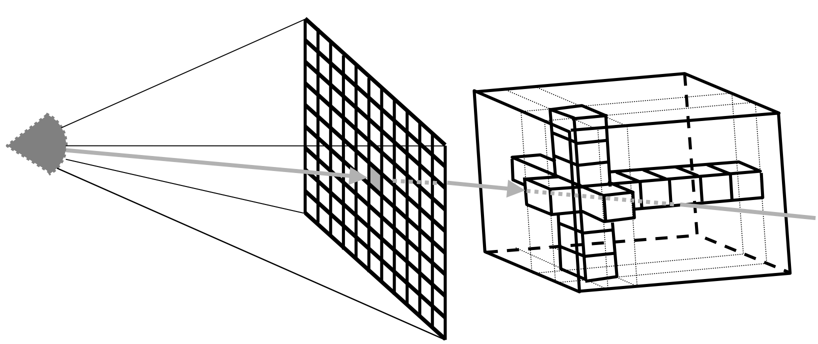
6. Direct Volume Rendering

- Directly get a 3D representation of the volume data
  - The data is considered to represent a semi-transparent light-emitting medium
    - Also gaseous phenomena can be simulated
  - Approaches are based on the laws of physics (emission, absorption, scattering)
  - The volume data is used as a whole (look inside, see all interior structures)

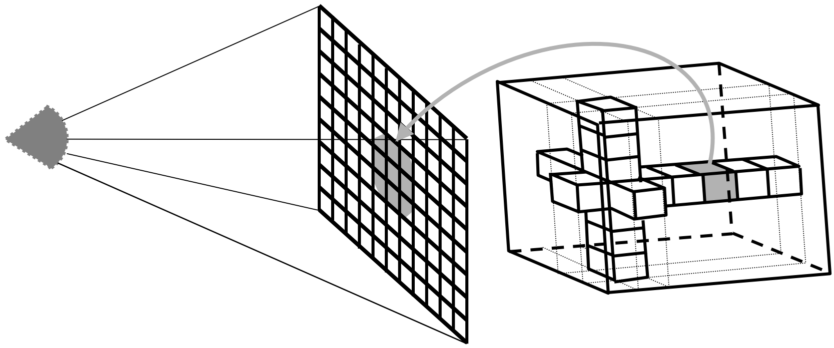
6. Direct Volume Rendering

- Backward methods
  - Image space, image order algorithms
  - Performed pixel by pixel
  - Example: Ray-Casting



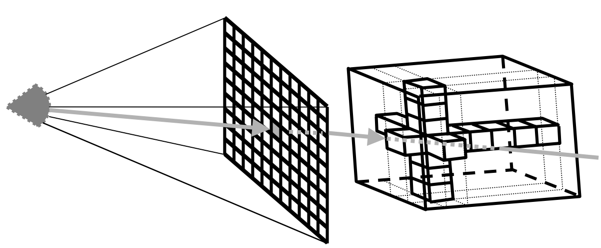
6. Direct Volume Rendering

- Forward Methods
  - Object space, object order algorithm
  - Cell projection
  - Performed voxel by voxel
  - Examples: Slicing, shear-warp, splatting



6.1. Ray Casting

- Similar to ray tracing in surface-based computer graphics
- In volume rendering we only deal with primary rays; hence: *ray-casting*
- Natural image-order technique
- As opposed to surface graphics - how do we calculate the ray/surface intersection ?

