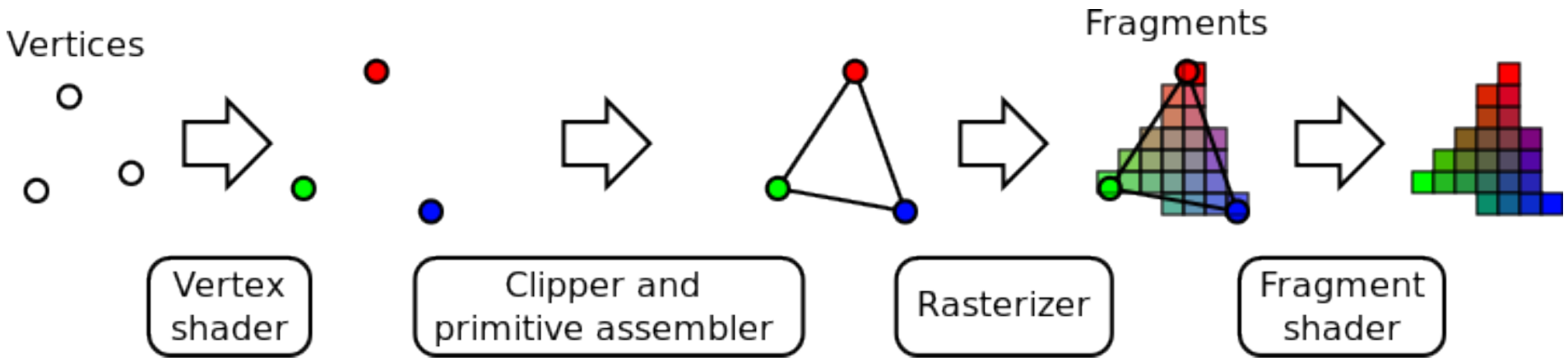
Rendering techniques for visualisation

Scientific Visualisation
HT 2018
Lecture 5

Fredrik Nysjö
Centre for Image analysis
Uppsala University

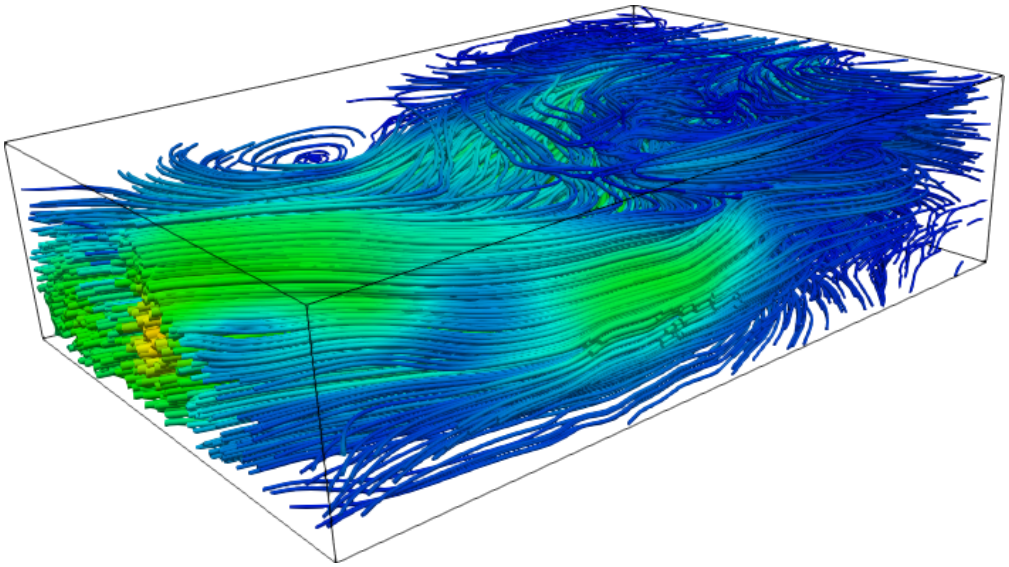
First: A trip through the programmable graphics pipeline

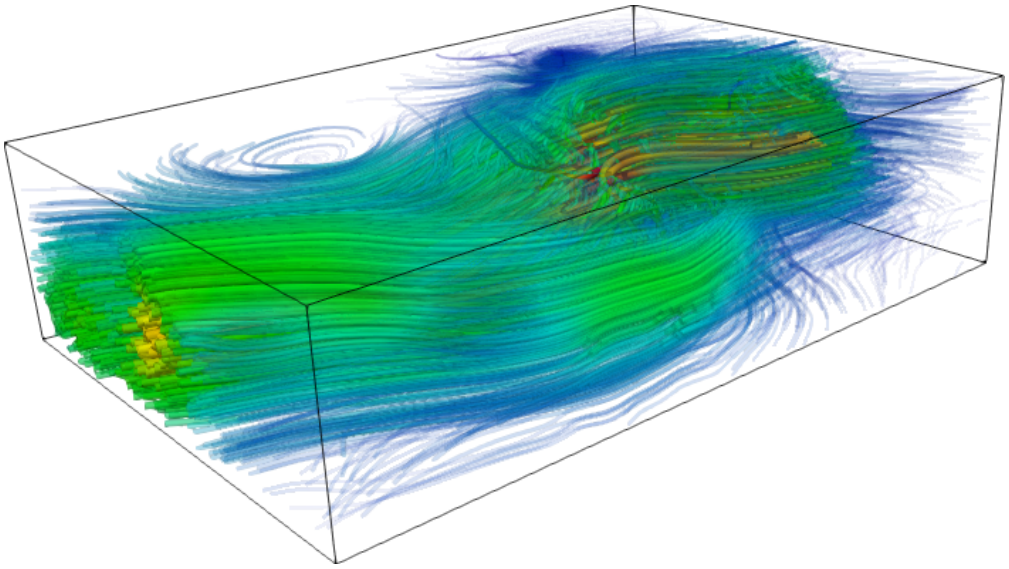


Transparency

Transparency

- Many uses in visualization:
  - Visualizing data with different layers
  - Showing hidden structures and reducing clutter
  - Volume rendering
  - Overlays and user-interfaces (UI)





Alpha blending

- Composites (blends) the values of two fragments
- The OVER operator [Porter and Duff '84] used for back-to-front alpha blending:

$$dst' = s_f * src + (1 - s_f) * dst$$

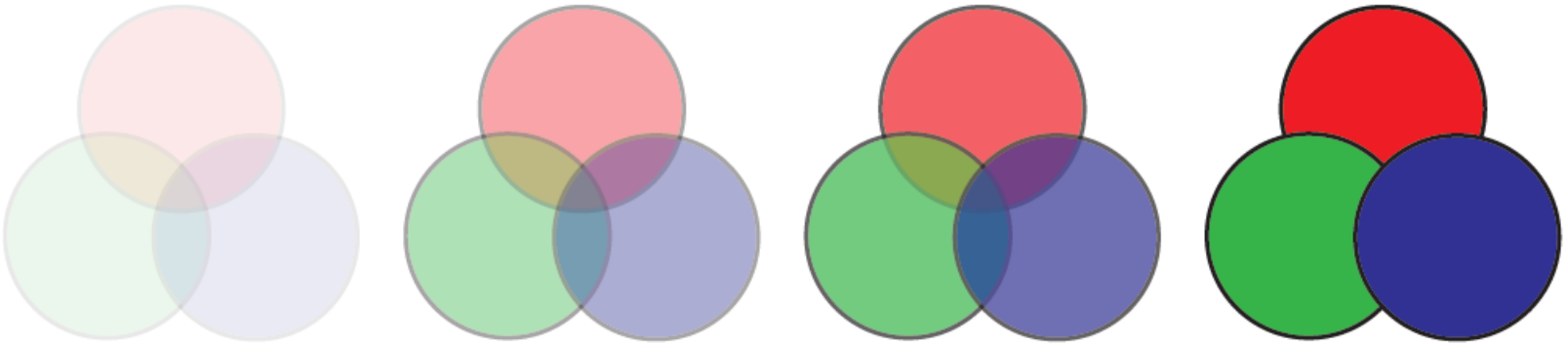
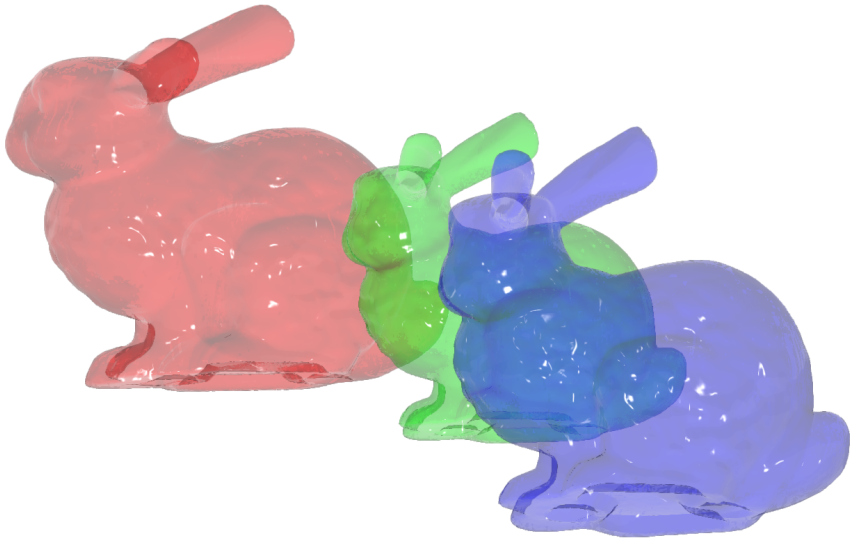


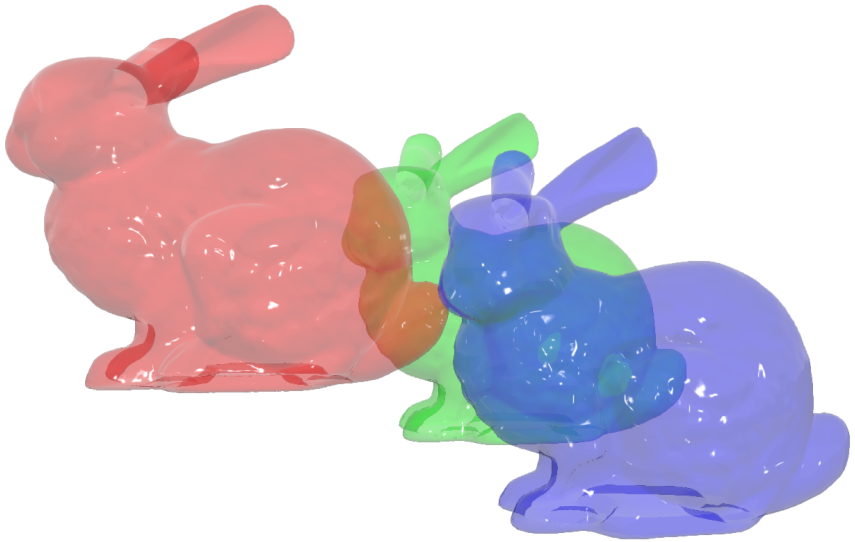
Image source: http://www.realtimerendering.com

