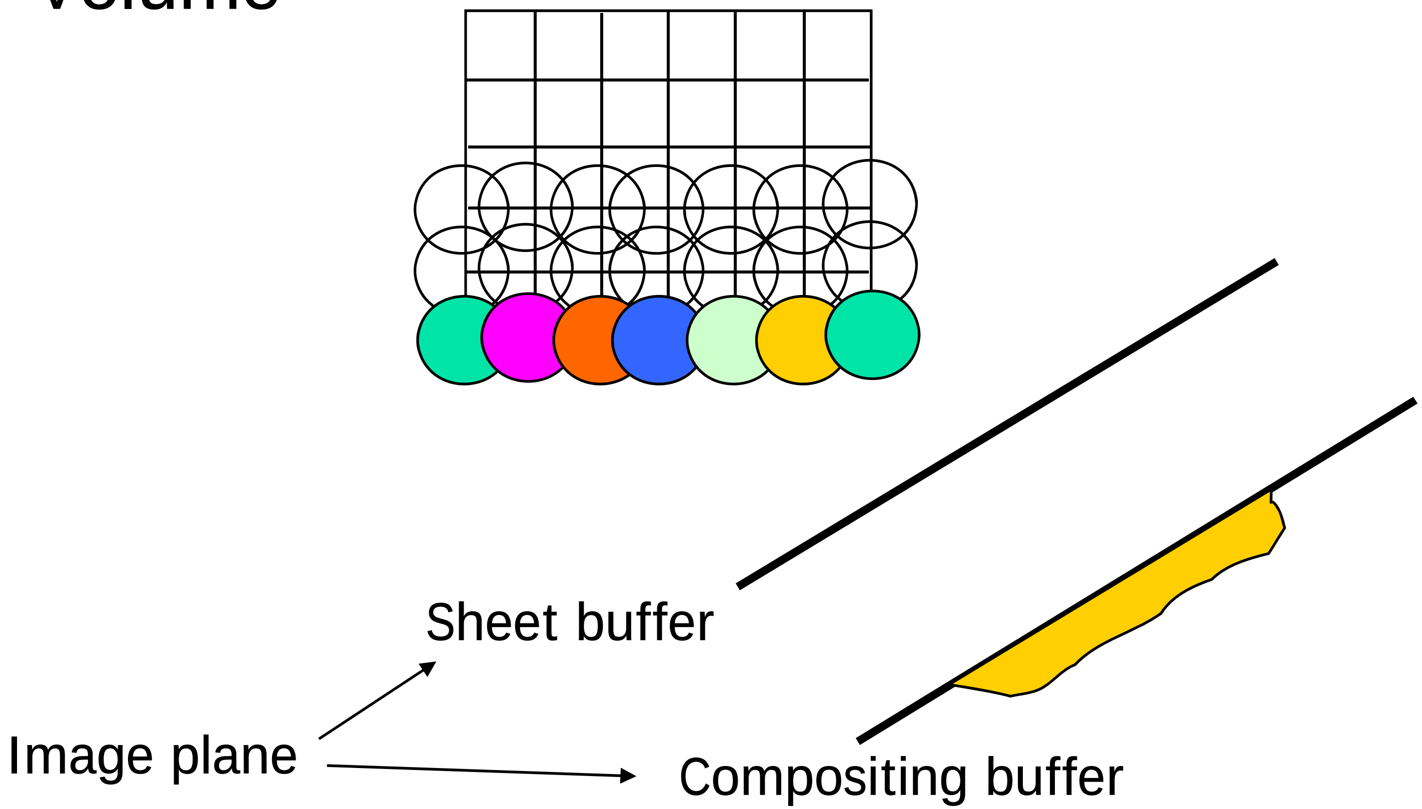
- Volume



$$c_{add} = c_1\alpha_1 + c_2\alpha_2$$

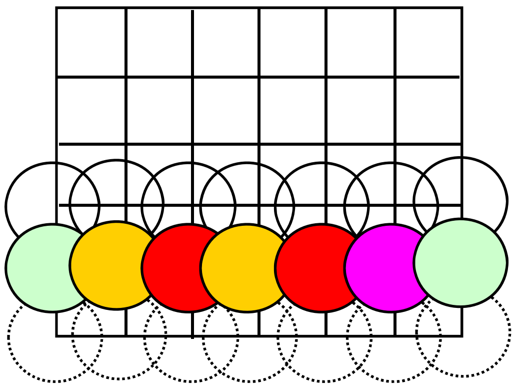
Axis Aligned Splatting

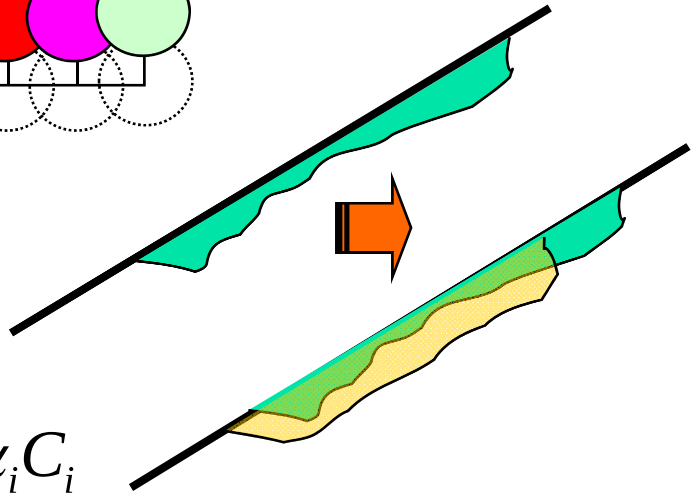
- Volume



Axis Aligned Splatting

- Volume





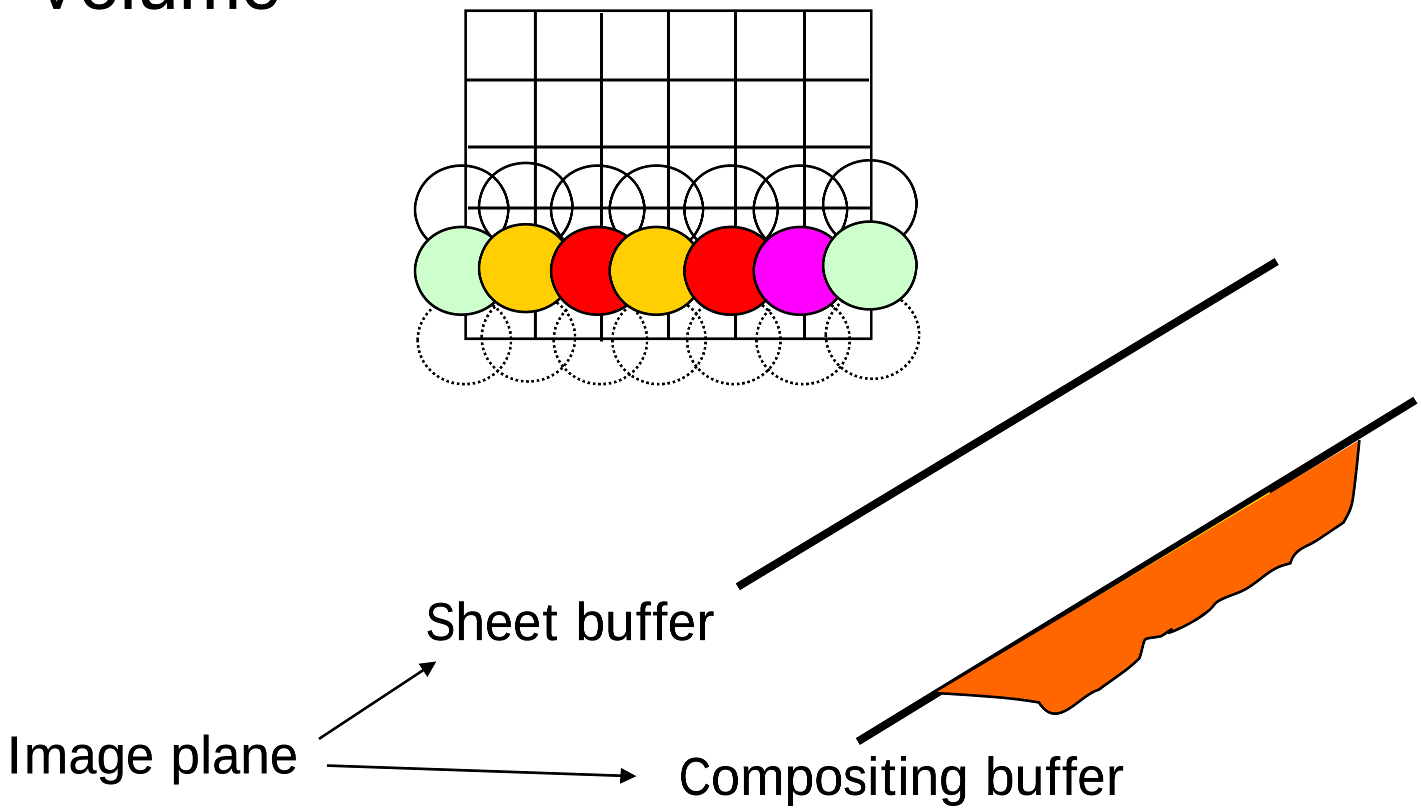
composition

$$I_{out} = I_{in} + (1 - \alpha_{in})\alpha_i C_i$$