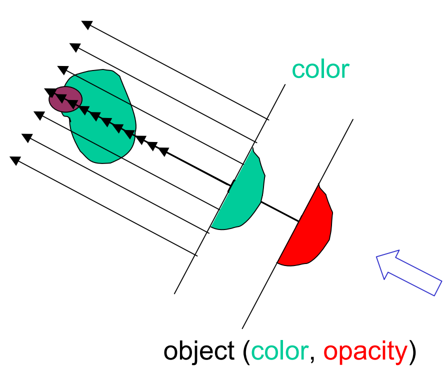


6.1. Ray Casting

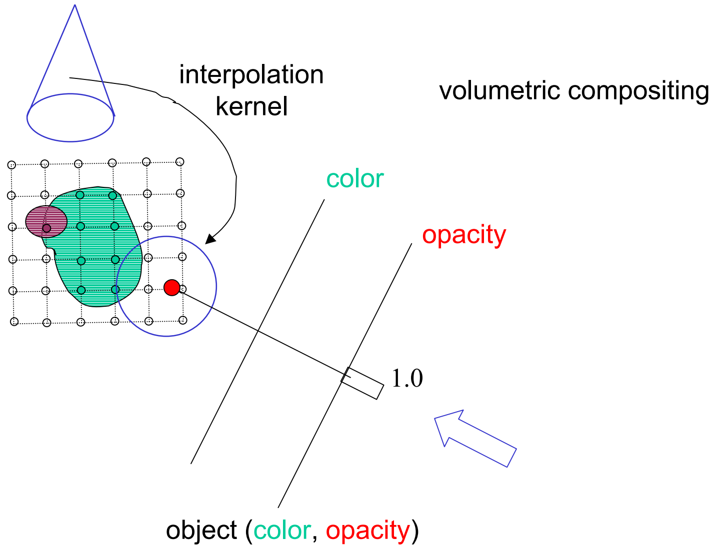
- Since we have no surfaces - we need to carefully step through the volume
- A ray is cast into the volume, sampling the volume at certain intervals
- Sampling intervals are usually equidistant, but don't have to be (e.g. importance sampling)
- At each sampling location, a sample is interpolated / reconstructed from the voxel grid
- Popular filters are: nearest neighbor (box), trilinear, or more sophisticated (Gaussian, cubic spline)

6.1. Ray Casting

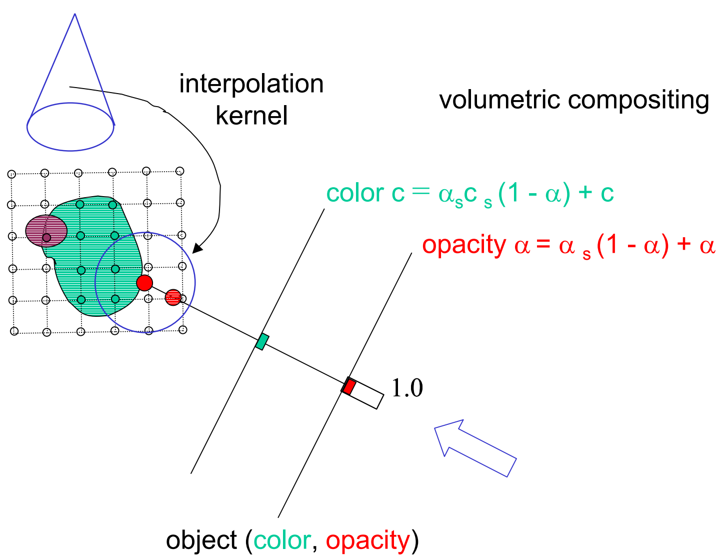
- Volumetric ray integration:
  - Tracing of rays
  - Accumulation of color and opacity along ray: compositing



6.1. Ray Casting

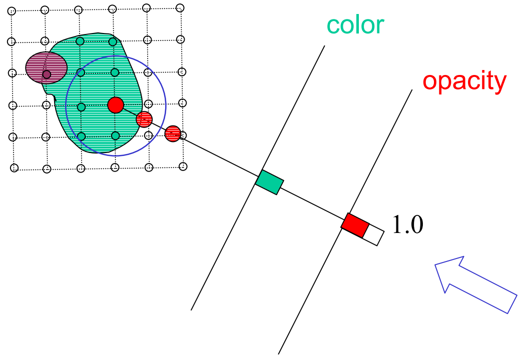


6.1. Ray Casting



6.1. Ray Casting

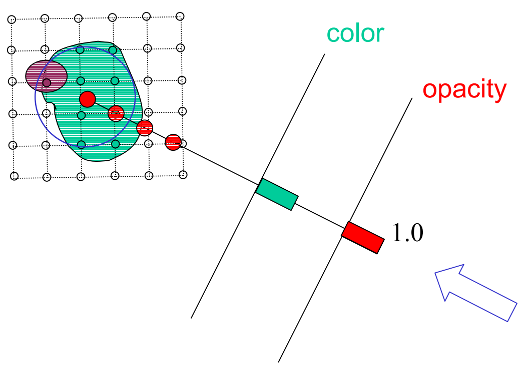
volumetric compositing



object (color, opacity)

6.1. Ray Casting

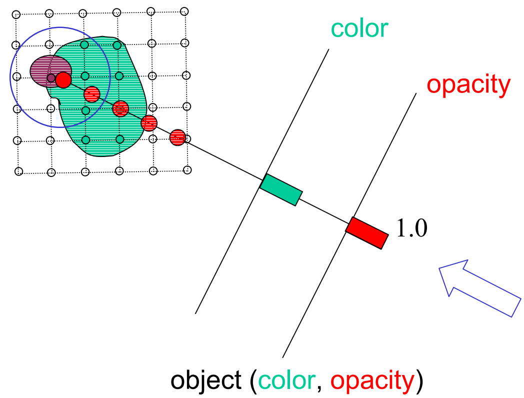
volumetric compositing



object (color, opacity)