Order-dependent transparency

- The OVER operator is not commutative!
- Thus, requires depth sorting (~O(n log n)):
  1. Sort primitives (triangles) by distance from camera
  2. Render back-to-front (Painter's algorithm)



Unsorted (incorrect) transparency



Sorted (correct) transparency

Image courtesy: Johan Nysjö

Painter's algorithm

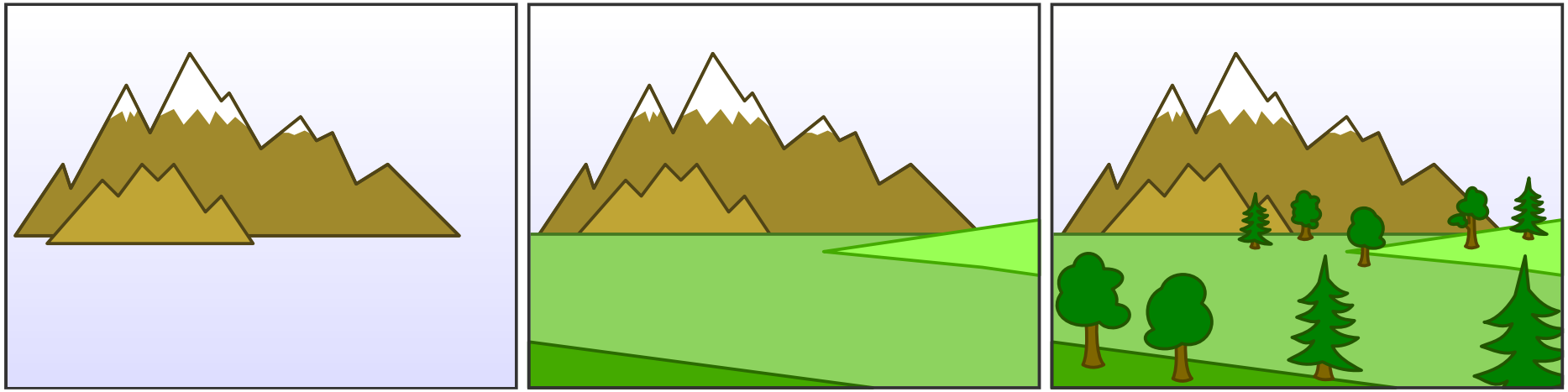
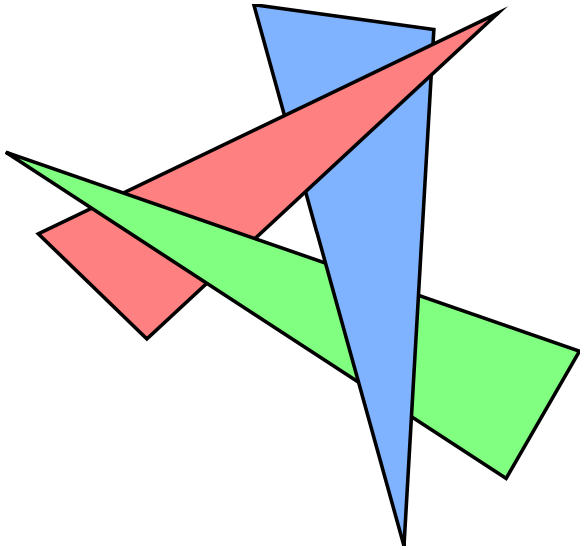


Image source: https://en.wikipedia.org/wiki/Painter's_algorithm

Painter's algorithm: Failure case



Order-independent transparency

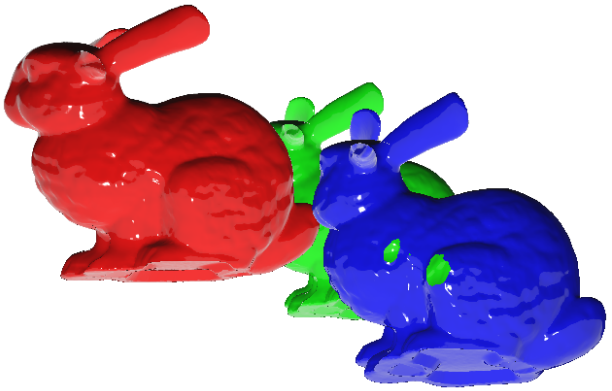
- Order-independent transparency (OIT) methods does not require primitive sorting
- Stores fragments for sorting afterwards (A-buffer) or uses commutative blend equations (blended OIT)
- Other techniques: Depth-peeling (available in VTK)

Weighted blended OIT

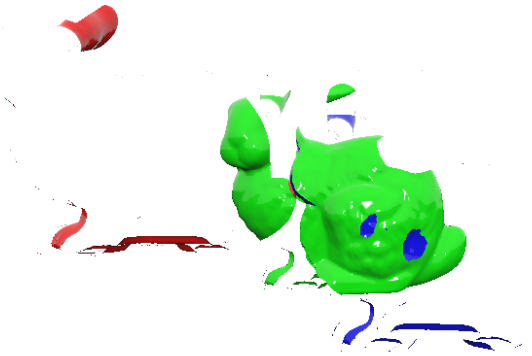
Image source: https://developer.nvidia.com

Depth-peeling (basic idea)