$$\alpha_{out} = \alpha_{in} + \alpha_i (1 - \alpha_{in})$$

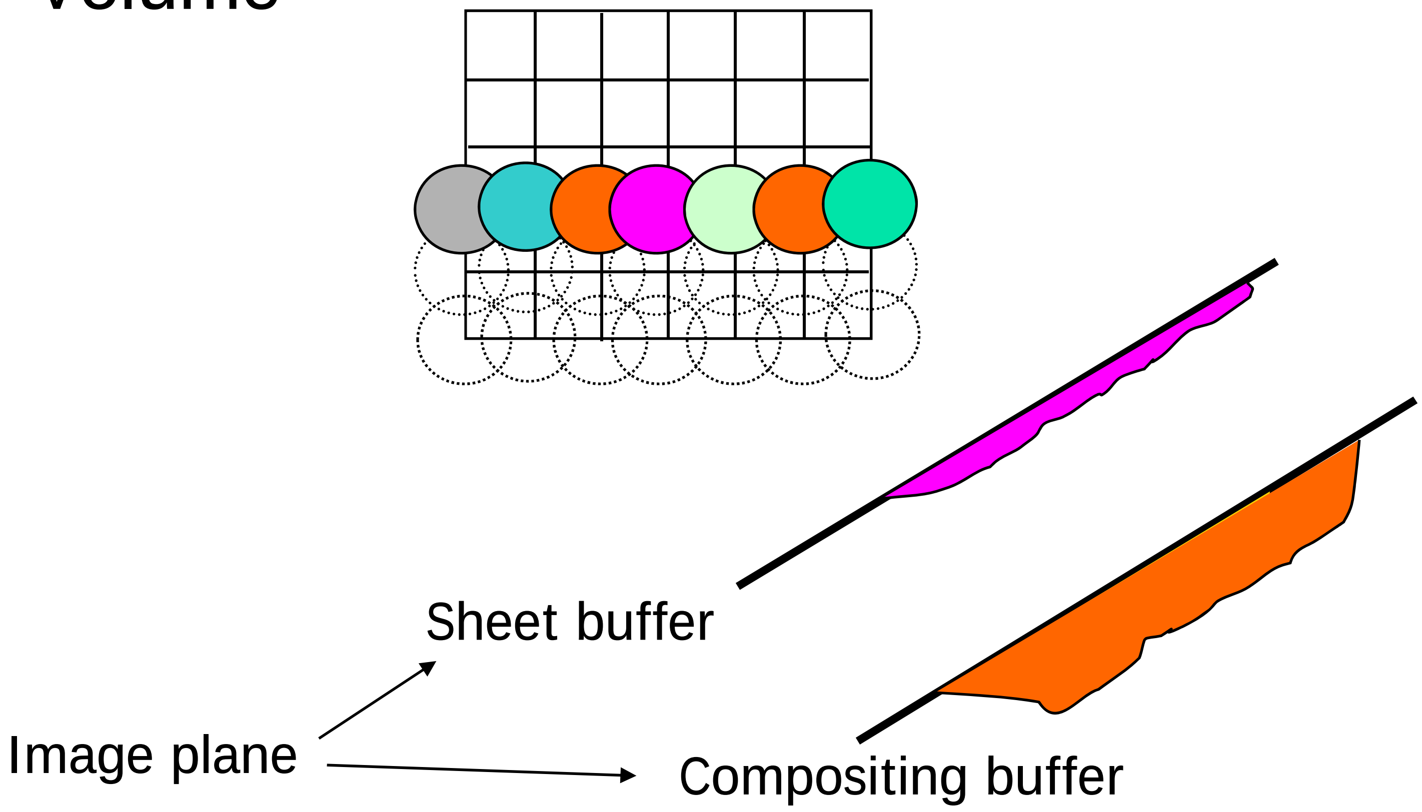
Axis Aligned Splatting

- Volume



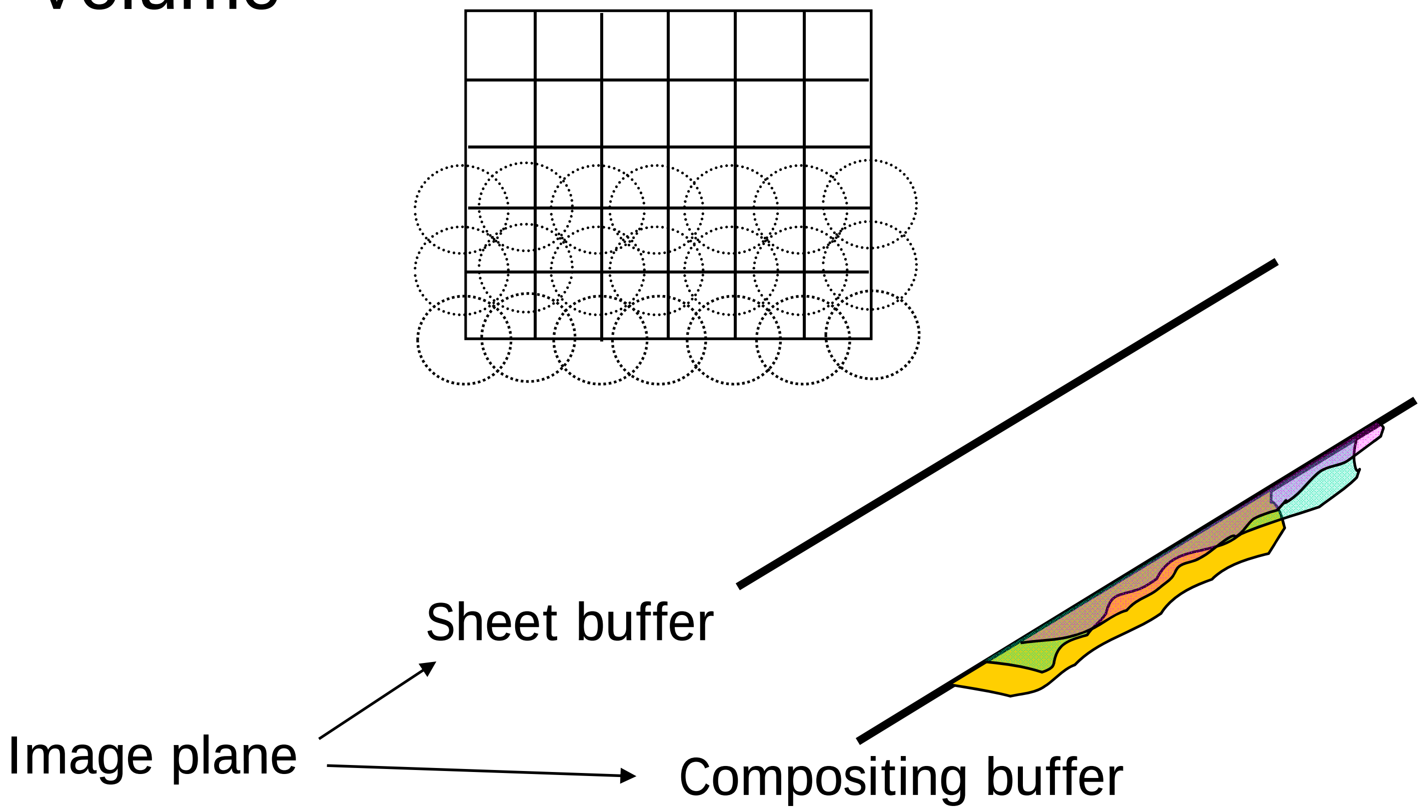
Axis Aligned Splatting

- Volume



Axis Aligned Splatting

- Volume



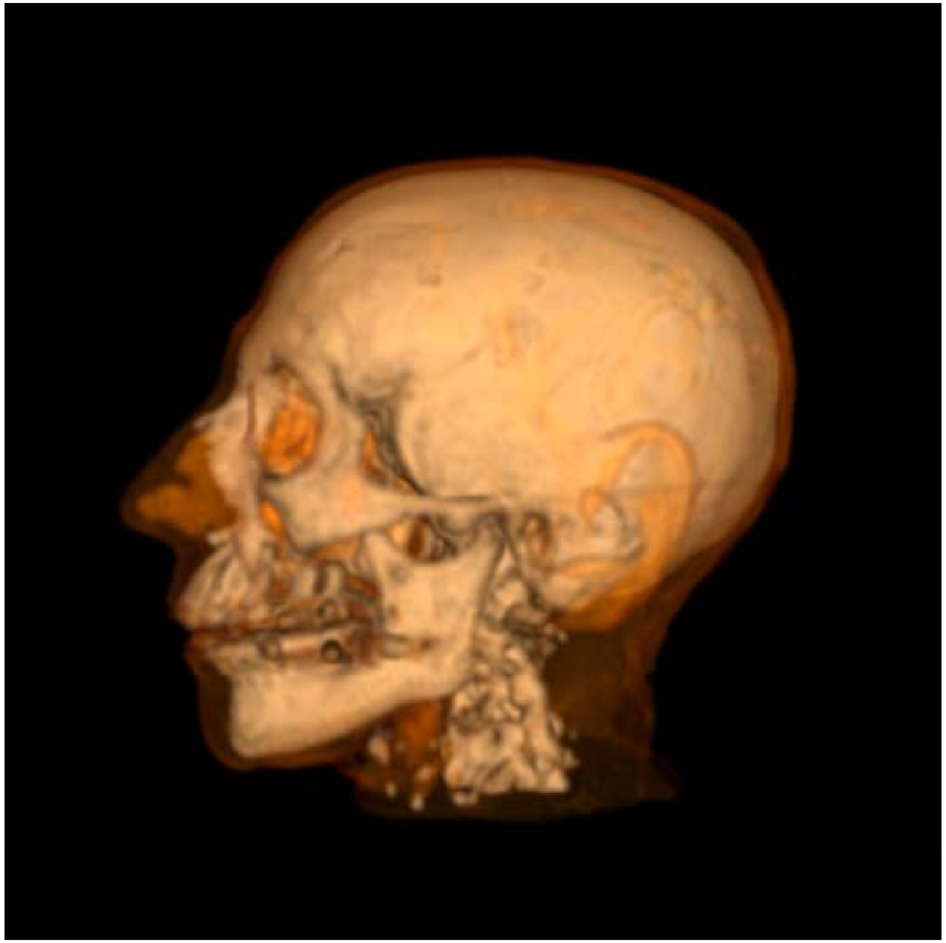
UNC Head: 208x256x225

#Rendered splats:

2,955,242

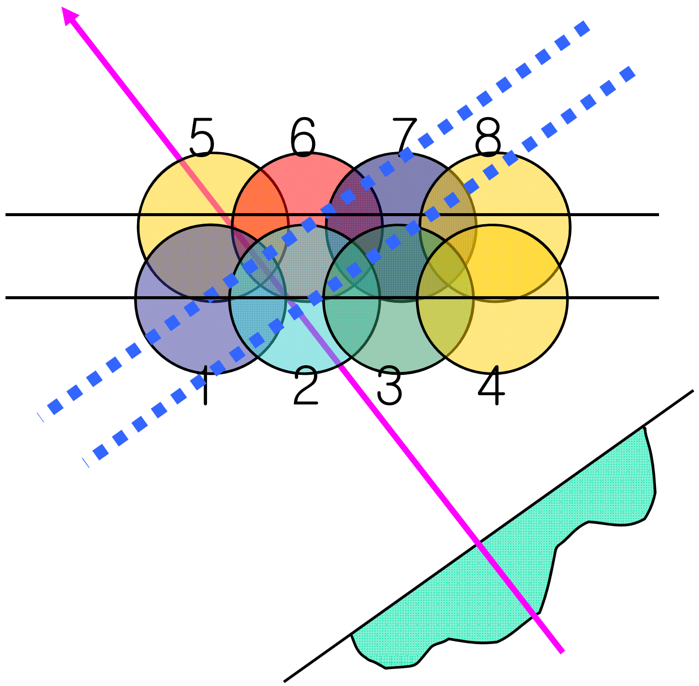
2.86 fps

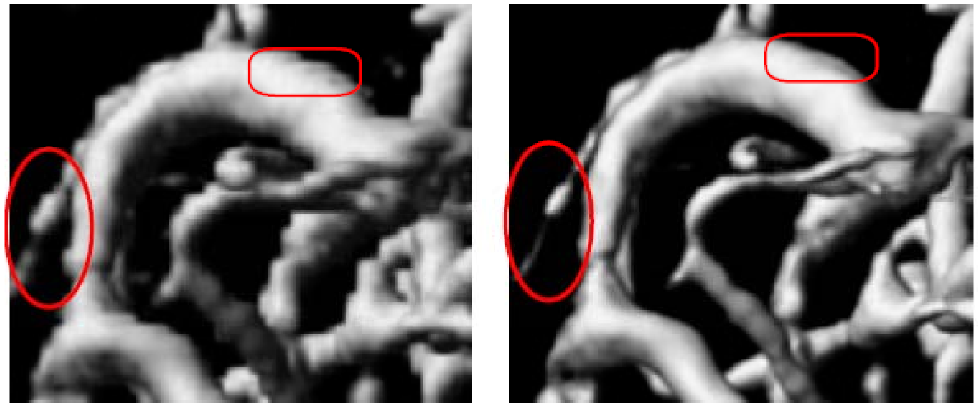
8.5M splats / sec



Problems of Axis Aligned Splatting

- Color bleeding



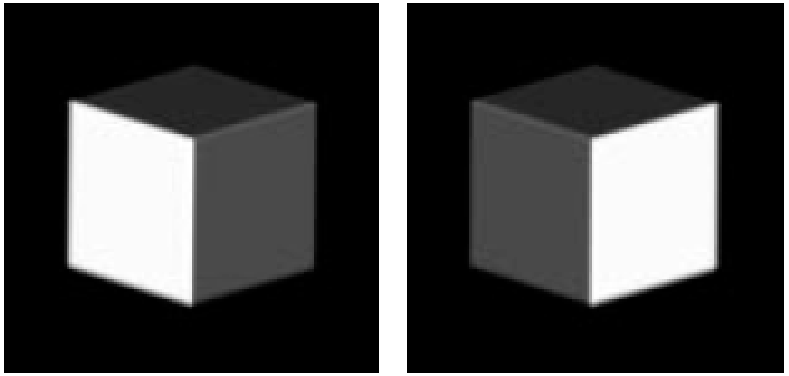


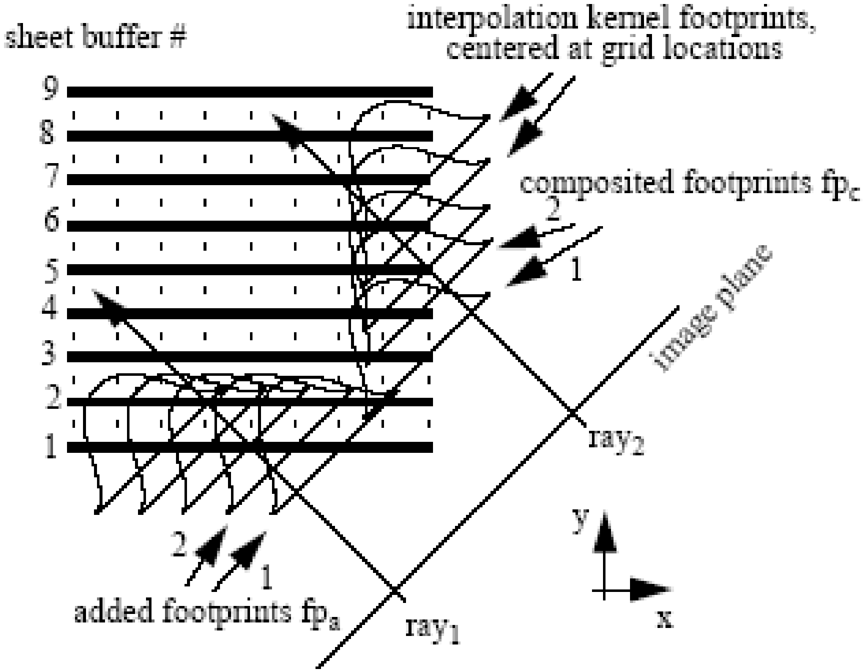
If opacity of *Composite(1&2)* < 1
Then add color 5
➔ (color bleeding)

If opacity of *Composite(1&6)* > 1
Then no contribution of color 5

Problems of Axis Aligned Splatting

- Popping artifacts





$$c_{add} = c_1\alpha_1 + c_2\alpha_2 > c_{composte} = \alpha_2 c_2 (1-\alpha_1) + \alpha_1 c_1$$

