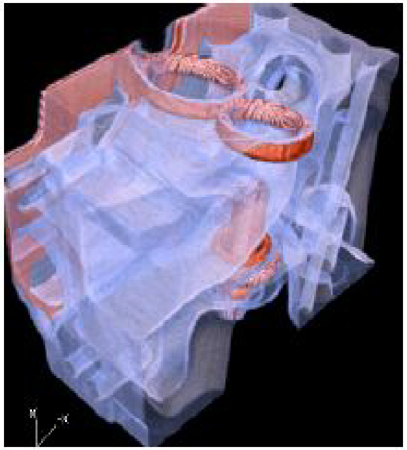
Volume Visualization

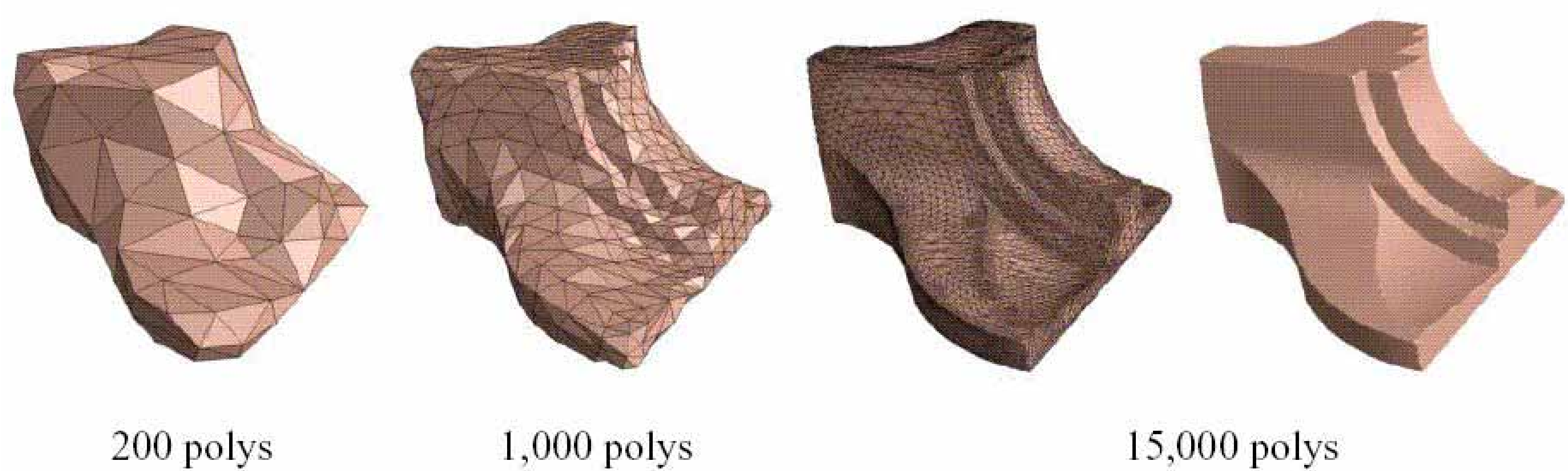



CPSC 414

Abhijeet Ghosh

Surface Graphics

- Objects explicitly defined by a surface or boundary representation:
  - a mesh of polygons



Surface Graphics

- Pros
  - fast rendering algorithms available
  - hardware acceleration cheap (PC game boards!)
  - OpenGL API for programming
  - use texture mapping for added realism