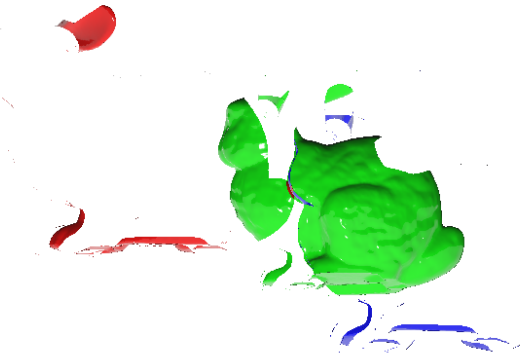
Peeled layer 1



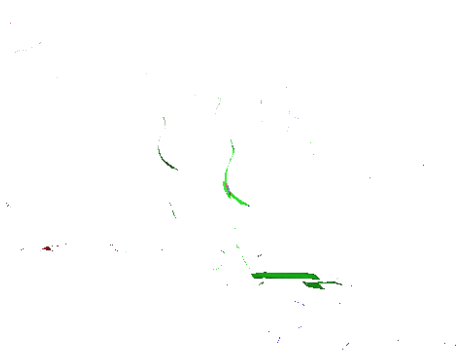
Peeled layer 2



Peeled layer 3



Peeled layer 4



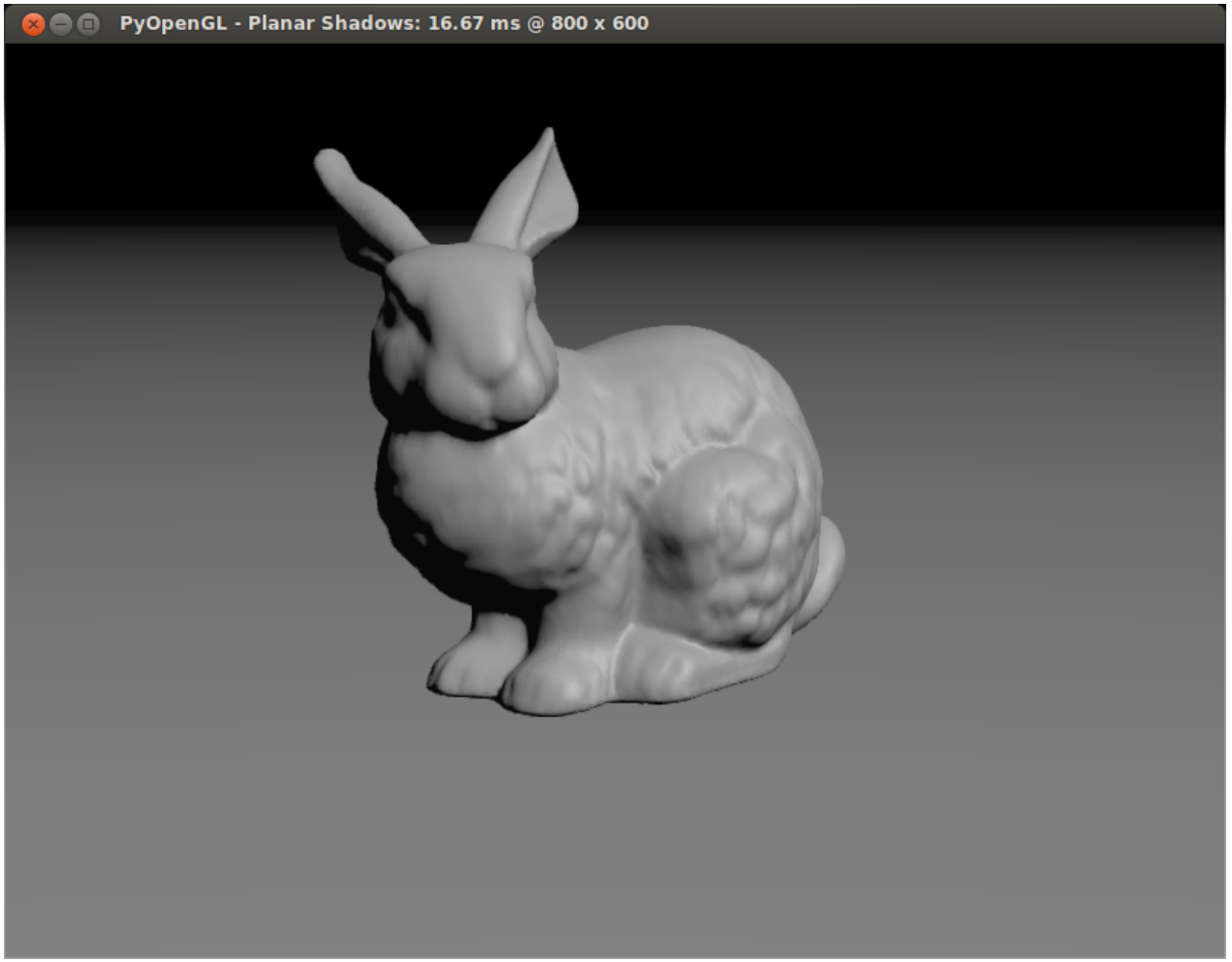
Peeled layer 5

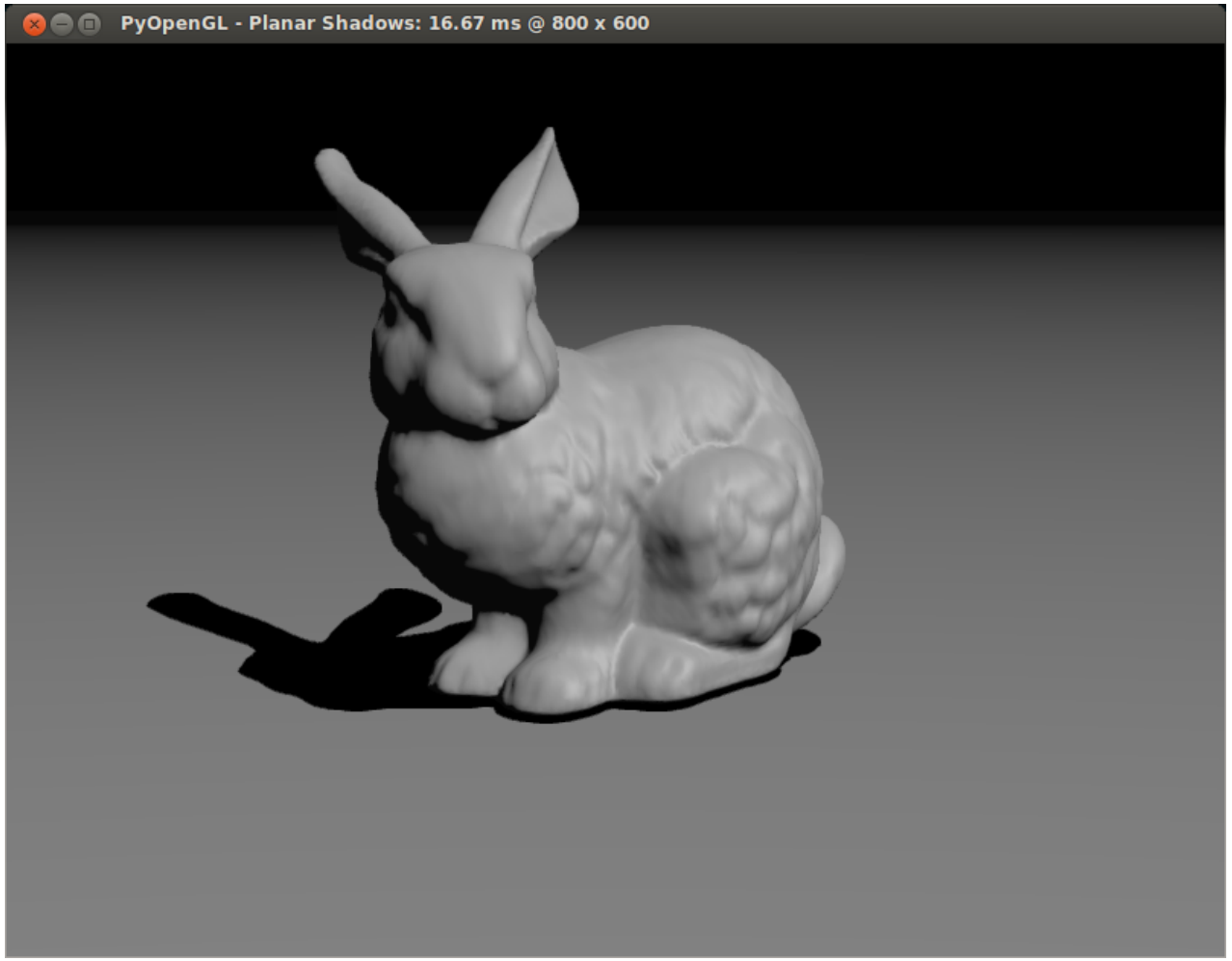
Front-to-back
blended result

Image courtesy: Johan Nysjö

Shadows and Ambient Occlusion

Is the bunny standing on the ground or floating in the air?



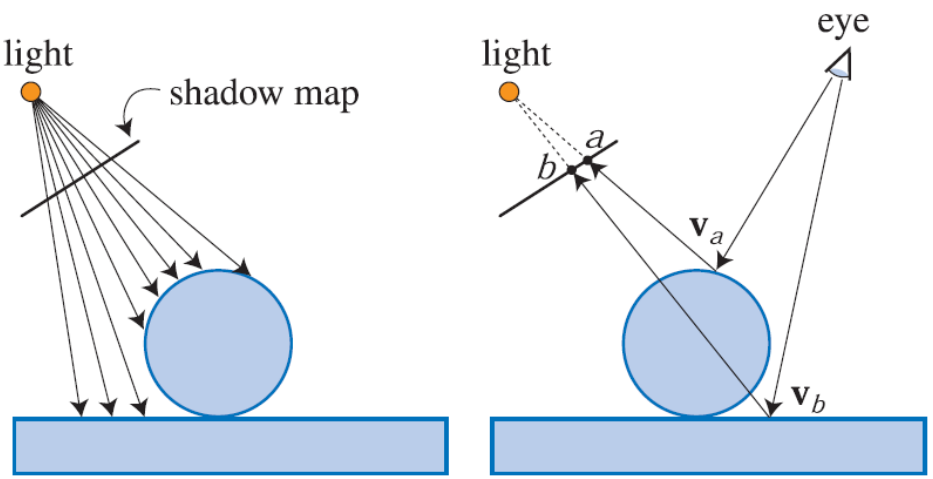

PyOpenGL - Planar Shadows: 16.67 ms @ 800 x 600

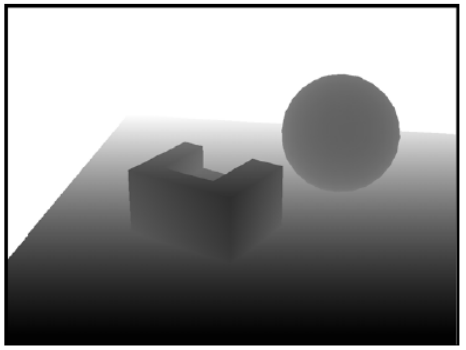
Shadows

- Important depth cue: helps us perceive depth and spatial relations between 3D objects
- Common techniques:
  - **projective shadows**
  - **shadow mapping** (lots of variations)
  - **shadow volumes**
- Trade-off between speed and quality

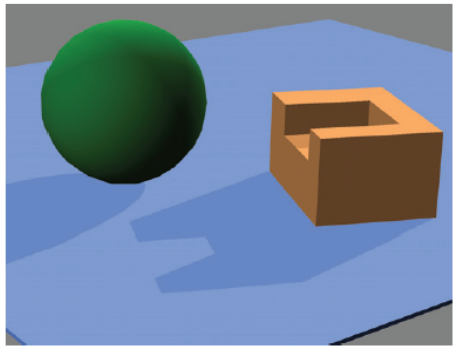
Shadow mapping

- Basic idea:
  1. Render the scene depth values (from the light's perspective) into a texture to generate a **shadow map**
  2. Render the scene again (color and depth values) from the viewer's perspective and compare the location of each fragment with the shadow map to determine if the point is in shadow





Shadow map

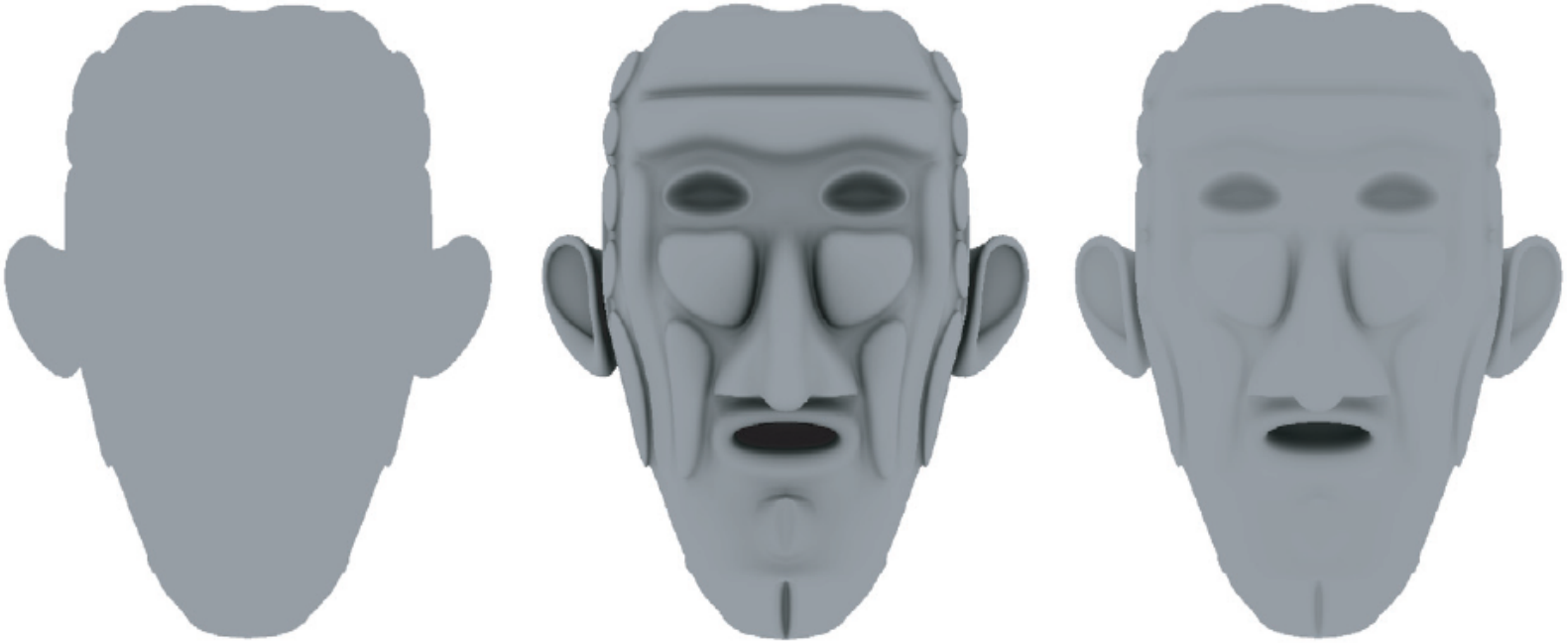


Same scene rendered from the viewers perspective (with the shadow map applied)

Image courtesy: http://www.realtimerendering.com

Ambient occlusion

- Simulates self-shadowing and shadowing of ambient light
- Surface points that are occluded becomes darker



No shadows | With ambient occlusion | Full global illumination

Image courtesy: http://www.realtimerendering.com

Screen-space ambient occlusion

- Render scene depth and normals to textures and compute dynamic ambient occlusion in a post-processing pass
- Most techniques uses a rotating filter mask with uniformly distributed points to sample the depth and normal textures
- A subsequent blurring pass is required to remove noise

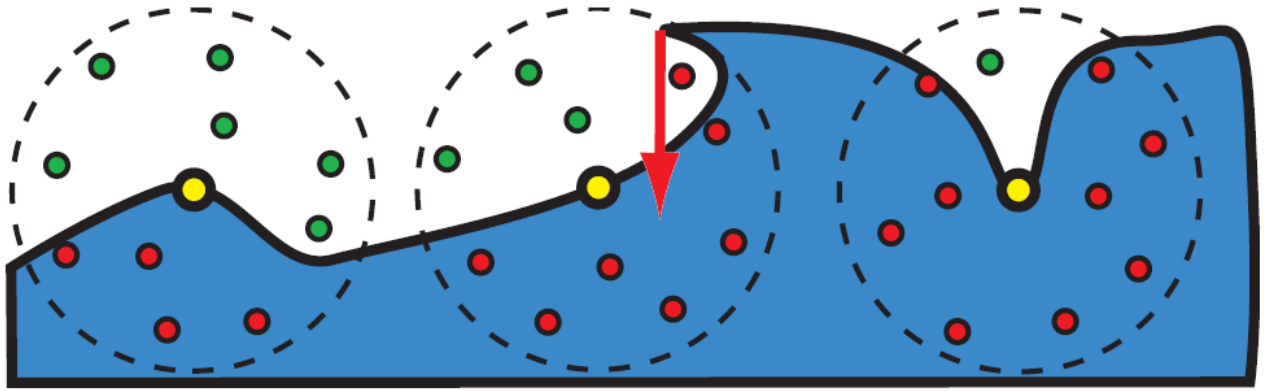


Image courtesy: http://www.realtimerendering.com

Screen-space ambient occlusion

Ambient lighting

*

Occlusion

=

Final shading

Volume Rendering

Volume rendering applications

- Medical visualization
- Scientific visualization
- Computer games and visual effects
  - Clouds
  - Fire
  - Smoke
  - Volumetric fog