ray1 ray2

Image-Aligned Sheet-Buffer

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

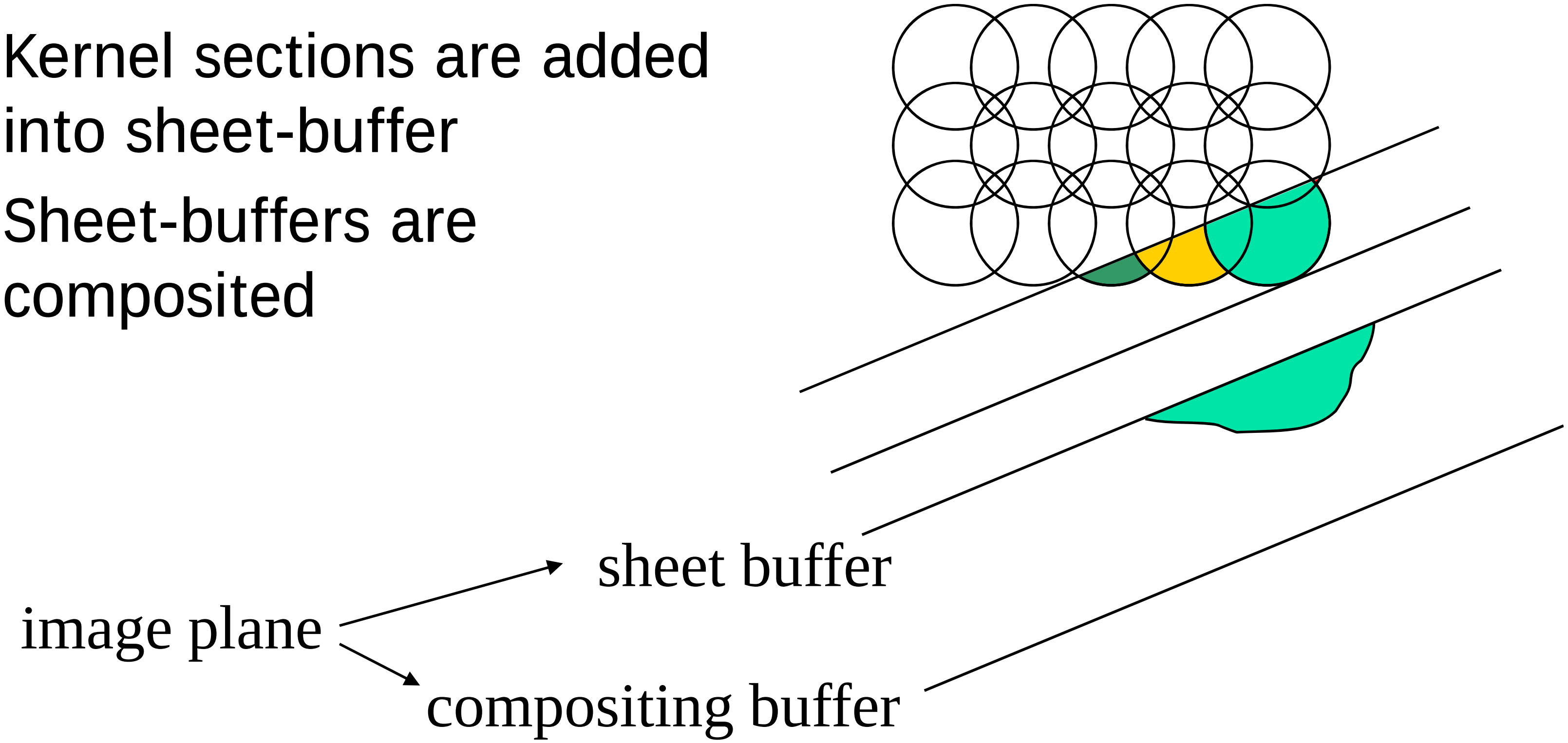


Image-Aligned Sheet-Buffer

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

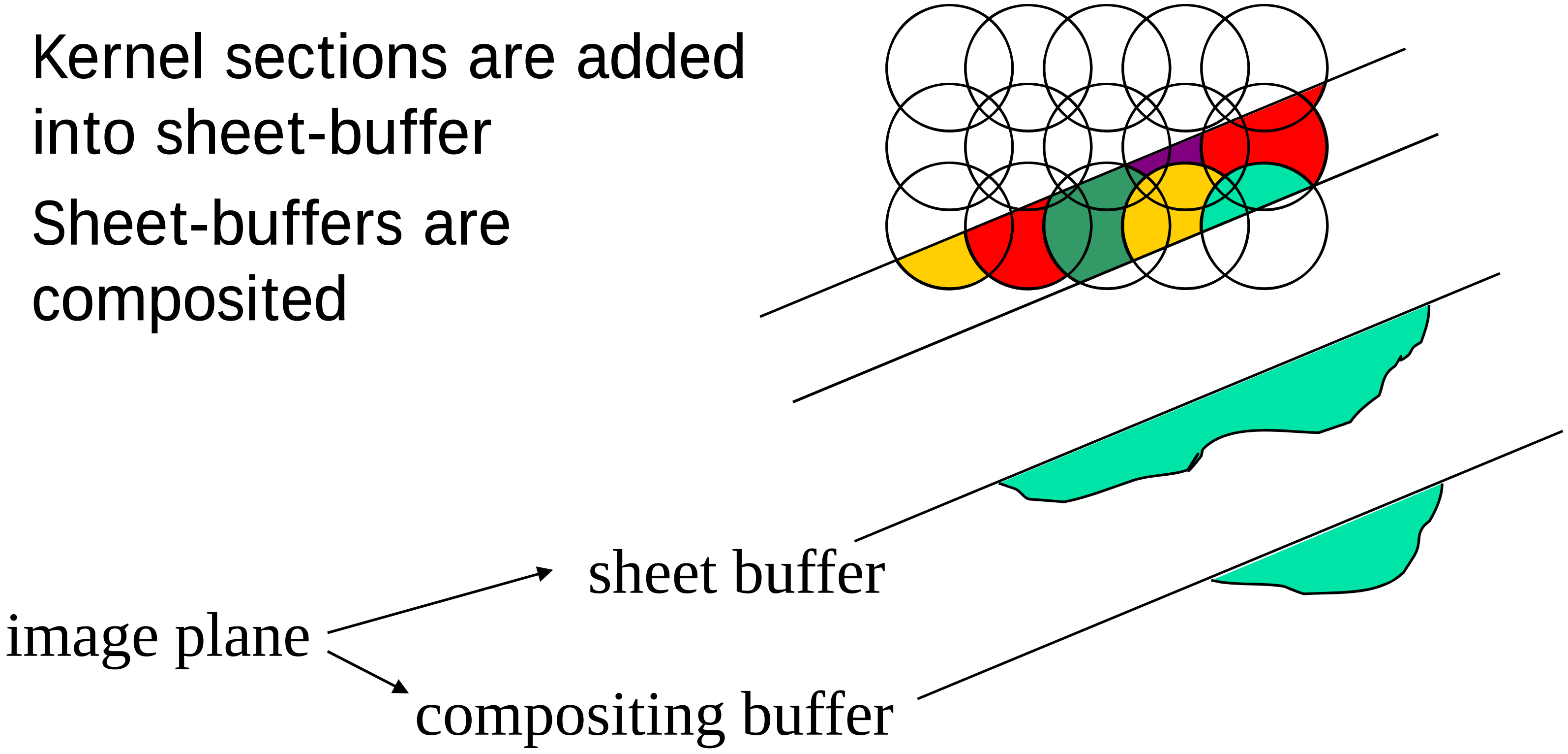


Image-Aligned Sheet-Buffer

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

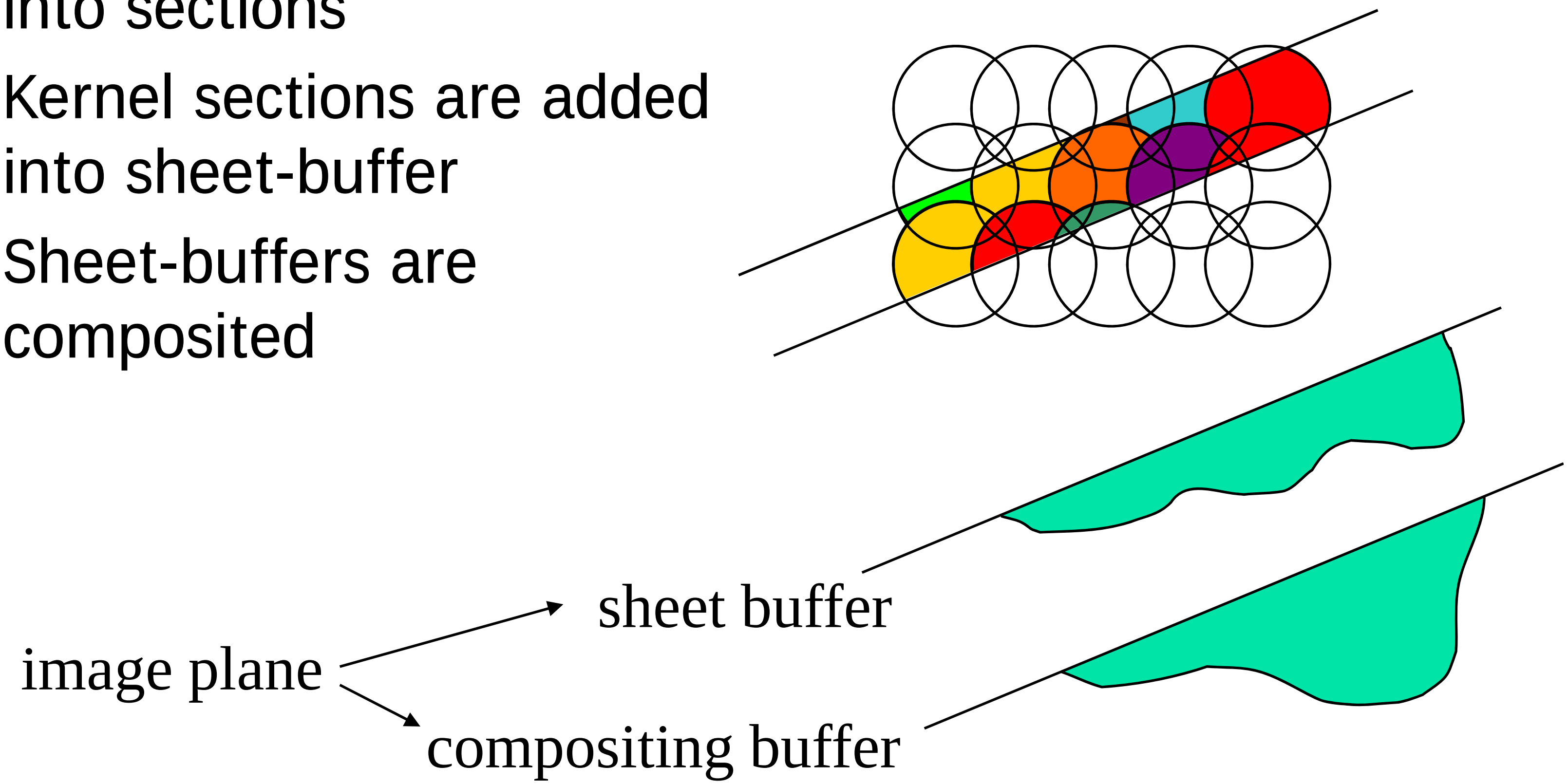


Image-Aligned Sheet-Buffer