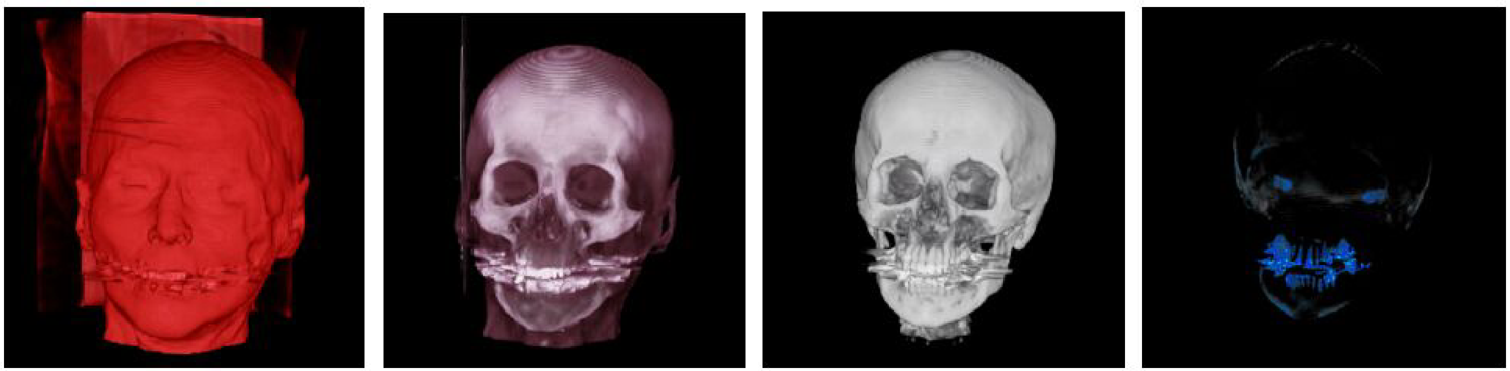
- Cons
  - discards interior of object, maintaining only the shell
  - operations such cutting, slicing & dissection not possible
  - no artificial viewing modes such as semi-transparencies, X-ray
  - surface-less phenomena such as clouds, fog & gas are hard to model and represent

Volume Graphics

- Maintains a discrete representation close to the underlying 3D object
- Different aspects of the dataset can be emphasized via changes in transfer functions
  - translate raw densities into colors and transparencies
- When the nature of the data is not known, it is difficult to create the right polygonal mesh
  - easier to voxelize!

Volume Graphics

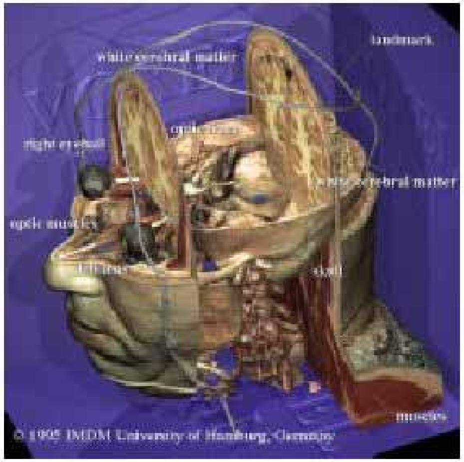
- Pros
  - formidable technique for data exploration



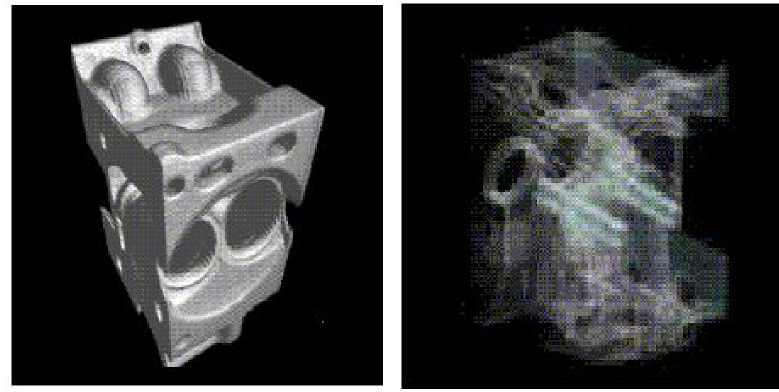
volumetric human head (CT scan)

- Cons
  - rendering algorithm has high complexity!
  - special purpose hardware costly (~$3,000-$10,000)

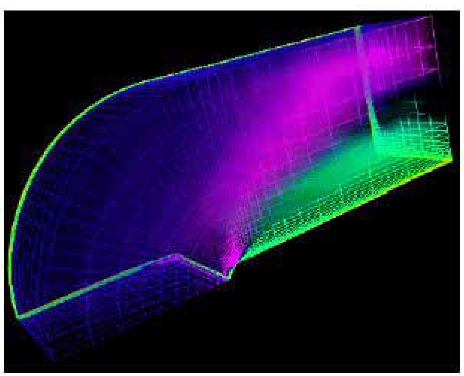
Volume Graphics – Examples



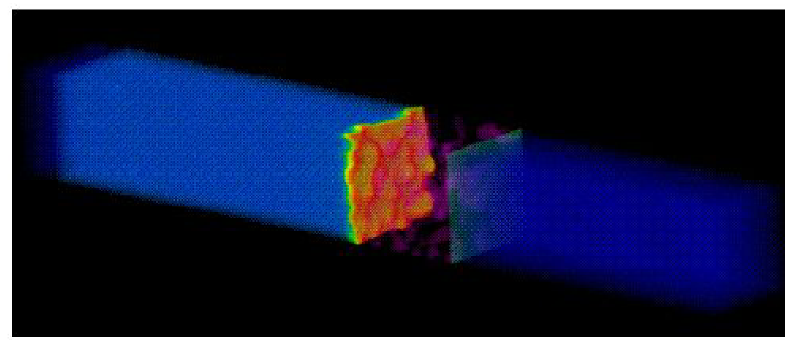
Anatomical atlas from visible human (CT & MRI) datasets