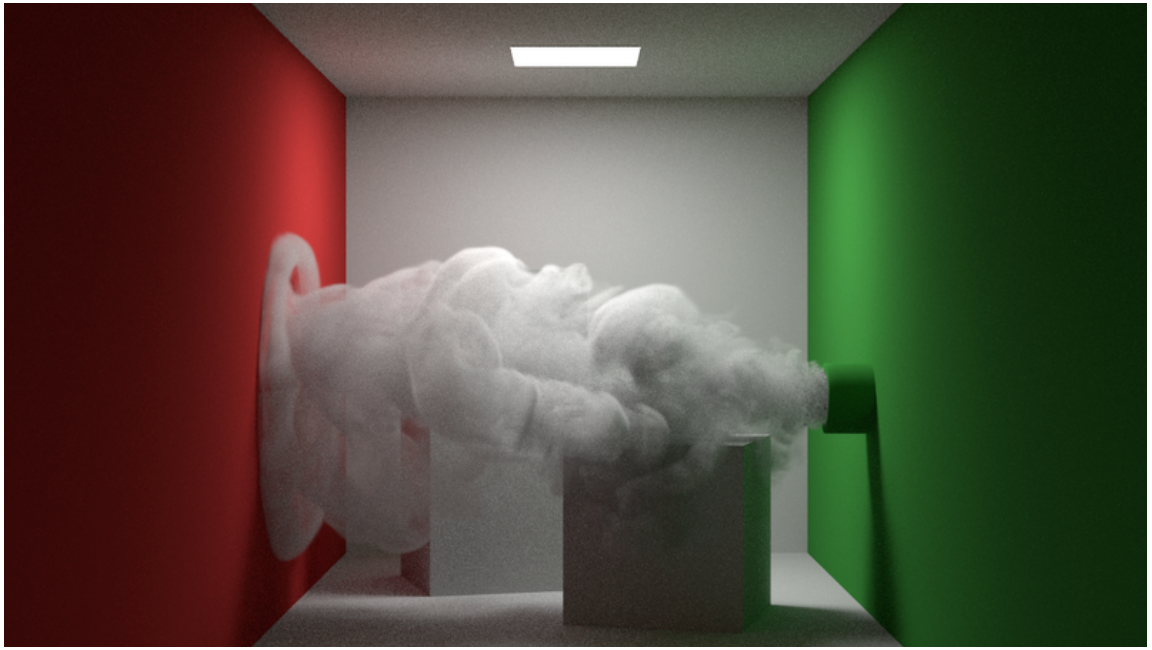
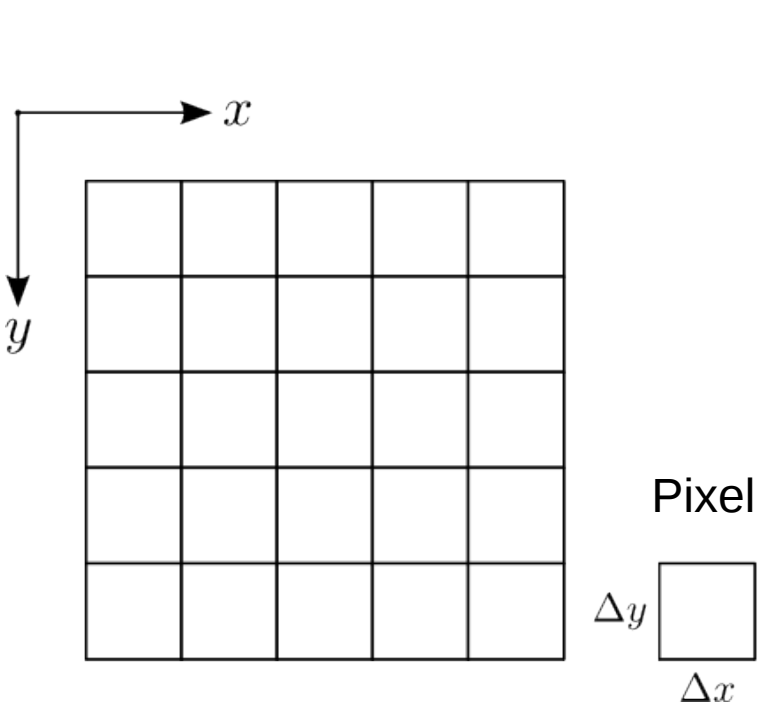
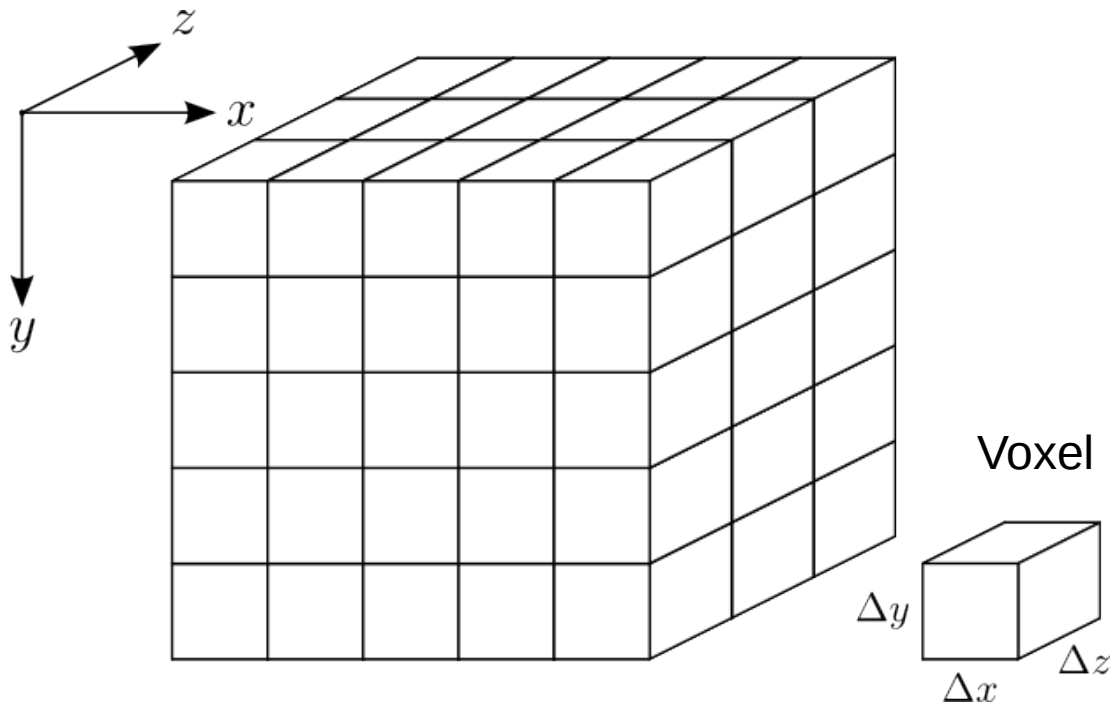