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

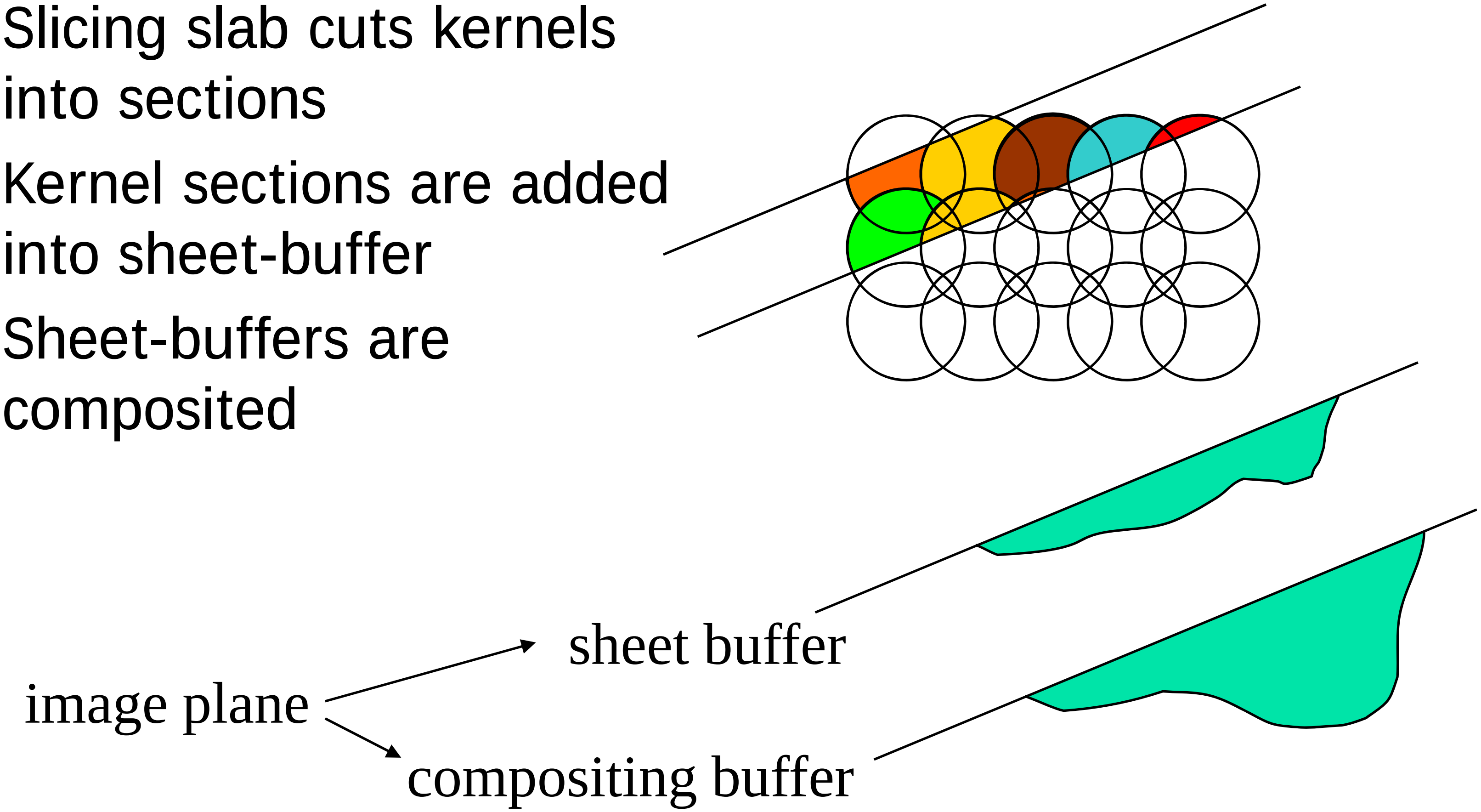


Image-Aligned Sheet-Buffer

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

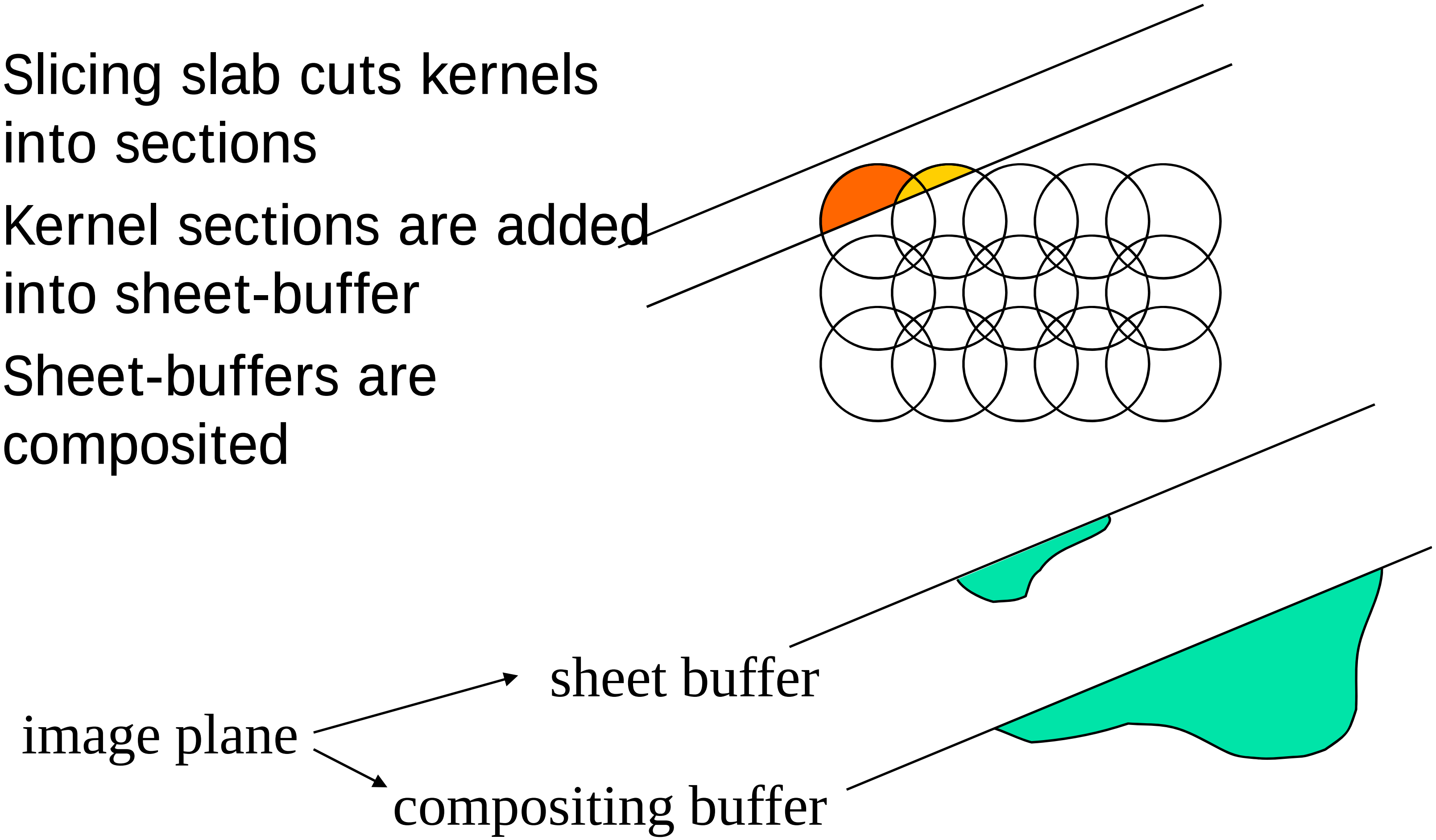


Image-Aligned Sheet-Buffer

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

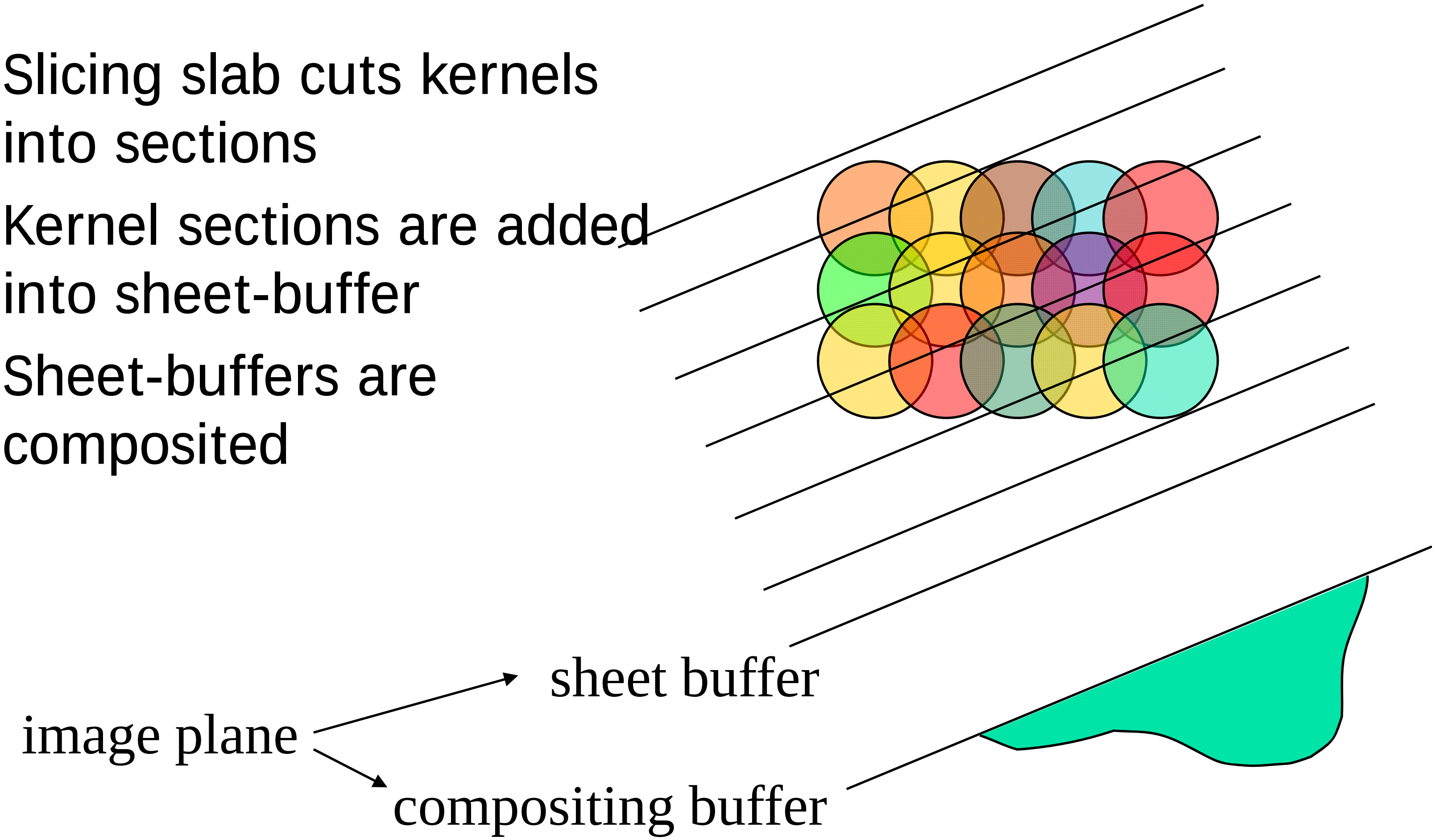
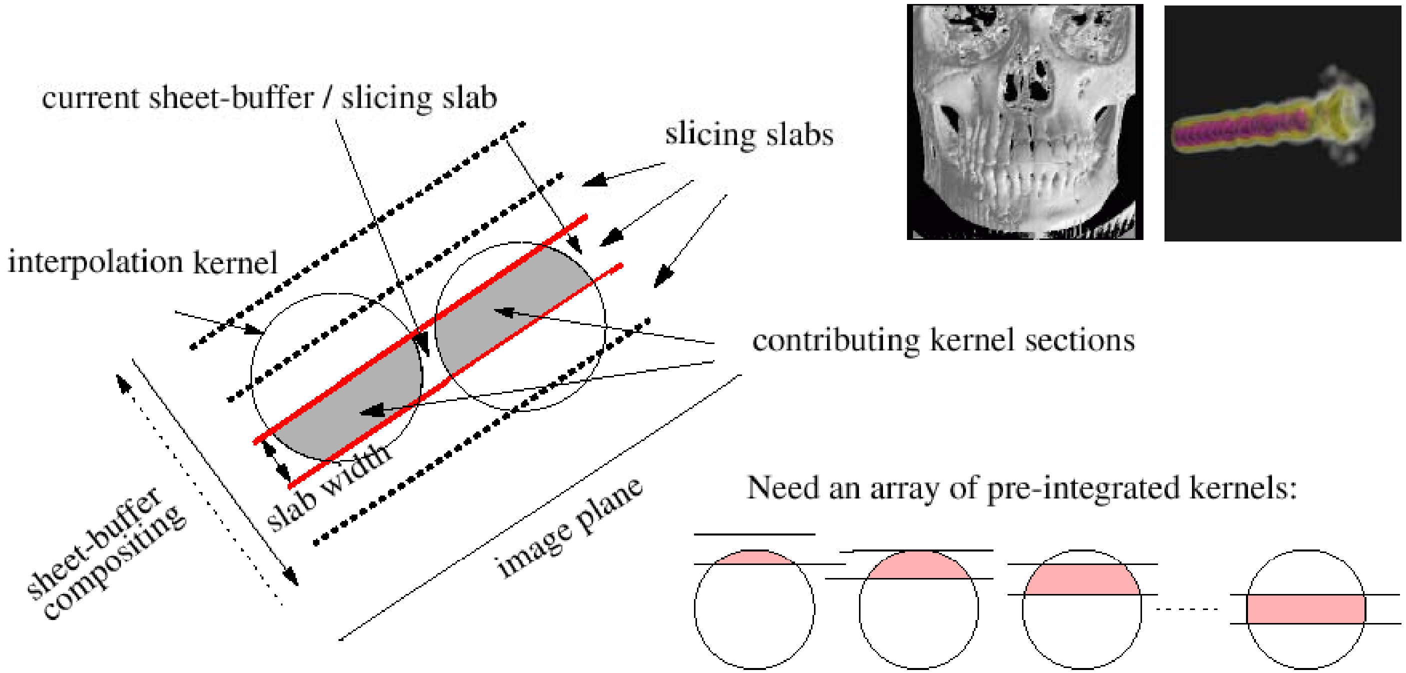