Industrial CT - structural failure and security applications



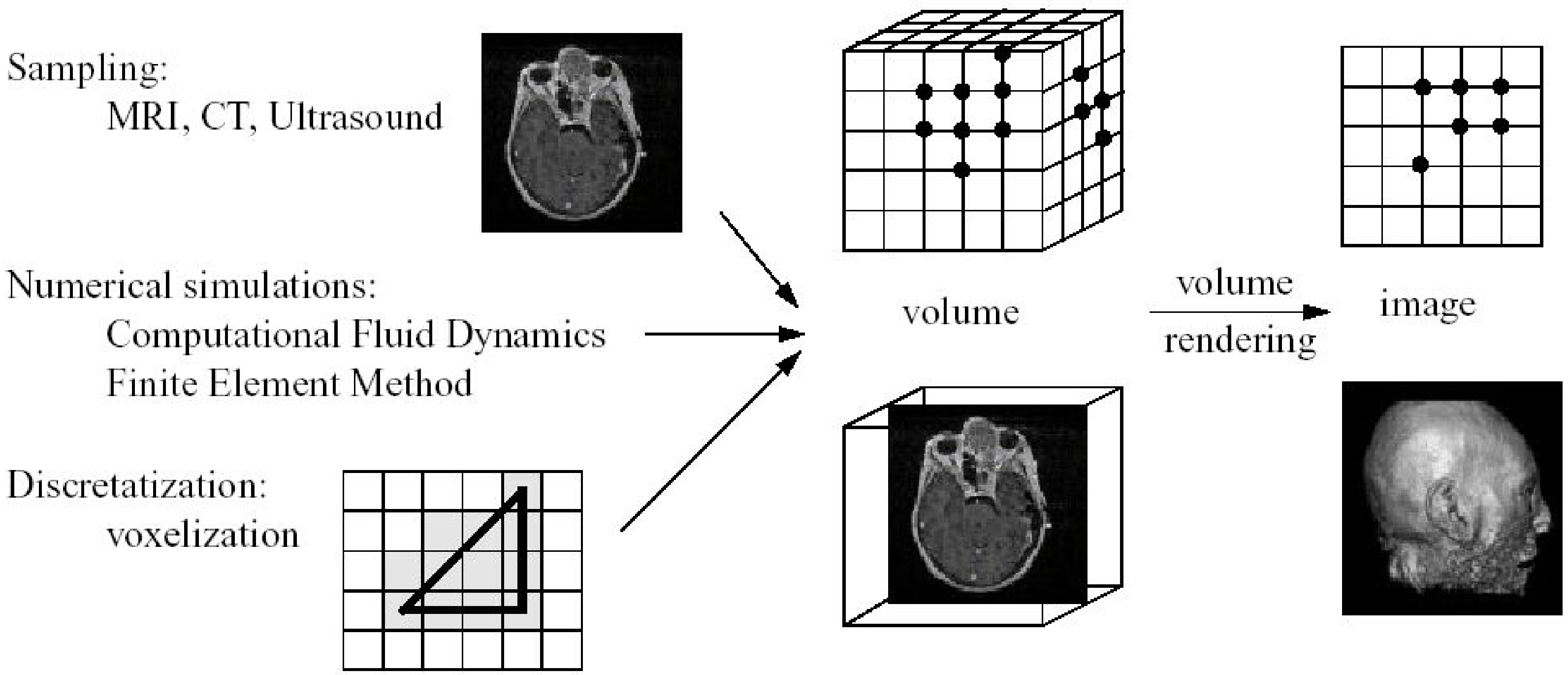
flow around an airplane wing



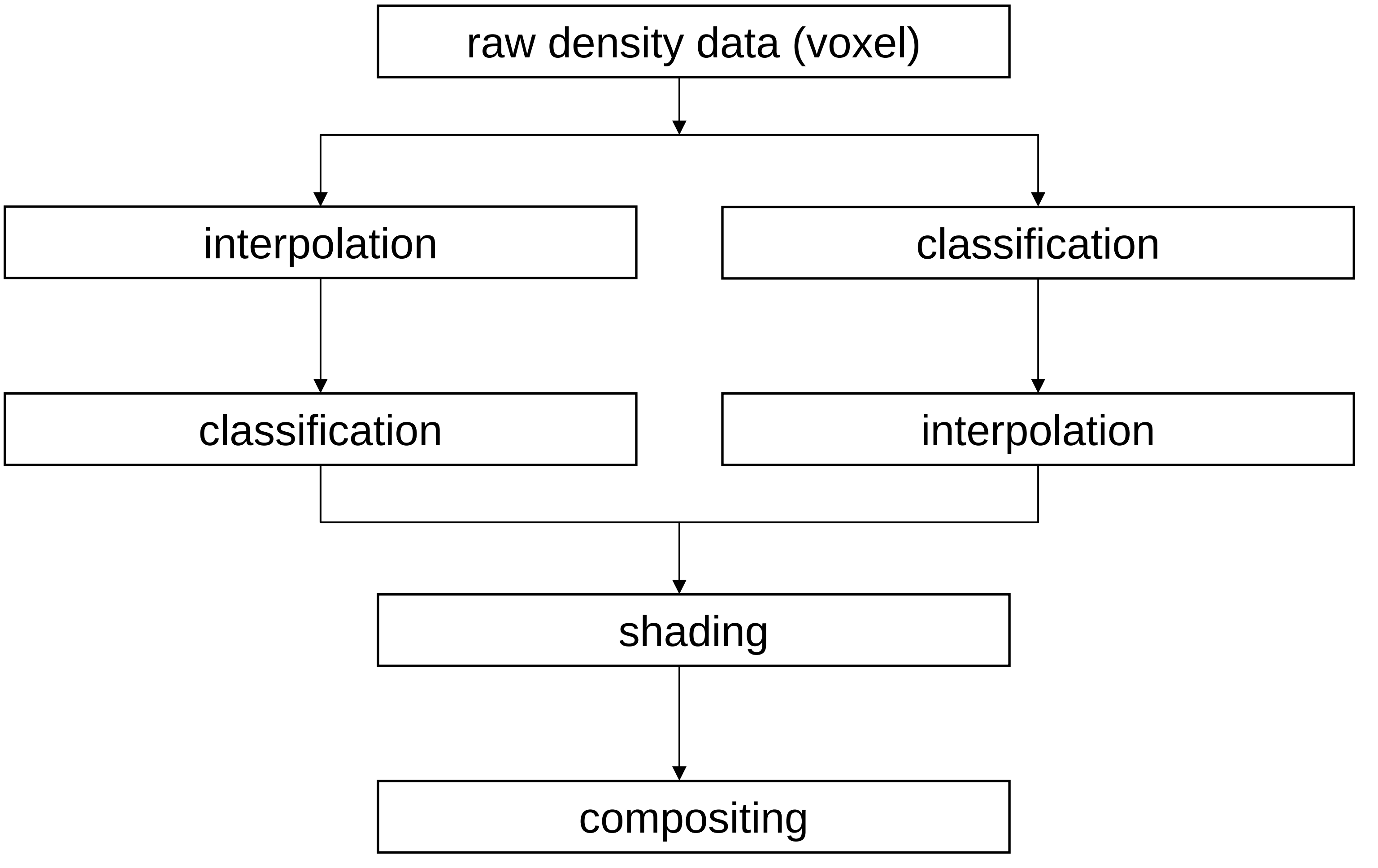
Shockwave visualization – simulation with Navier-Stokes PDEs

Volume Graphics - Basics

- A volume is 3D array of point samples, called *voxels*
  - the point samples are located at the grid points
  - the process of generating a 2D image from the 3D volume is called *volume rendering*