Image source: http://www.blender.org

Digital images

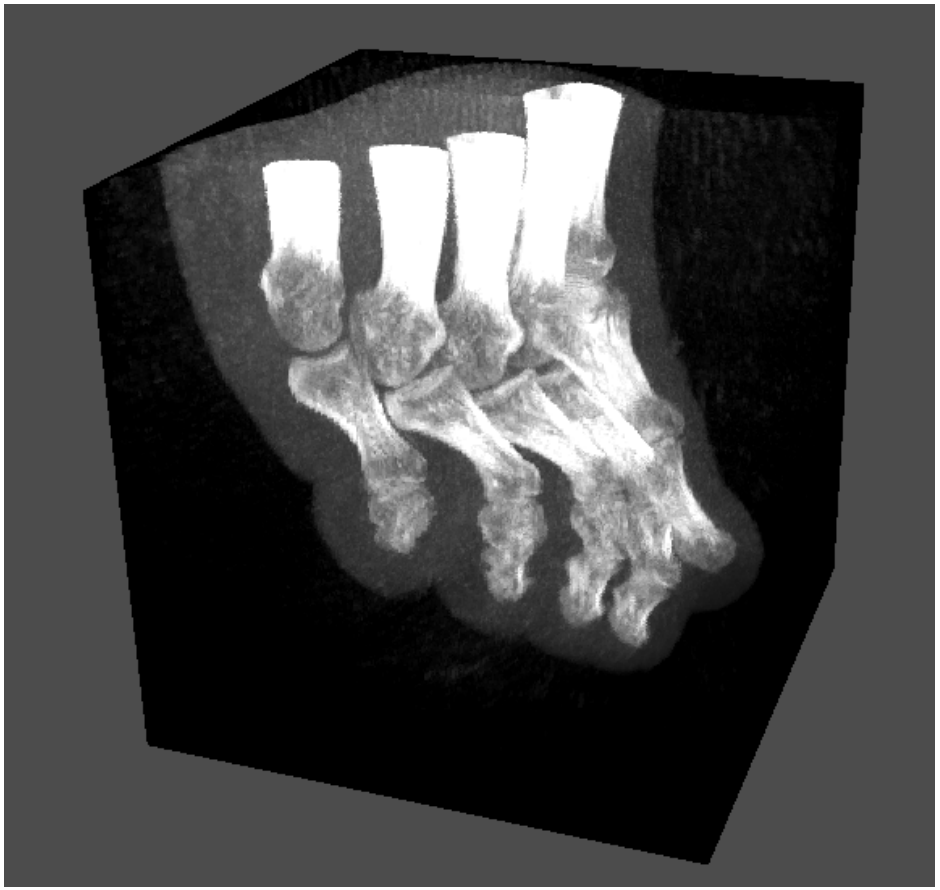


2D image



3D (volume) image

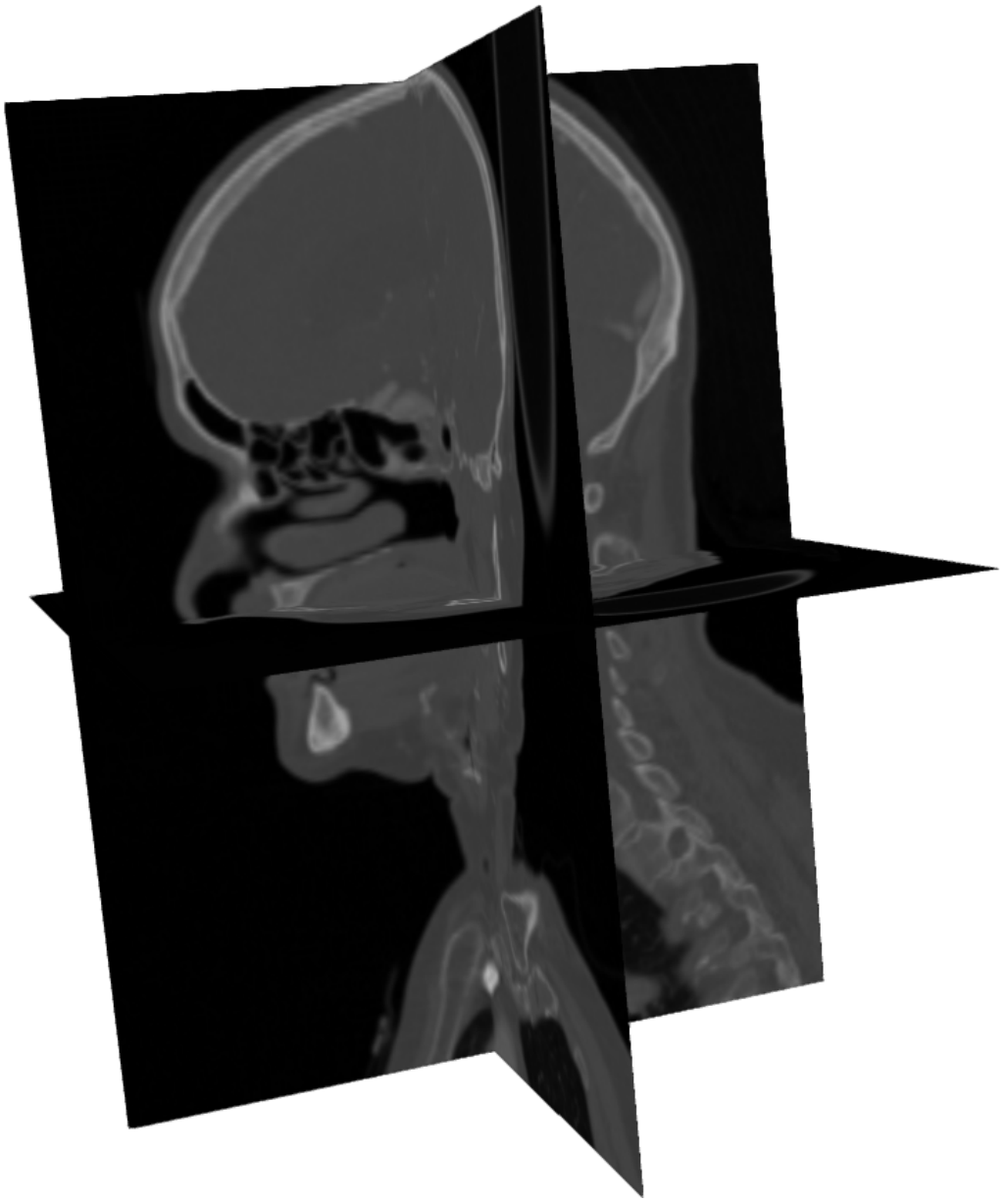
Volume data



Computed tomography (CT) image of a foot. The intensity values of the voxels represent different tissue types (bone, soft tissue, skin, etc)

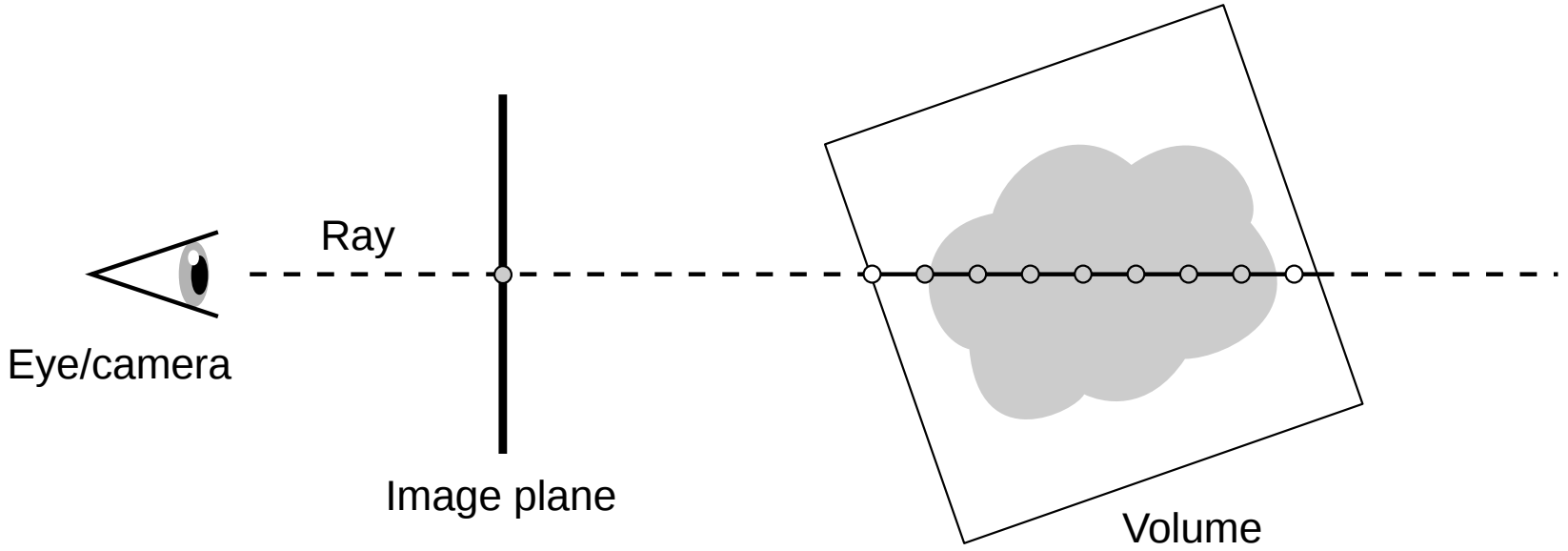
- Represented as regular 3D grids with scalar or vector values
- Can be aquired with, e.g., a CT scanner or be generated procedurally
- Voxels can be anisotropic (have non-uniform size)

Multi-planar reformatting (2D slices)

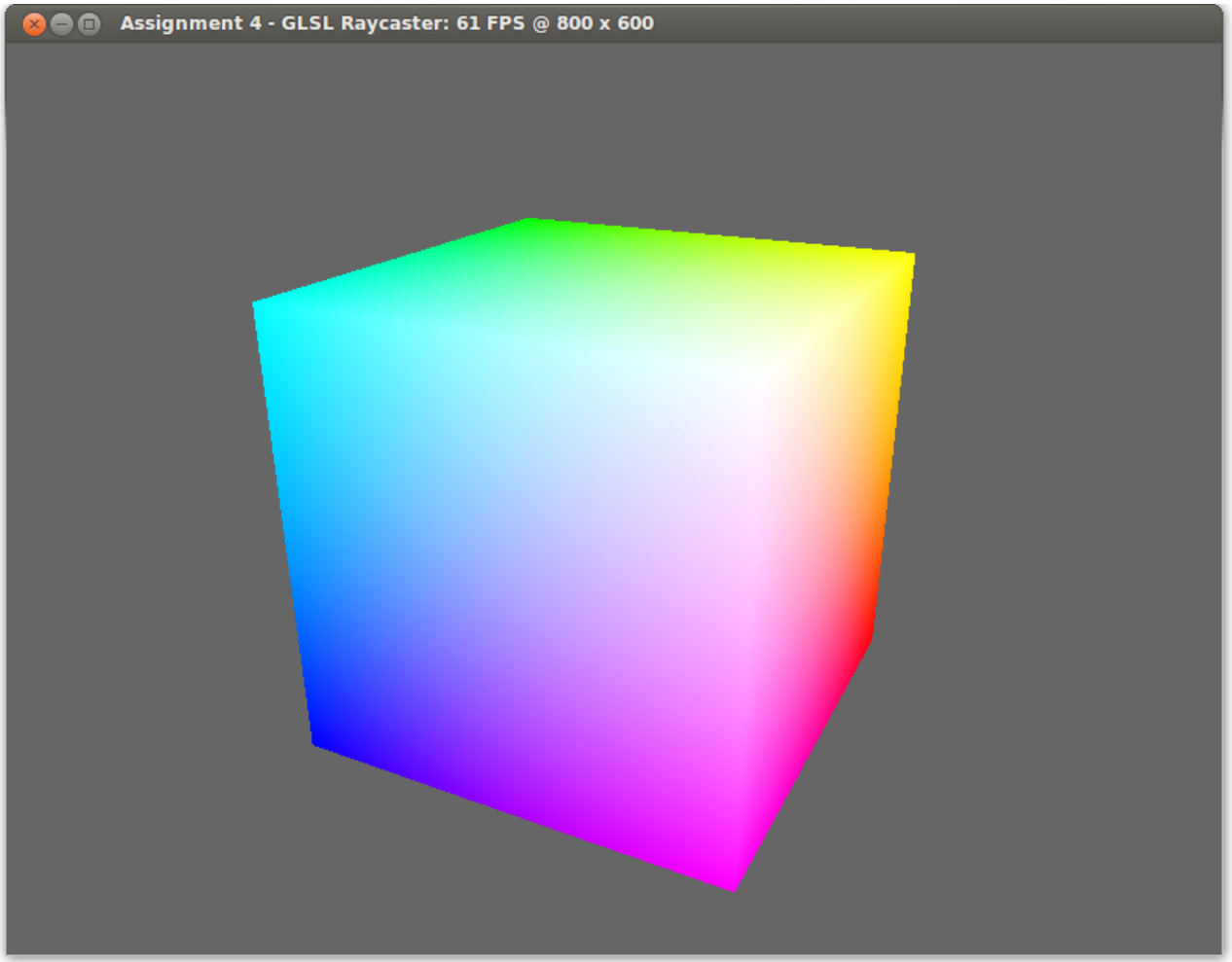


Ray-casting

- For each fragment (i.e., each pixel in the viewport), cast a ray from the starting point (front face) and sample the volume along the ray direction at even intervals until the ray reaches the end point (back face)