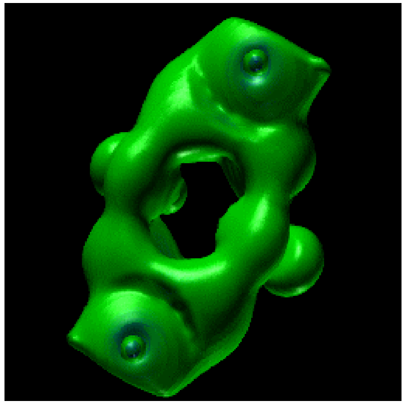
Image-Aligned Splatting

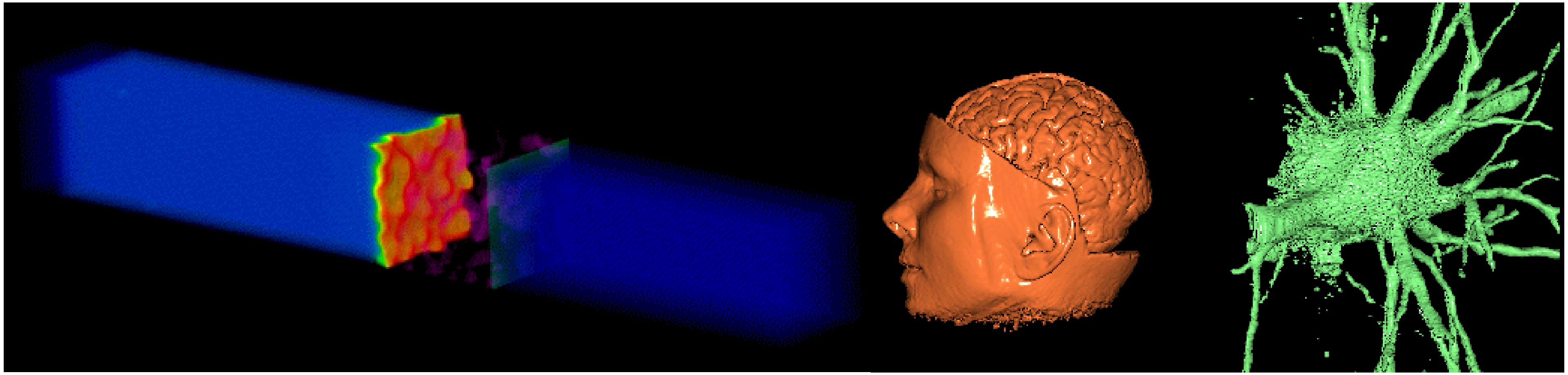
- Note: We need an array of footprint tables now. A separate footprint table for each slice of the 3D reconstruction kernel.



IASB Splatting

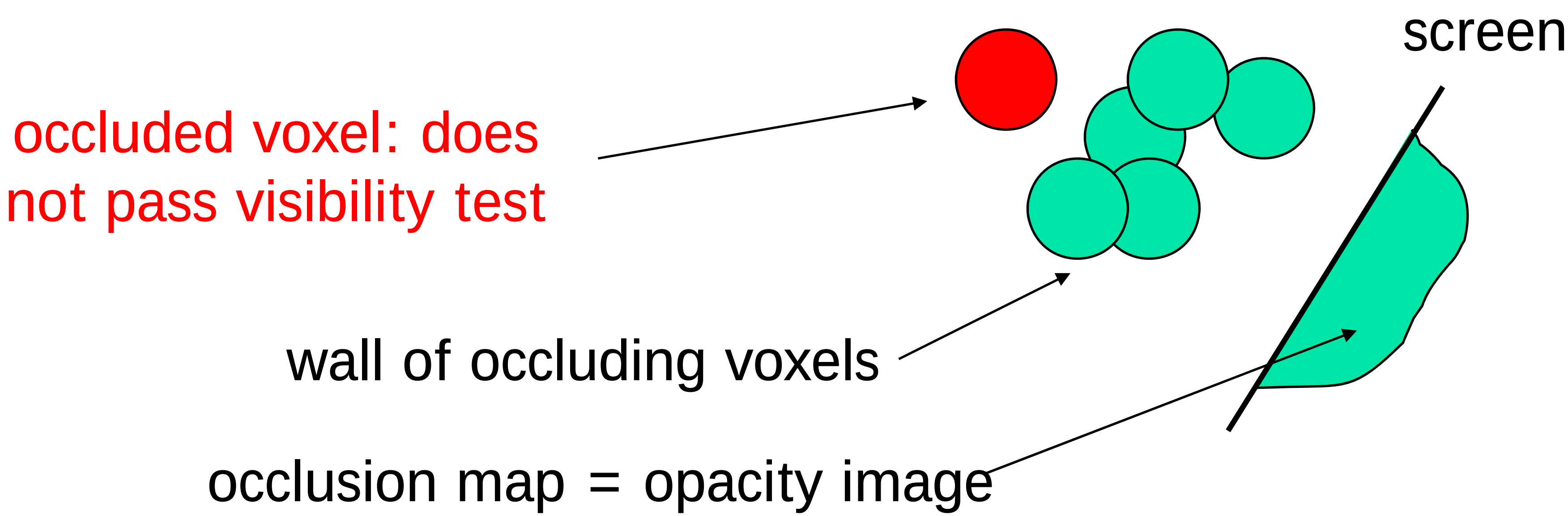
- No popping or color bleeding
- Sharp, noise-free images





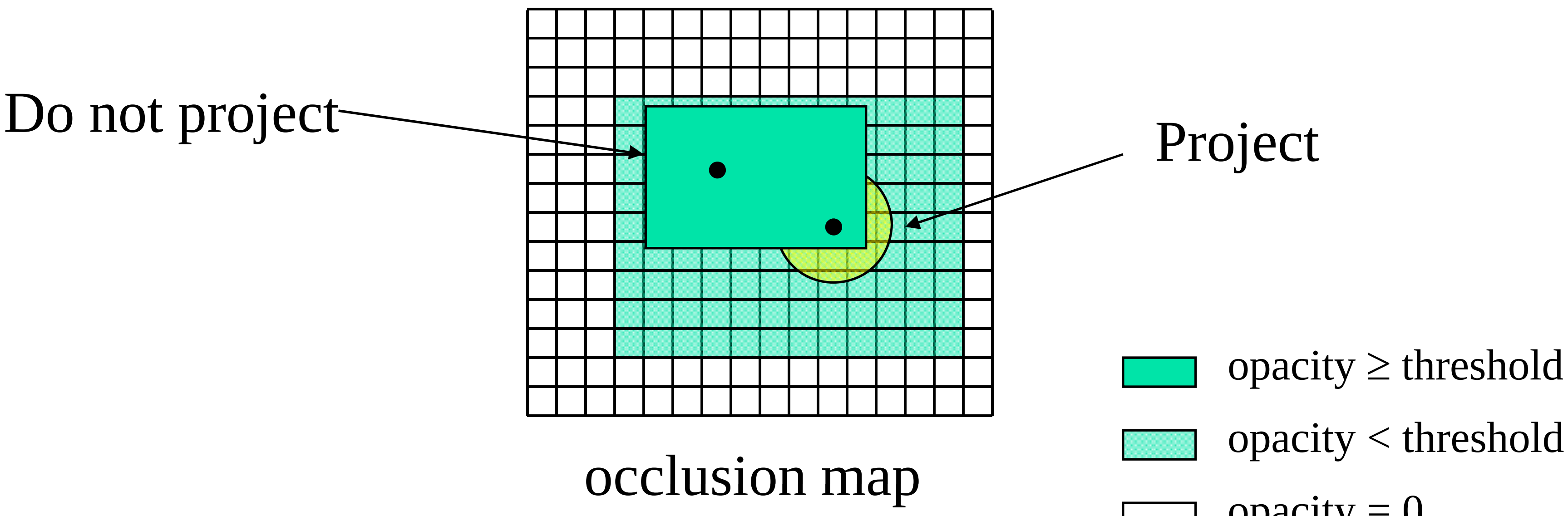
Occlusion Culling

- A voxel is only visible if the volume material in front is not opaque



Visibility Test Based on SAT of Occlusion Buffer

- Compute occlusion map after each sheet
- Cull both individual voxel and voxel sets with a summed area table of occlusion map