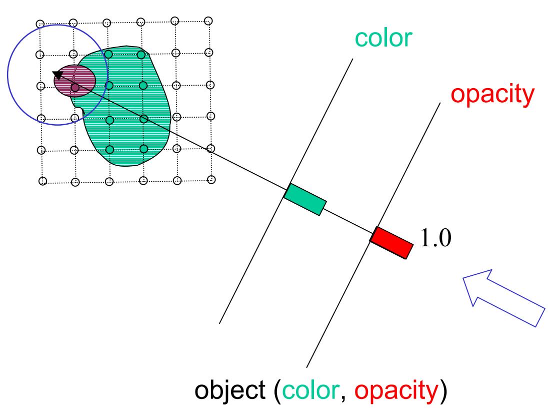
6.1. Ray Casting

volumetric compositing

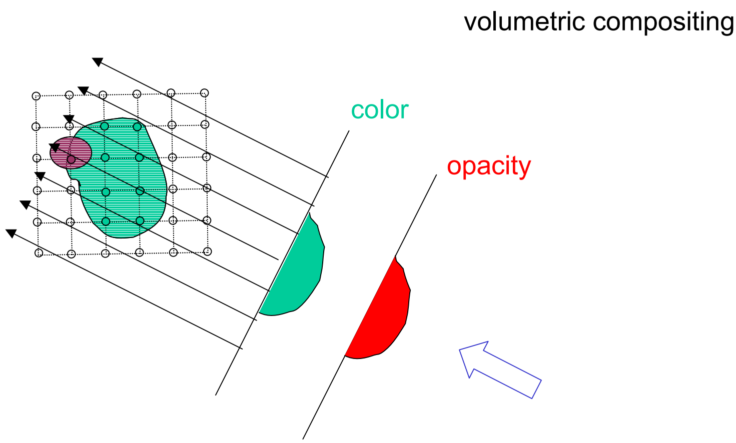


6.1. Ray Casting

volumetric compositing



6.1. Ray Casting



6.1. Ray Casting

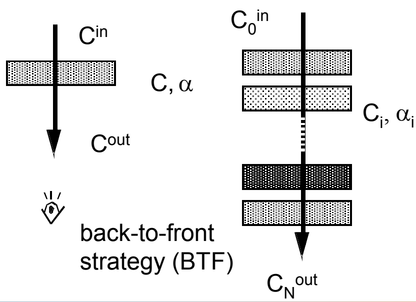
- How is color and opacity at each integration step determined?

- Opacity and (emissive) color in each cell according to classification
- Additional color due to external lighting:
  according to volumetric shading (e.g. Blinn-Phong)

- No shadowing, no secondary effects

6.1. Ray Casting

- Compositing of semi-transparent voxels
  - Physical model: emissive gas with absorption
  - Approximation: density-emitter-model (no scattering)
  - Over operator [Porter & Duff 1984]
  - $C^{out} = C^{in}\,(1 - \alpha) + C\,\alpha = \tilde{C}$ *over* C^{in} with premultiplied Color $\tilde{C} = C\,\alpha$

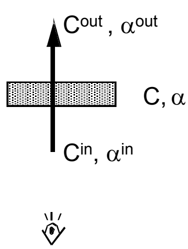


$$C_i^{in} = C_{i-1}^{out}$$

$$\begin{aligned} C_i^{out} &= C_i^{in}(1-\alpha_i) + C_i\alpha_i \\ &= (C_{i-1}^{in}(1-\alpha_{i-1}) + C_{i-1}\,\alpha_{i-1})\,(1-\alpha_i) + C_i\,\alpha_i \\ &= \sum_{i=1,n} C_i\,\alpha_i\,\prod_{j=i+1,n}(1-\alpha_j) \end{aligned}$$

6.1. Ray Casting

- Compositing of semi-transparent voxels *(cont.)*
  - Variation of previous approach for front-to-back strategy (FTB)
  - $C^{out} = C^{in} + (1 - \alpha^{in})\,\alpha C$
  - $\alpha^{out} = \alpha^{in} + (1 - \alpha^{in})\,\alpha$
  - Needs to maintain α !