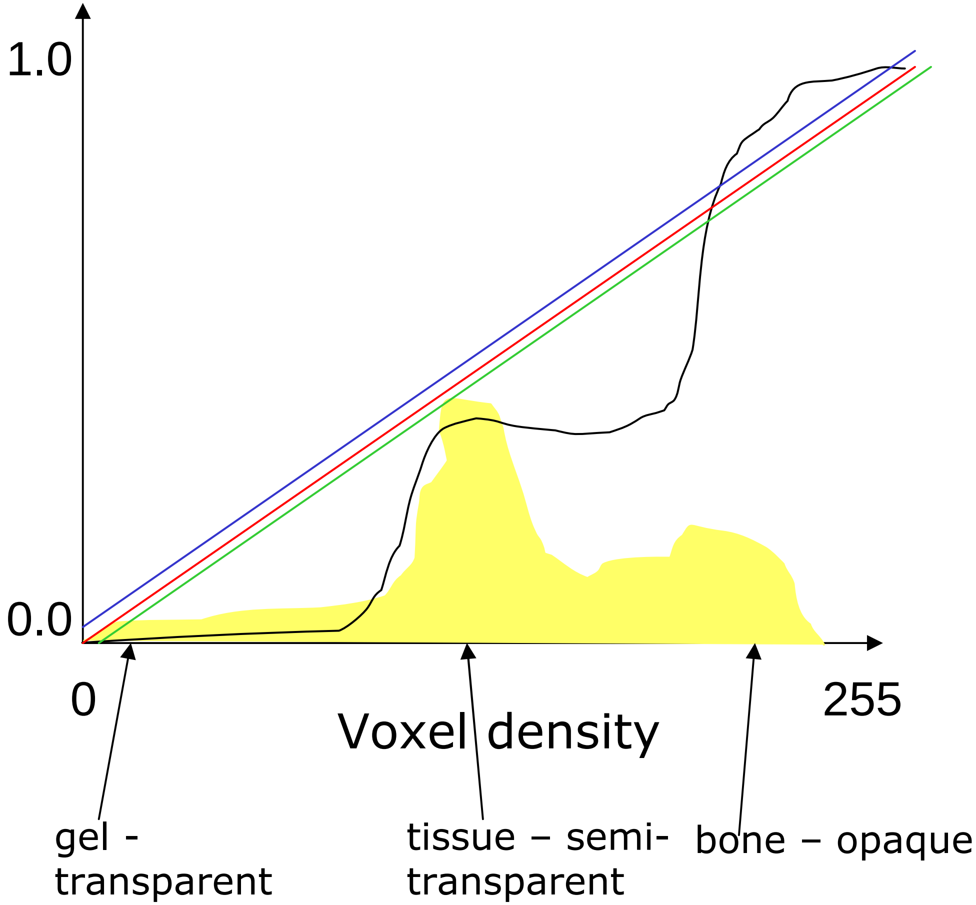
Volume Rendering Pipeline

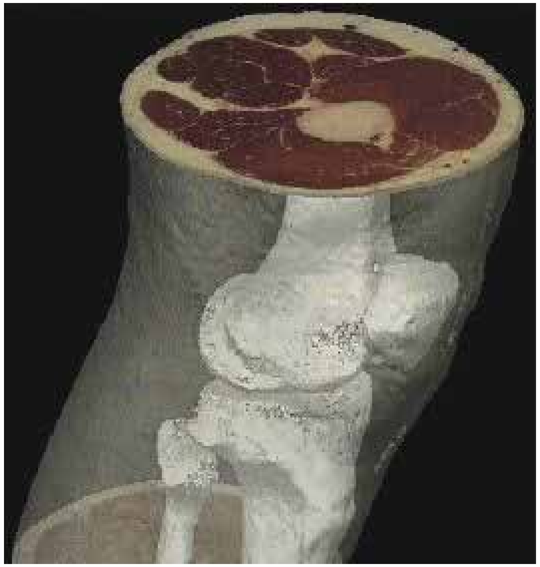


Classification

- A raw voxel stores only density
- Density may have a different meanings:
  - stress, strain, temperature
  - absorption
  - material tag
- Need for assigning meaningful visual attributes such as colors
- Classification is translation of raw values to color and opacity
- Classification done using RGBα transfer functions!