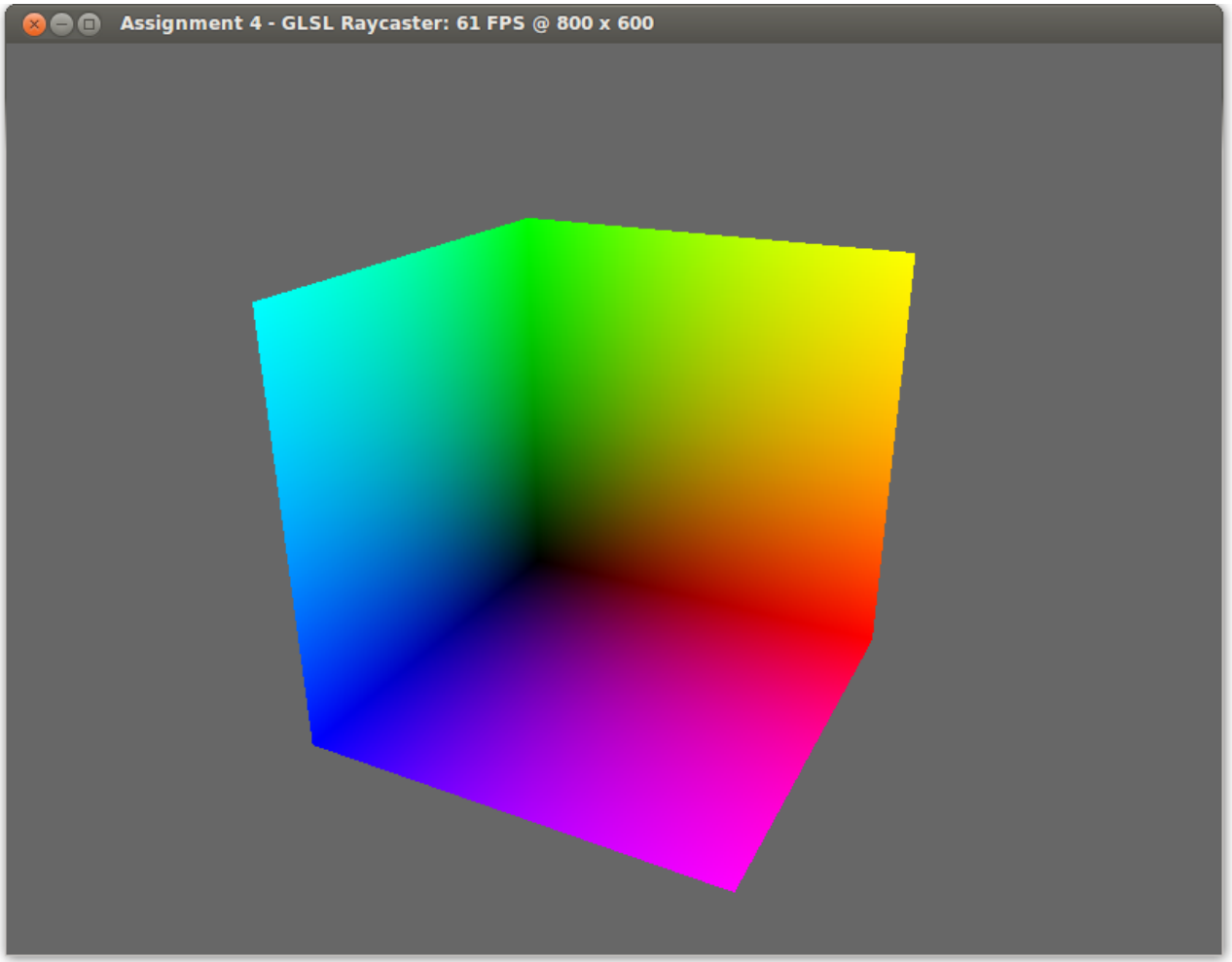
Front-face image (starting points)



Ray-start positions* displayed as RGB colors

*Specified as 3D texture coordinates (s,t,p) ranging from 0.0 to 1.0

Back-face image (end points)



Ray-end positions* displayed as RGB colors

*Specified as 3D texture coordinates (s,t,p) ranging from 0.0 to 1.0

GPU-accelerated ray-casting

- Basic algorithm:
  1. Render the **front face** of the volume image's bounding box to a 2D RGB texture to obtain **ray starting points**
  2. Render the **back face** of the bounding box to a 2D RGB texture to obtain **ray end points**
  3. Render a fullscreen quad and (in the ray-casting fragment shader) subtract the back-face texture from the front-face texture to obtain **ray direction**
  4. Given the ray starting points and direction vectors, **cast a ray** from each fragment into the volume image
  5. Let the ray gather color and transparency information from the voxels it passes through

Maximum intensity projection (MIP)

- Basic idea: extract the maximum intensity value along each ray to create an X-ray-like projection of the data

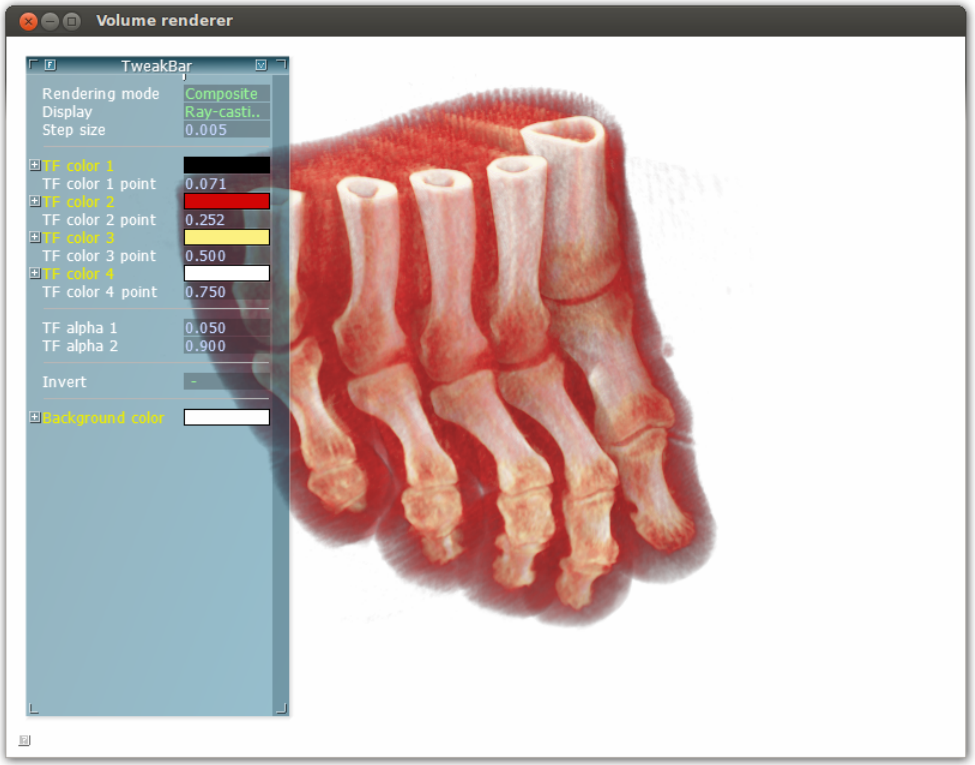
Isosurface rendering

- Stop raycasting at first opacity value above threshold, and compute surface normal and other attributes for shading
- With direct volume rendering, the isovalue can be updated interactively

Front-to-back alpha blending

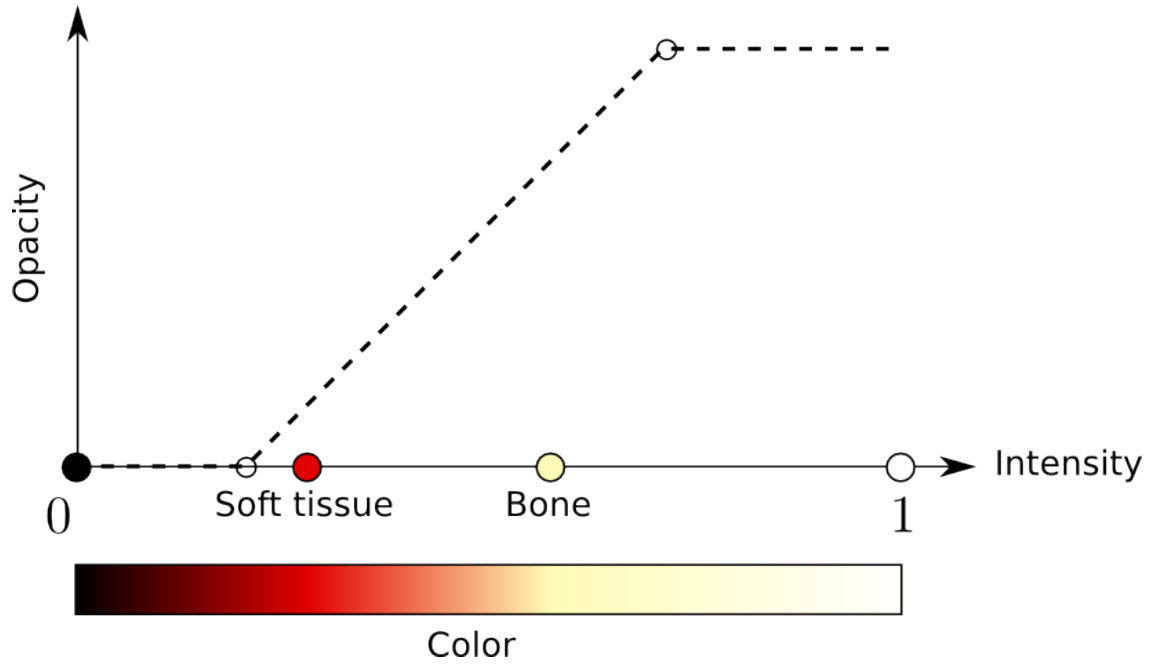
Front-to-back alpha blending

- Define a **transfer function** (TF) that maps voxel intensity to color and opacity
- Create a semi-transparent projection of the volume by accumulating opacity values along each ray while composing (blending) colors

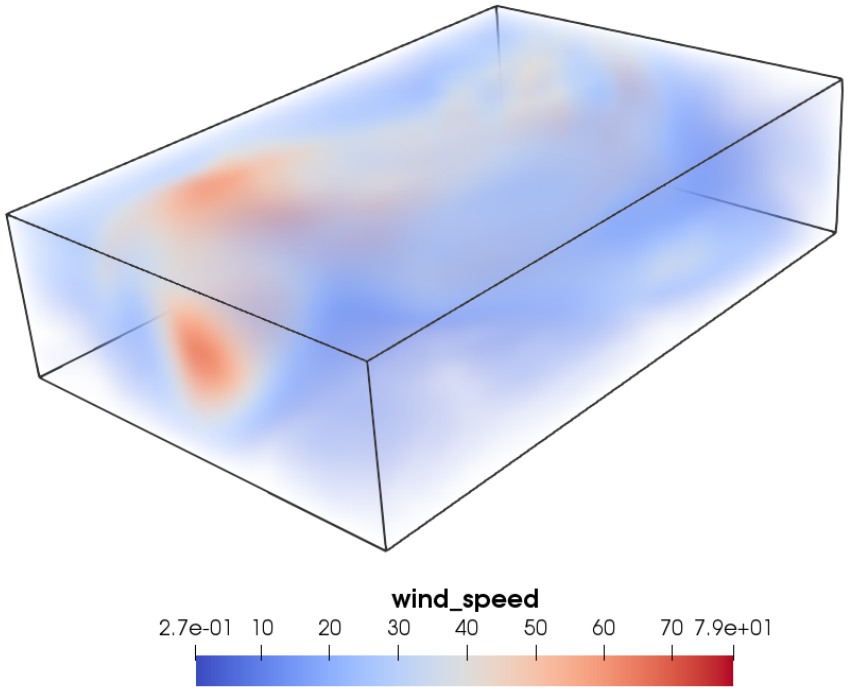


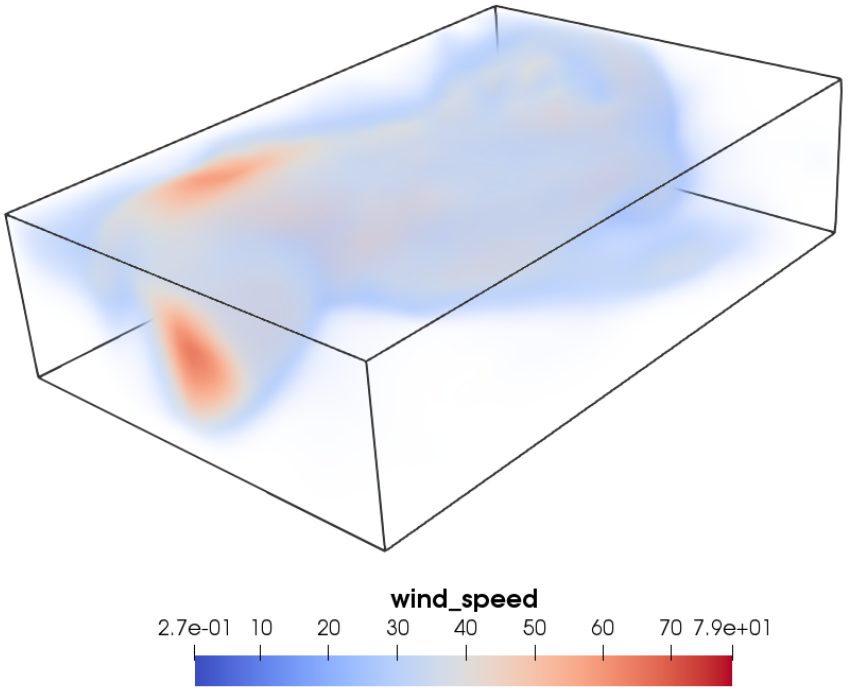
TF example (Human CT data)