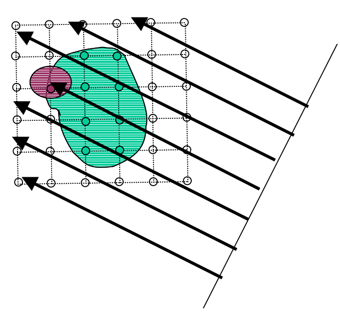
6.1. Ray Casting

- Traversal strategies
  - Front-to-back (most often used in ray casting)
  - Back-to-front (e.g., in texture-based volume rendering)
  - Discrete (Bresenham) or continuous (DDA) traversal of cells

6.2. Acceleration Techniques for Ray Casting

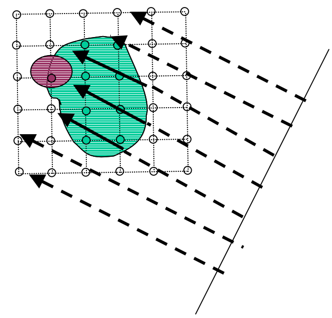
- Problem: ray casting is time consuming
- Idea:
  - Neglect „irrelevant“ information to accelerate the rendering process
  - Exploit coherence

- Early-ray termination
  - Idea: colors from faraway regions do not contribute if accumulated opacity is to high
  - Stop traversal if contribution of sample becomes irrelevant
  - User-set opacity level for termination
  - Front-to-back compositing



6.2. Acceleration Techniques for Ray Casting

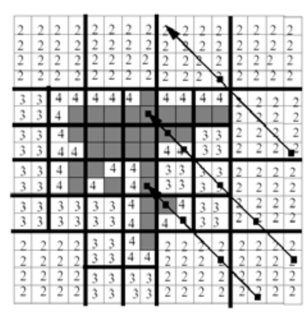
- Space leaping
  - Skip empty cells
- Homogeneity-acceleration
  - Approximate homogeneous regions with fewer sample points



- Approaches:
  - Hierarchical spatial data structure
  - Bounding boxes around objects
  - Adaptive ray traversal
  - Proximity clouds

6.2. Acceleration Techniques for Ray Casting

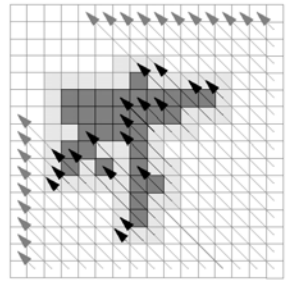
- Hierarchical spatial data structure
  - Octree
  - Mean value and variance stored in nodes of octree
  - Flat Pyramid: store octree level in “empty” voxels



- Bounding boxes around objects
  - Polygon assisted ray casting (PARC)

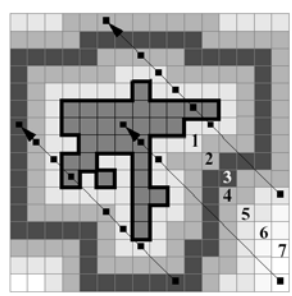
6.2. Acceleration Techniques for Ray Casting

- Adaptive ray traversal
  - Different „velocities“ for traversal
  - Different distance between samples
  - Based on vicinity flag
  - Layer of „vicinity voxels“ around non-transparent parts of the volume



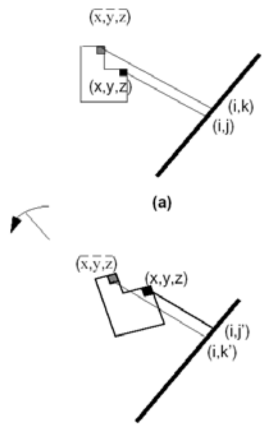
6.2. Acceleration Techniques for Ray Casting

- Proximity clouds
  - Extension of vicinity voxels
  - Store distance to closest „object“ in voxels
  - Allows large integration steps



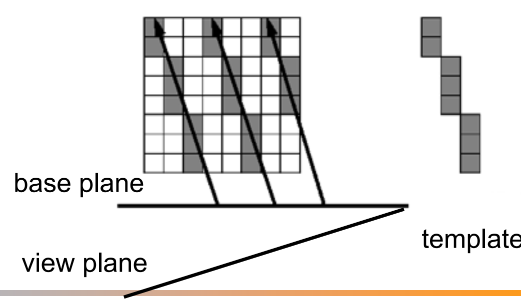
6.2. Acceleration Techniques for Ray Casting