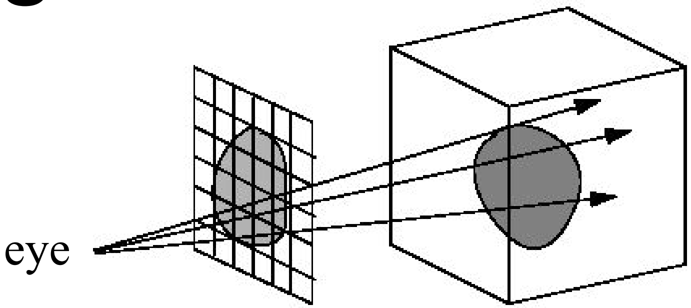
Transfer Functions



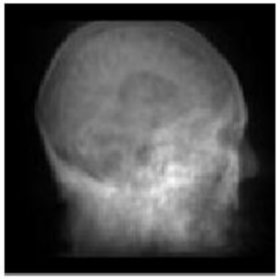


Volume Rendering Modes

- For each pixel in the image, a ray is cast into the volume:

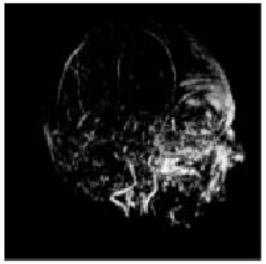


- Four main volume rendering modes exist:



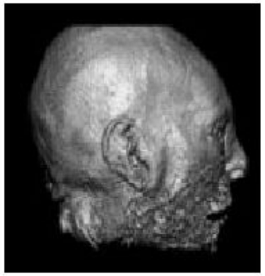
X-Ray:

Rays sum contributions along their path linearly



Maximum Intensity Projection:

A pixel stores the largest intensity values along its ray



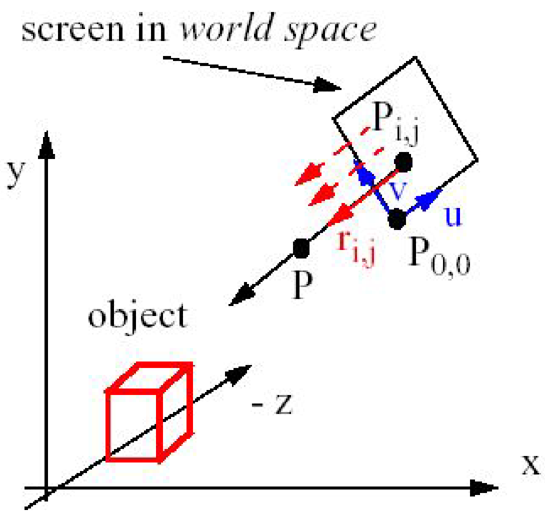
Iso-surface:

Rays **composite** contributions only from voxels of a certain intensity defining a surface

Full-volume:

Rays **composite** contributions along their path linearly

Ray casting – Orthographic



All rays are parallel

A ray is specified as:

r_{ij} = n, the view vector

$n = u \times v$

A point P on a ray is given by:

$P = P_{i,j} + t \times n$

t = step size along ray

Image order projection:
- scan the image in row order,

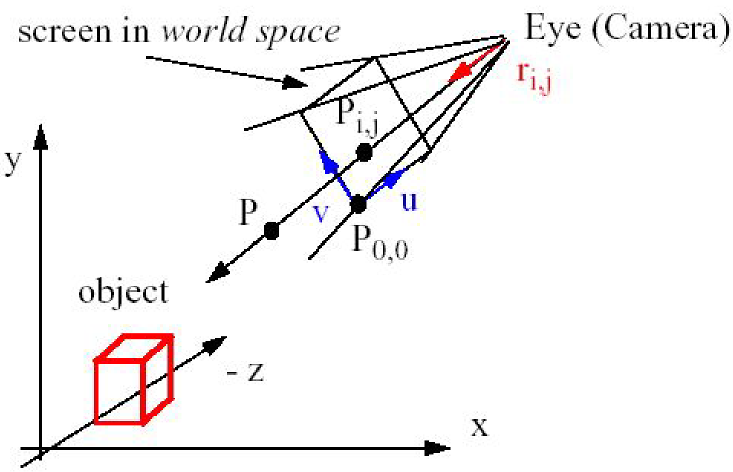
$P_{i,j} = P_{0,0} + j(N_i - 1) + i$

- $P_{i,j}$ location of pixel i, j in world space

$0 <= i <= N_i$ $\quad$ $0 <= j <= N_j$

$P_{0,0}$ = image origin in world space

Ray casting – Perspective



A ray is specified by:

- eye position (Eye)
- screen pixel location ($P_{i,j}$)

r_{ij} = the view vector

= Pi, j – Eye / | Pi, j – Eye |

A point P on a ray is given by:

$P = Eye + t \times r_{i,j}$

t = step size along ray

Image order projection:
- scan the image in row order,

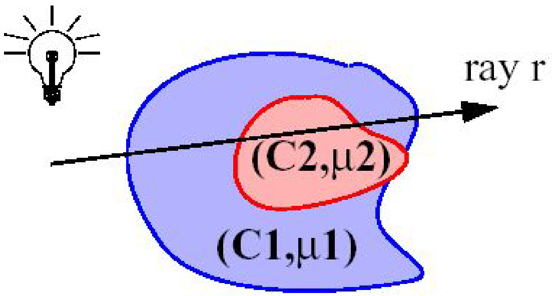
$P_{i,j} = P_{0,0} + j(N_i - 1) + i$

- $P_{i,j}$ location of pixel i, j in world space

$0 <= i <= N_i$ $\quad$ $0 <= j <= N_j$

$P_{0,0}$ = image origin in world space

Volume Rendering Integral



- Consider a volume consisting of particles:
  - each has color C and light attenuating density μ
- A rendering ray accumulates attenuated colors
- The continuous volume rendering integral:

$$I_\lambda(\boldsymbol{x}, \boldsymbol{r}) = \int_0^L C_\lambda(s)\mu(s)e^{\left(-\int_0^s \mu(t)dt\right)} ds$$

analytic evaluation of the integral not efficient

- Approximate it by discretizing it into sampling intervals of width Δs:

$$I_\lambda(\boldsymbol{x}, \boldsymbol{r}) = \sum_{i=0}^{L/\Delta s} C_\lambda(i\Delta s)\mu(i\Delta s)\Delta s \cdot \prod_{j=0}^{i-1} e^{(-\mu(j\Delta s)\Delta s)}$$

Volume Rendering Integral

- A few approximations make the computation more efficient
- Define transparency $t(i\Delta s)$ as: $exp(-\mu(i\Delta s)\,\Delta s) = t(i\Delta s)$
- Opacity α is defined as (1 - transparency): $\alpha(i\Delta s) = (1 - t(i\Delta s))$
- Approximate the exponential term by a two term Taylor expansion:

$$t(i\Delta s) = exp(-\mu(i\Delta s)\,\Delta s) \approx 1 - \mu(i\Delta s)\,\Delta s$$

- Then we can write: $\mu(i\Delta s)\,\Delta s \approx 1 - t(i\Delta s) = \alpha(i\Delta s)$
- Discretized volume rendering integral:

$$I_\lambda(x, r) = \sum_{i=0}^{L/\Delta s} C_\lambda(i\Delta s)\alpha(i\Delta s) \cdot \prod_{j=0}^{i-1} (1 - \alpha(j\Delta s))$$

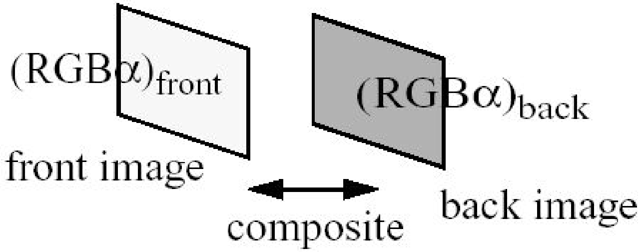
- This equation is used for stepwise compositing of samples along a ray

Compositing

- It is the accumulation of colors weighted by opacities

- Colors and opacities of back pixels are attenuated by opacities of front pixels:

$$rgb = RGB_{back}\,\alpha_{back}(1-\alpha_{front}) + RGB_{front}\,\alpha_{front}$$
$$\alpha \;= \alpha_{back}\,(1-\alpha_{front}) + \alpha_{front}$$



- This leads to the *front-to-back* compositing equation:

$$c = C(i\Delta s)\alpha(i\Delta s)(1 - \alpha) + c$$
$$\alpha = \alpha(i\Delta s)(1 - \alpha) + \alpha$$

advantage – early ray termination!

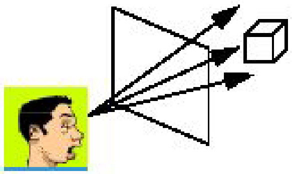
- *back-to-front compositing:*

$$c = c(1 - \alpha(i\Delta s)) + C(i\Delta s)$$
$$\alpha = \alpha\,(1 - \alpha(i\Delta s)) + \alpha(i\Delta s)$$

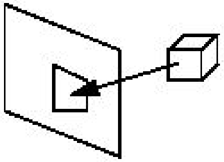
advantage – object order approach suitable for hardware implementation!