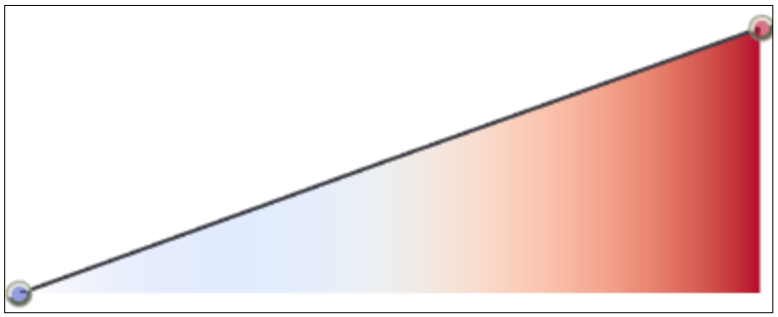
- Bone tissue (which has high intensity) will become white/yellowish and less transparent
- Soft tissue (which has low intensity) will become red/orange and more transparent

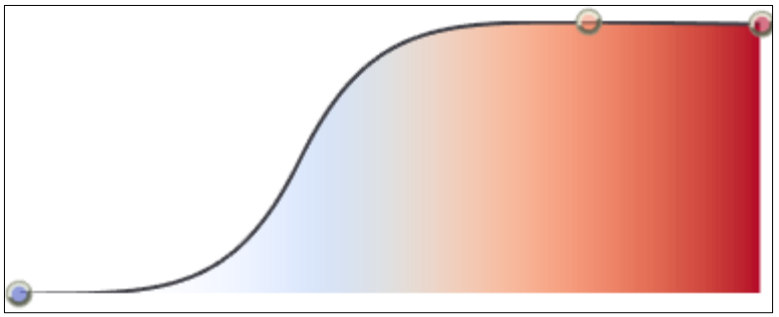


Other TF examples (Wind speed)

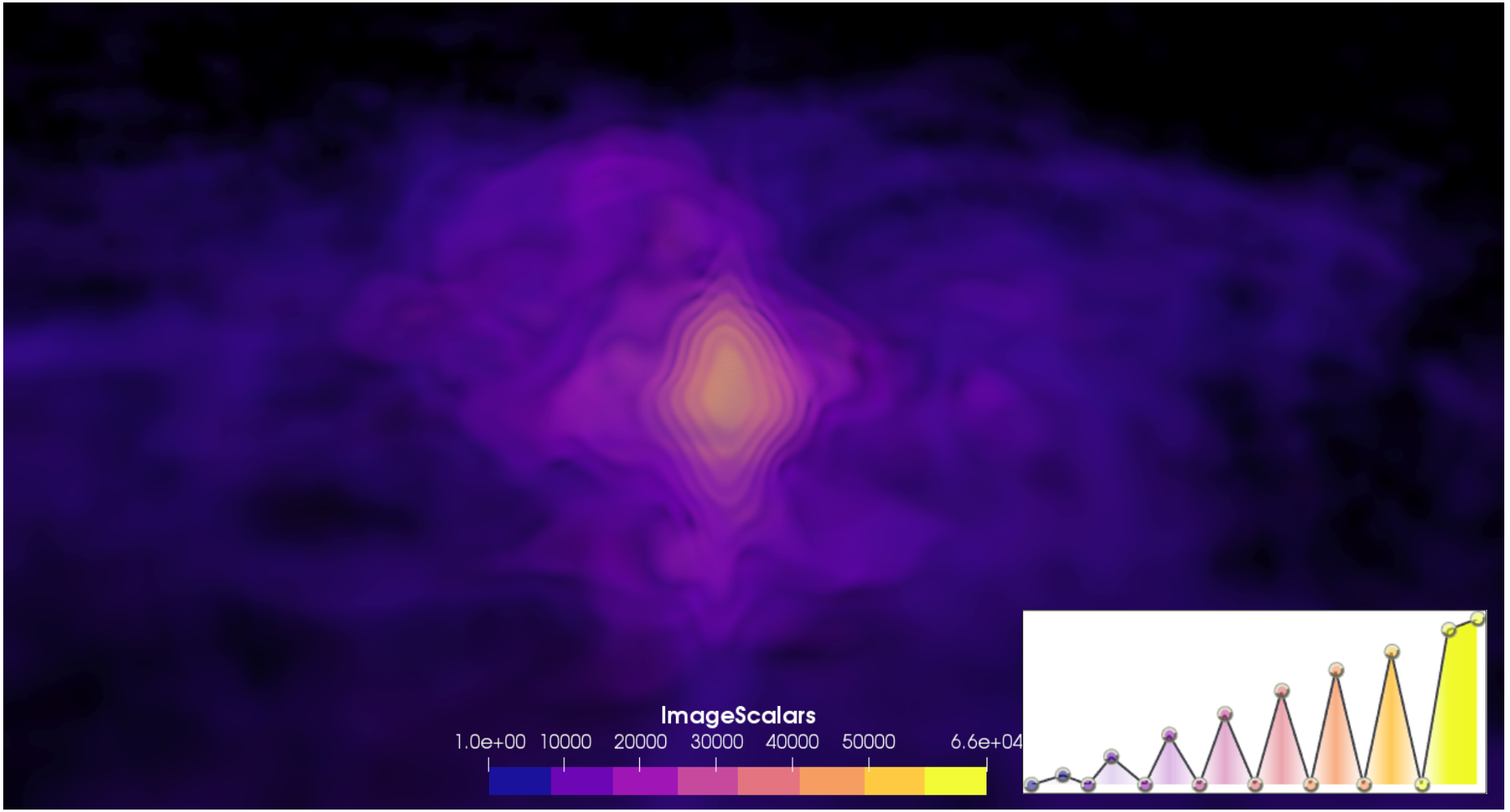








Other TF examples (Solar dust)



Acceleration methods: Empty-space skipping

- Improves performance by rendering a more fine-graind bounding geometry (from min-max value blocks)

Front faces

Back faces

Direct volume rendering vs. geometry-based rendering

- Direct volume rendering
  - **+** Allows showing other aspects of the data (MIP, transparency)
  - **+** Isovalue and other parameters can be changed interactively. Easy to add cut planes!
  - **–** Slow (without proper acceleration data structures)

- Geometry-based rendering
  - **+** Fast (uses the standard GPU rasterization pipeline)
  - **+** High data reduction (but information can be lost)
  - **–** Changing isovalue requires recomputing the whole geometry

Geometry-based rendering: Polygonisation

- Examples: Marching cubes, dual contouring, ...
- Anders covered this topic in lecture 3