Occlusion Culling

- Build a summed area table (SAT) from the opacity buffer
- To test whether a rectangular region is opaque or not, check the four corners

$$(O_{ur} - O_{ul} - O_{lr} + O_{ll})$$

- Can cull voxel sets directly

Surface Rendering with Splatting

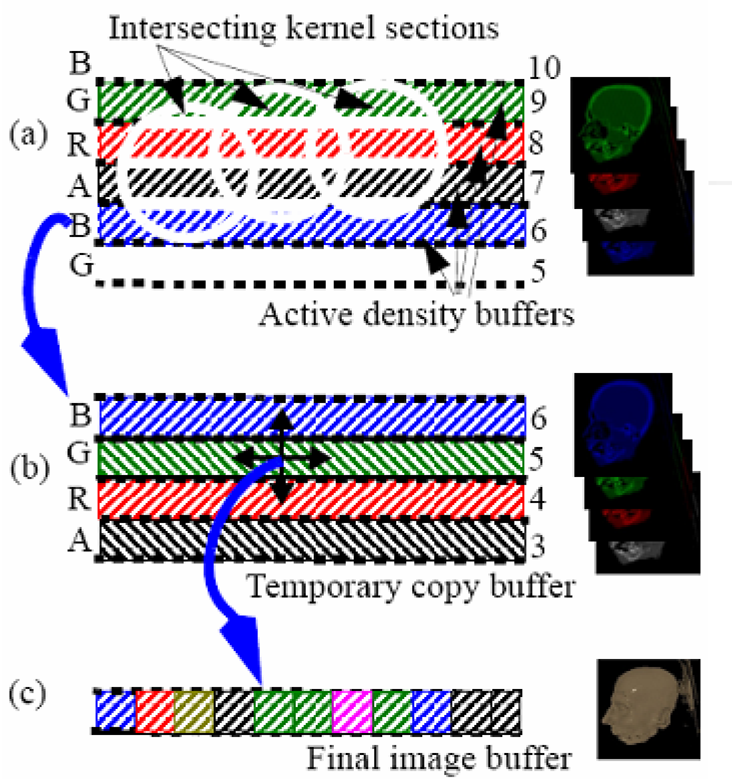
- Splatting can also be used in surface rendering
- The surface is represented with set of points on surface
- The right is taken from Stanford
- The project (called Qsplat) uses splatting to render scanned object
- The image shows Moses sculpture which was scanned as part of the Digital Michelangelo project

GPU Features for Splatting

- Vertex Arrays
  - OpenGL : DrawArrays / DrawElements
  - DirectX : DrawPrimitive / DrawIndexedPrimitive
- Point Sprites extension
  - Only one vertex is needed for each voxel
  - Nvidia boards after GeForce4
  - ATI Radeon 8500 and betters
- Early Z-rejection Test
  - Does depth test before the fragment is processed
- *N.Neophytou & K.Mueller , "GPU Accelerated Image Aligned Splatting"*

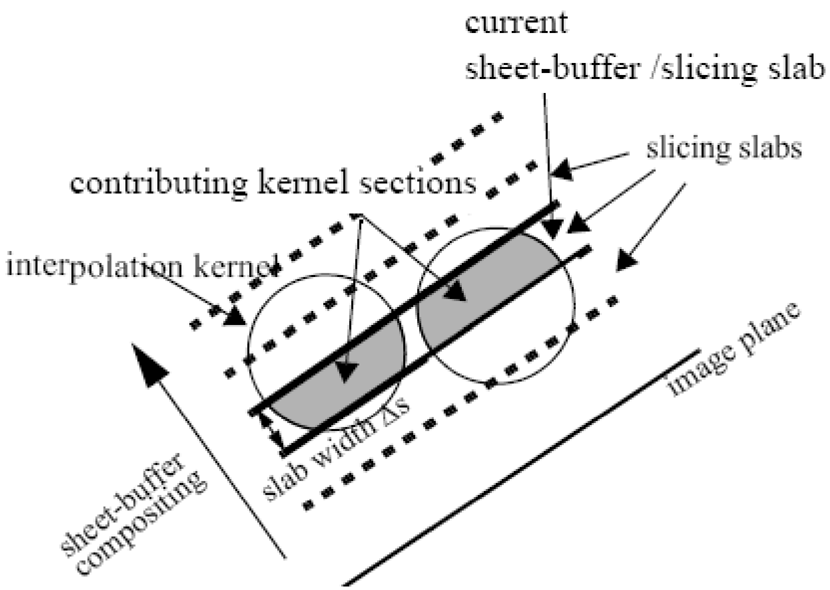
Overall Process

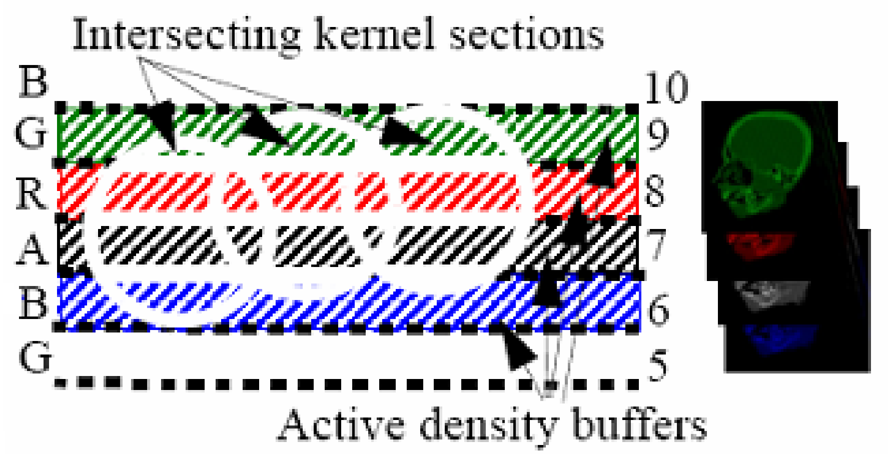
- Splatting Phase
- Copying Phase
- Compositing Phase



Splatting Phase

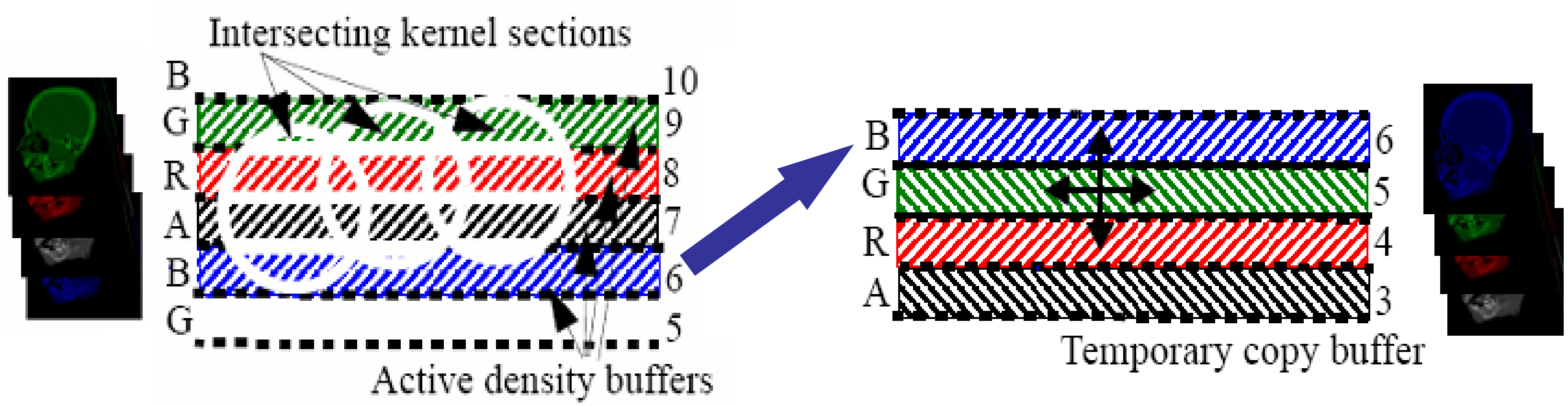
- Voxels are arranged into arrays according to the first image aligned slab that they intersect
- Every voxel is splatted into the active density buffers
- Use RGBA channel as four separate density slices
  - Assume each voxel to be rasterized four times
  - Post-shaded volume rendering





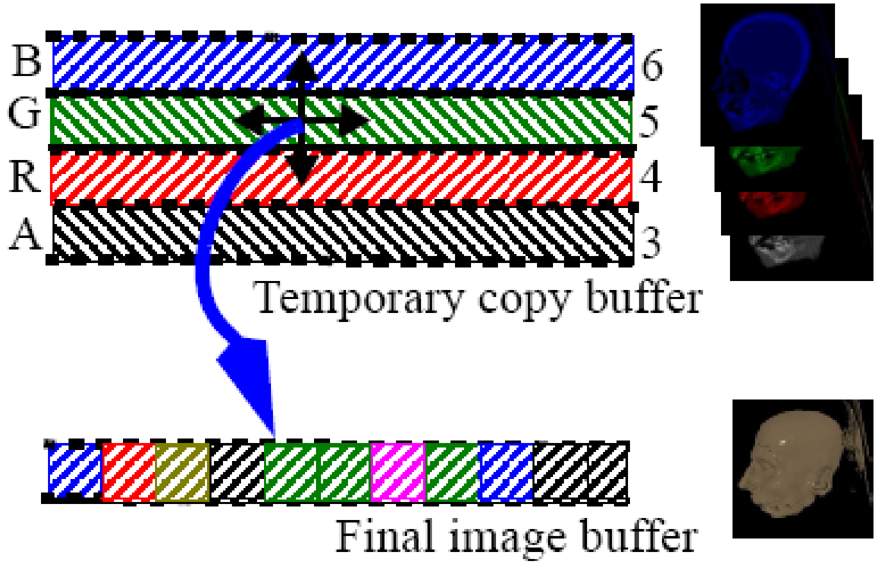
Copying Phase

- Completed slice is copied to the copy buffer
- Copy buffer holds the last four completed slices
- Gradients for the last slice but one are calculated on the fly, using its front and back sliced on the buffer