Volume Rendering Algorithms

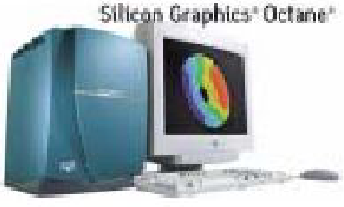
- Ray casting
  - image order, forward viewing



- Splatting
  - object order, backward viewing

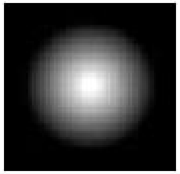


- 2D & 3D texture mapping h/w
  - object order
  - back-to-front compositing

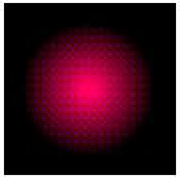


Splatting

- Each voxel represented as a fuzzy ball
  (a 3D Gaussian function)

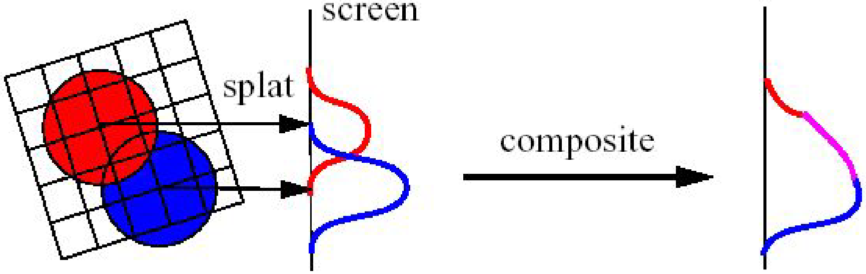


- Each such fuzzy voxel is given an RGBα value
  - based on the transfer function



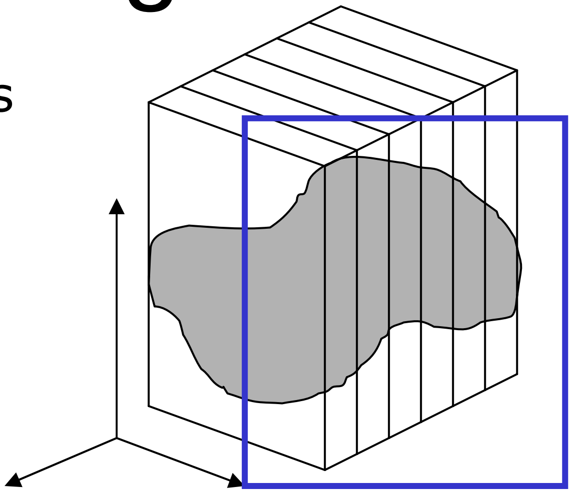
- Fuzzy balls projected onto the screen, leaving a footprint called *splat*

- Simplified algorithm:
  - traverse the voxels in front-to-back order
  - project the voxels to the screen and composite the splats

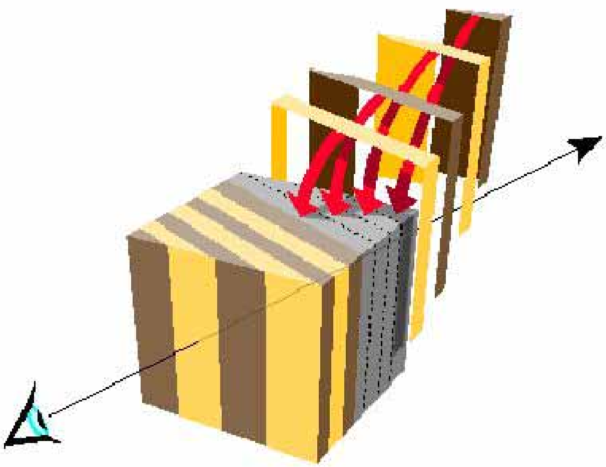


object order algorithm – project only interesting voxels hence fast

Texture Mapping



- 2D: volume as axis aligned 2D textures
  - back-to-front compositing
  - coherent memory access pattern
  - commodity hardware support
  - need for calculating texture coordinates and warping to image plane
- 3D: volume as image aligned 3D textures
  - requires more complex hardware
  - current generation PC game boards!
  - simpler algorithm for generating texture coordinates (directly use u, v, w)



- OpenGL support for compositing

```
glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Volume Visualization

- Acknowledgement:

Klaus Mueller
mueller@cs.sunysb.edu
Stony Brook University
New York - 11794, USA