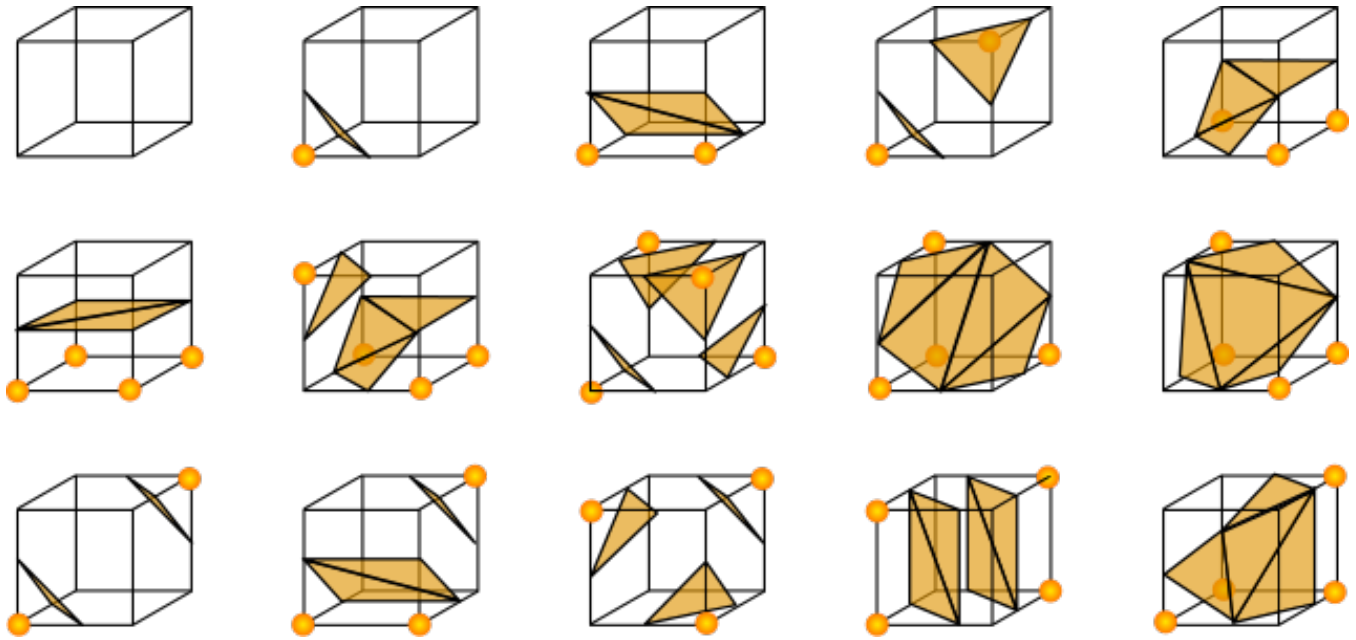


Image source: Wikipedia

Geometry-based rendering: Splatting

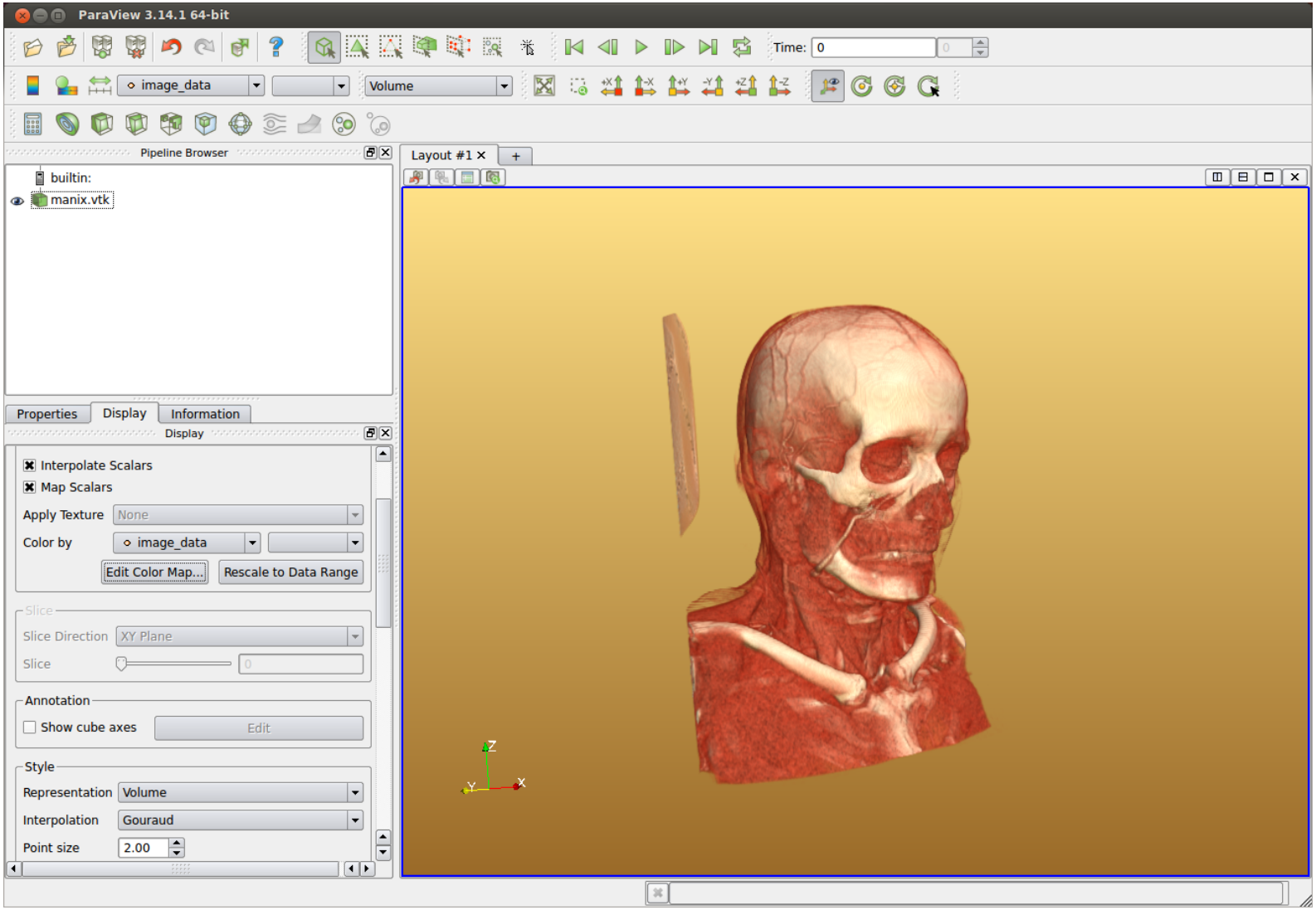
Image source: http://www.realtimerendering.com

Splatting

- Basic idea: for each boundary voxel (or grid cell with an intersection), emit a point and throw (”splat”) it onto the image plane
- Convolution is used to fill in holes
- Points can be extracted in a pre-processing step (similar to Marching Cubes)
- Efficient implementations exist for both the CPU and the GPU

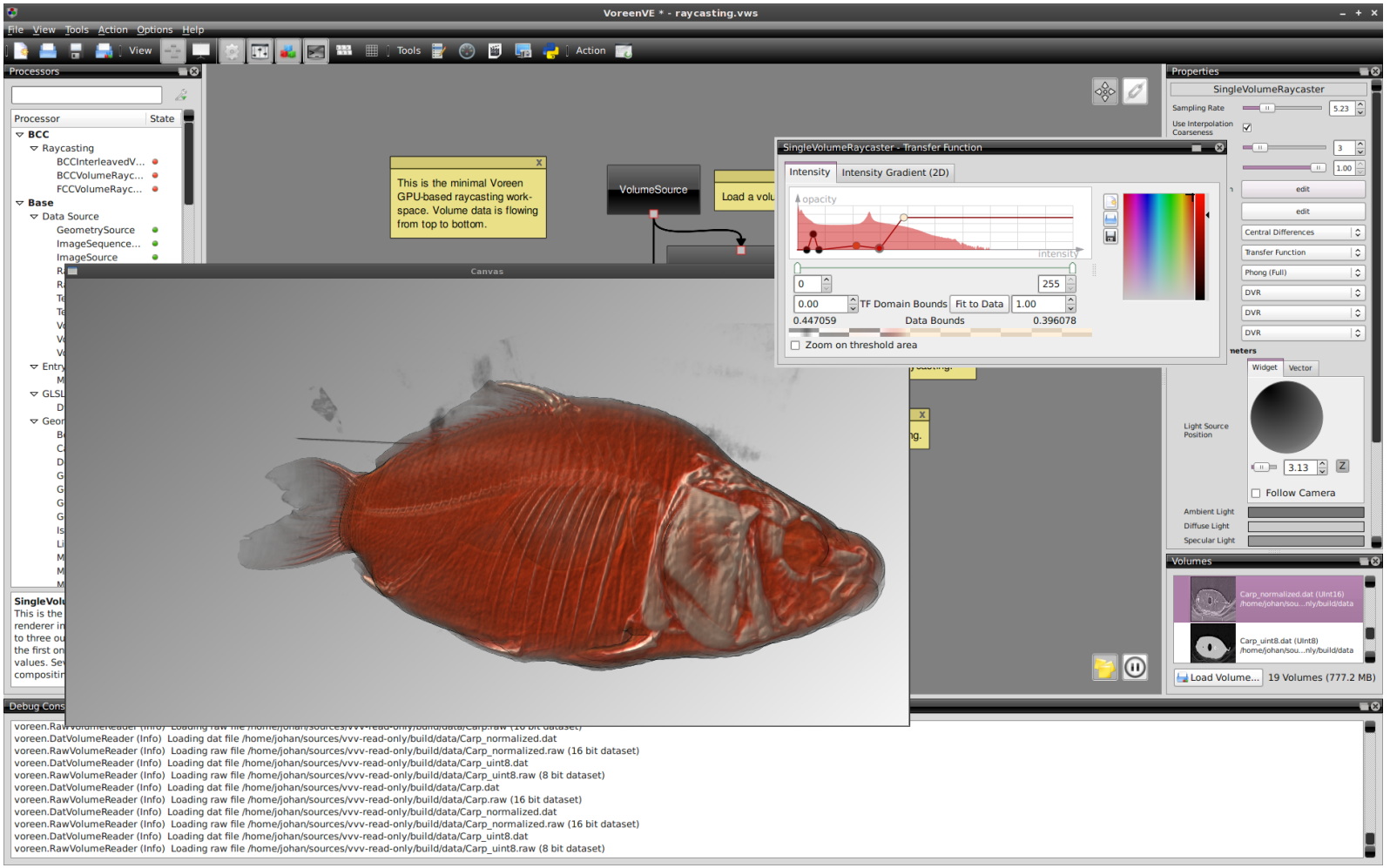
Image source: http://www.realtimerendering.com

Volume rendering software



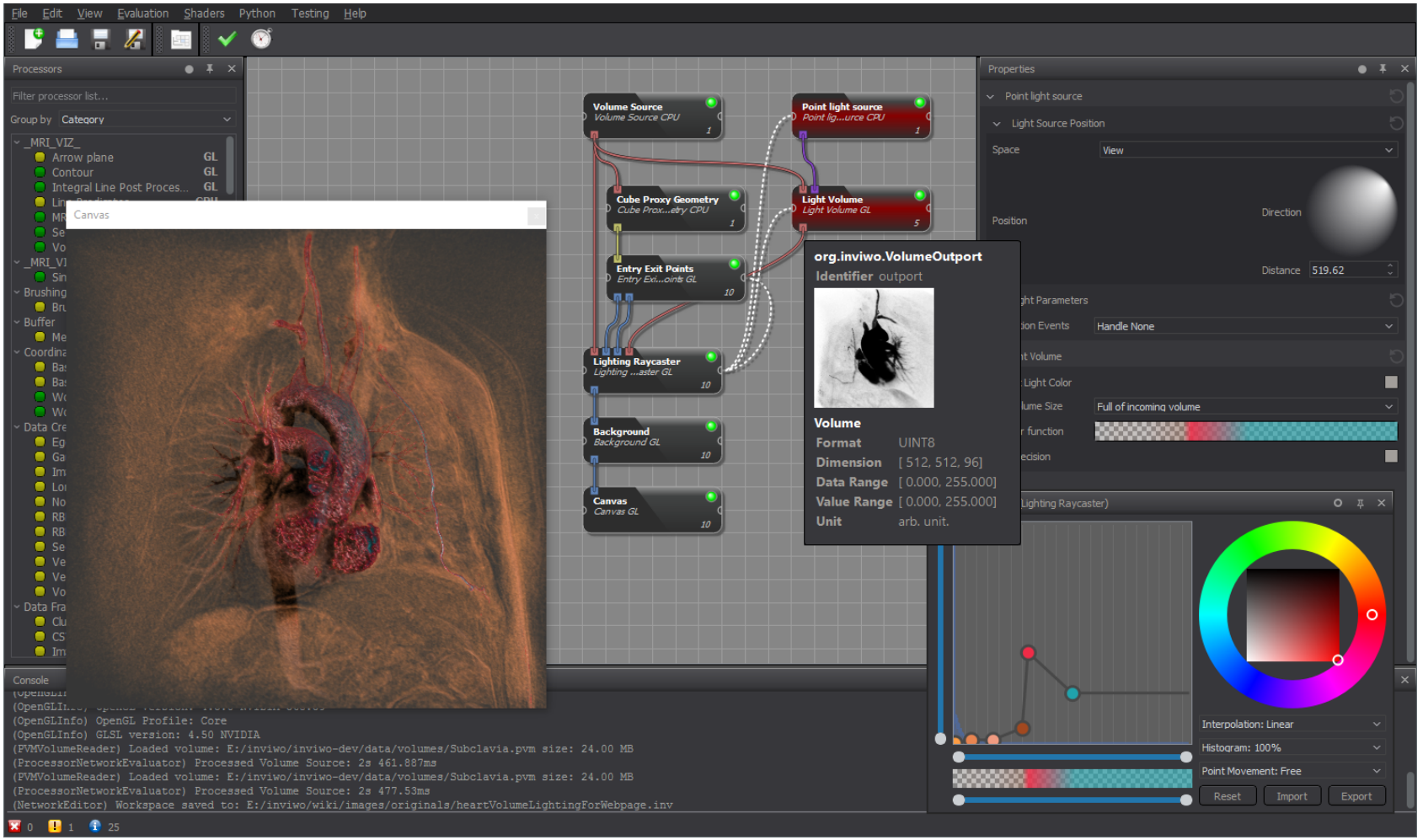
ParaView

Volume rendering software



Voreen

Volume rendering software



Inwivo

Large-Scale Visualisation (Extra slides)