- Exploiting temporal coherence in volume animations
  - C-buffer (Coordinates buffer)
  - Store coordinates of first opaque voxel
  - Removing potentially hidden voxels
  - Or adding potentially visible voxels
  - Criterion: change of position on image plane



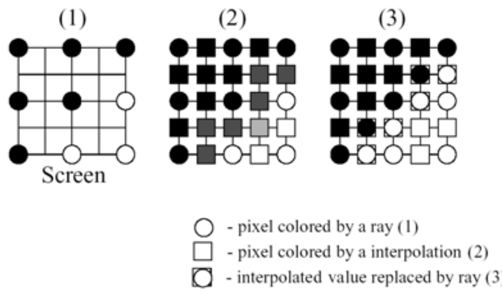
6.2. Acceleration Techniques for Ray Casting

- Template-based volume viewing [Yagel 1991]
  - Ray template stores traversal (steps) through volume
  - Generate template by 3D DDA for parallel rays (orthographic projection)
  - Starting ray templates at pixel positions leads to sampling artefacts
  - Start template ray from base plane parallel to volume orientation
  - Warp image on base plane to view plane



6.2. Acceleration Techniques for Ray Casting

- Adaptive screen sampling [Levoy 1990]
  - Rays are emitted from a subset of pixels (on image plane)
  - Missing values are interpolated
  - In areas of high value gradient additional rays are traced

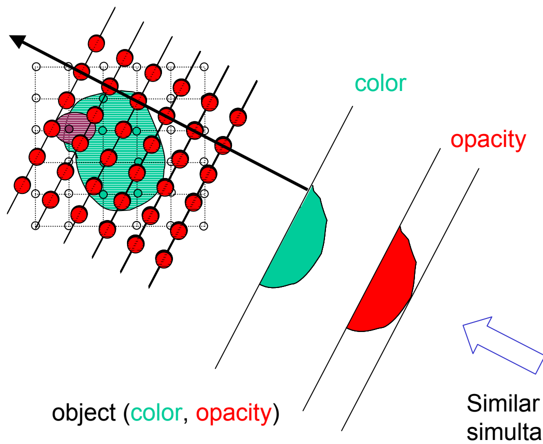


6.3. Texture-Based Volume Rendering

- Object-space approach
- Based on graphics hardware:
  - Rasterization
  - Texturing
  - Blending
- Proxy geometry because there are no volumetric primitives in graphics hardware
- Slices through the volume