Compositing Phase

- Shaded result is composited to the final image buffer

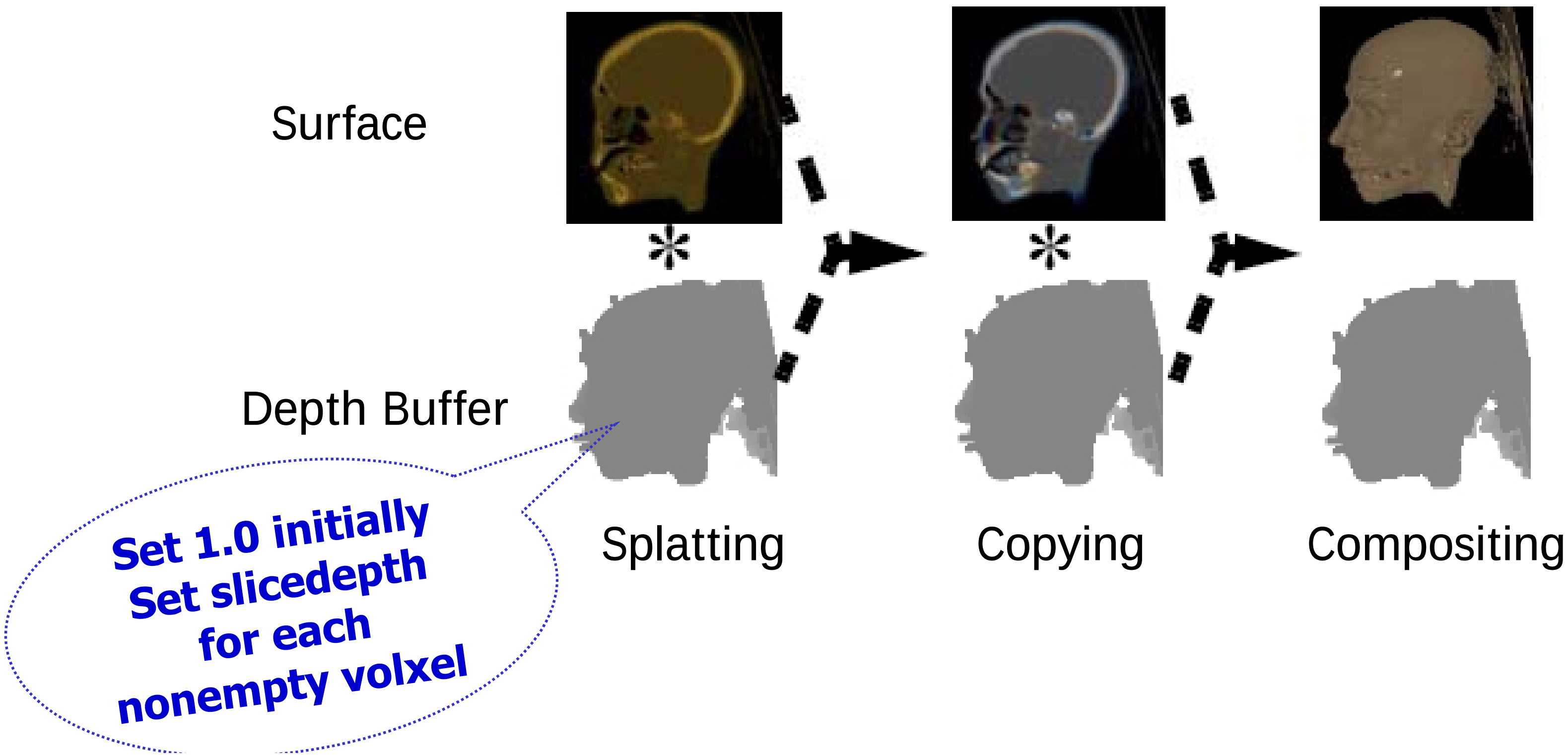


Avoiding Shading of Empty Regions

- Early Z-rejection is used
  - Prepare a depth buffer that all of the drawing surfaces share, and clear it to 1
  - Let sliceDepth(n)=(1023-n)/1024
  - Splat to current slices with depth writing turned on
  - Perform depth test with
    LESS OR EQUAL THAN sliceDepth(n)

Avoiding Shading of Empty Regions

- Only touched pixels is copied
- Only touched pixels are composited

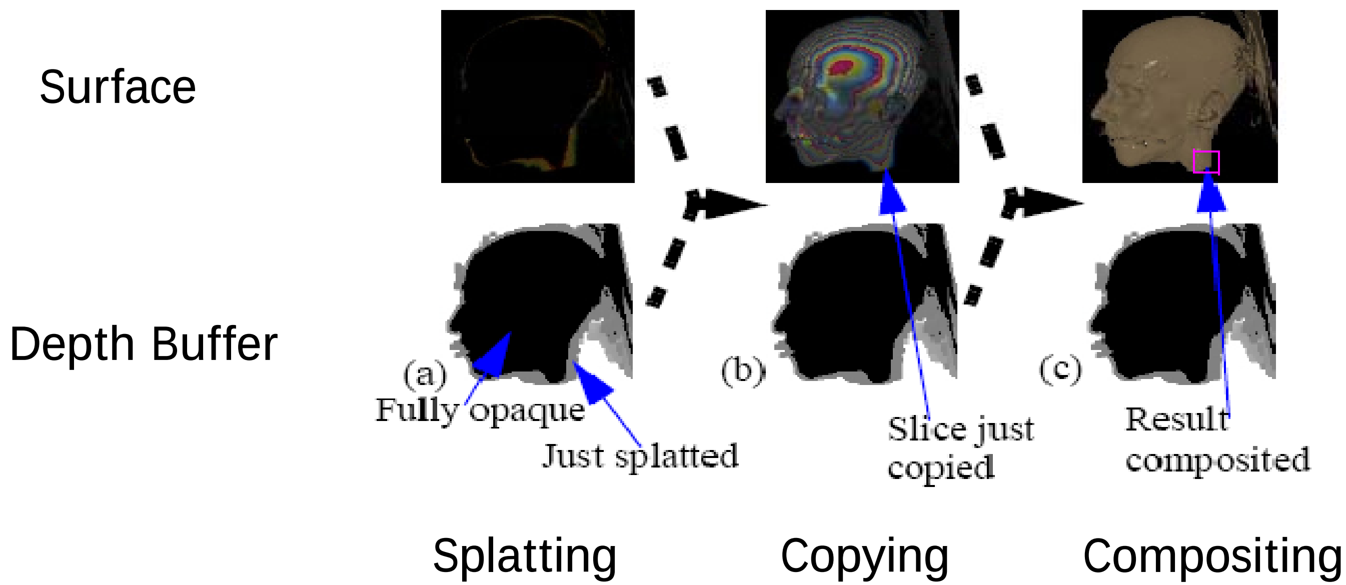


Skipping Opaque Regions

- Write opaque region data on the depth buffer
  - Read the image plane as a texture on compositing phase
  - Compare alpha value with predefined threshold
  - For opaque pixel, write 0 on the depth buffer
  - During splatting phase, perform depth test with NOT EQUAL TO 0

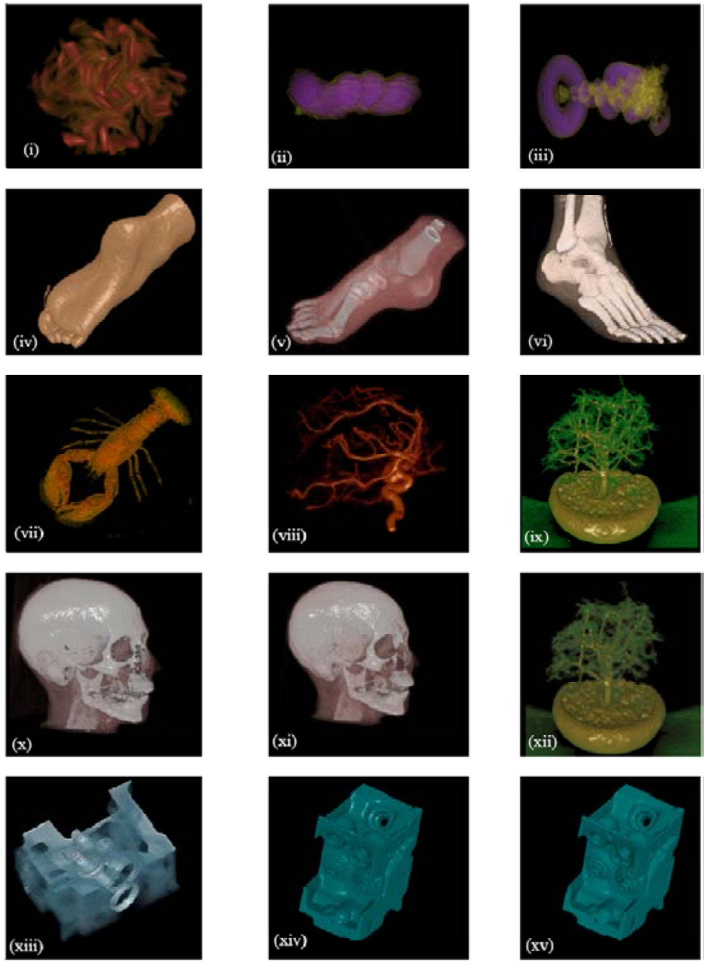
Skipping Opaque Regions

- All the pixels with depth 0 will be excluded even from the splatting phase
- Copying and compositing phase ignores them by depth bound test or alternative process



Results

Data set	Size	Effective Splats	FPS	Fig.6
Vortex	128^3	479K	5.2	i
Jet simul.	256^3	648K	4.0	ii
Turbulant	$104x129^2$	95K	6.1	iii
Foot Isosurf.	128^3	191K	7.2	iv
Foot semiTran.	128^3	184K	6.4	v
Foot semi-2	128^3	181K	7.6	vi
Lobster	320^2 x 34	219K	10.2	vii
Aneurism	128^3	17K	9.1	viii
Bonsai	256^3	1.3M	4.9	ix
BonsaiBCC	181^2x362	955K	2.5	xii
CT Head semi	128^3	526K	4.9	x
CT Head BCC	91^2x182	379K	3.1	xi
Engine Semi	256^2x128	1.2M	2.1	xiii
Engine ISO	256^2x128	1.3M	5.1	xiv
Engine BCC	181^2x182	963K	5.3	xv



Really use GPU for IASB!!!

- Good for only for small volumes
  - Iso-Surface rendering
- Capability of vertex shader (whether it can holds whole voxel list or not)
  - Each vertex needs at least 12byes for (x,y,z) and 2 Bytes for density.
- Calculation of distance between the voxel and view-aligned sheet plane
  - Preprocessing and view dependent
- Making View-aligned Voxel List
  - How to sort? 🡐 Heavy computation
  - Reordering must be performed whenever viewpoint changes