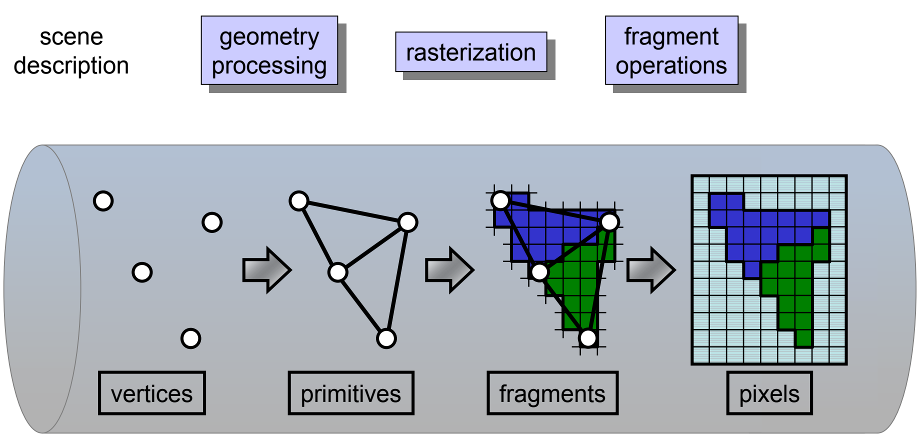
6.3. Texture-Based Volume Rendering

- Slice-based rendering



Similar to ray casting with simultaneous rays

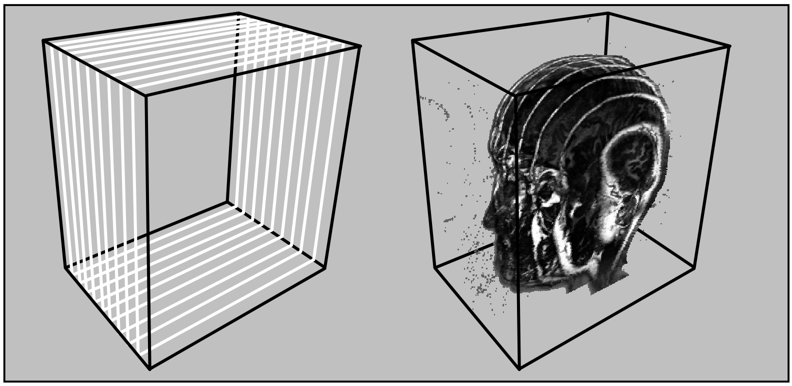
6.3. Texture-Based Volume Rendering



rendering pipeline

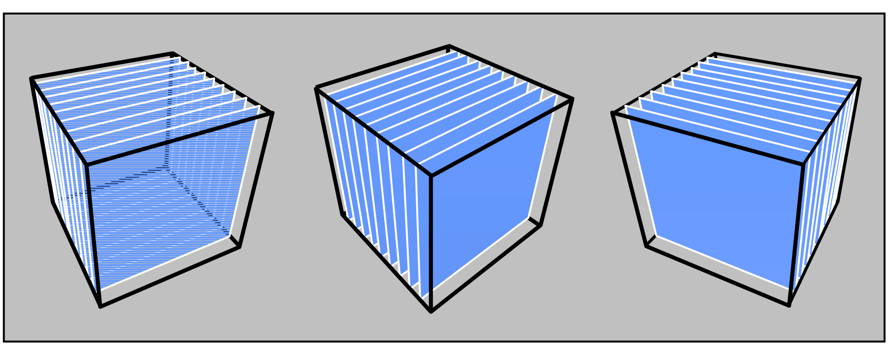
6.3. Texture-Based Volume Rendering

- Proxy geometry
  - Stack of texture-mapped slices
  - Generate fragments
  - Most often back-to-front traversal



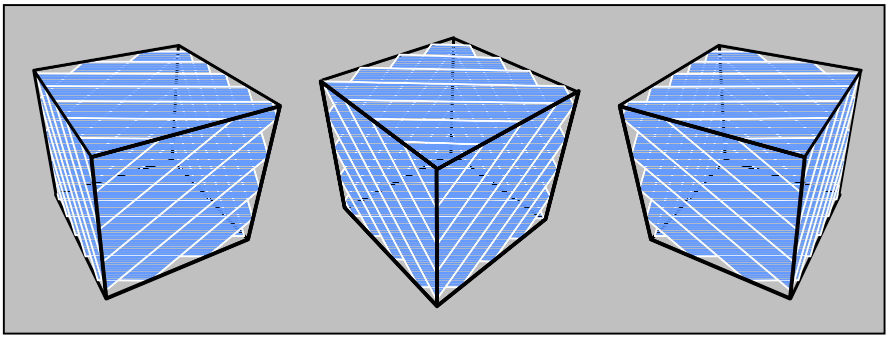
6.3. Texture-Based Volume Rendering

- 2D textured slices
  - Object-aligned slices
  - Three stacks of 2D textures
  - Bilinear interpolation

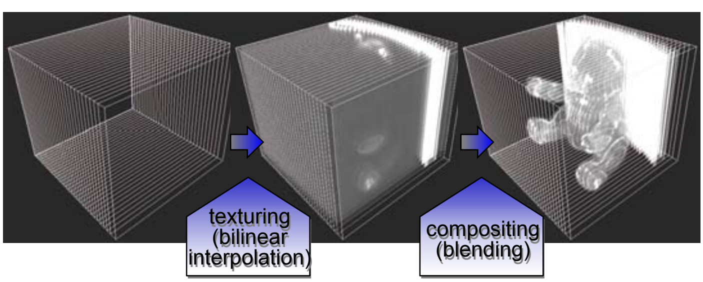


6.3. Texture-Based Volume Rendering

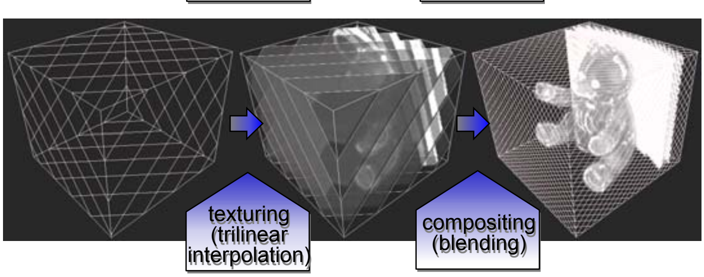
- 3D textured slices
  - View-aligned slices
  - Single 3D texture
  - Trilinear interpolation



6.3. Texture-Based Volume Rendering



2D textures
axis-aligned



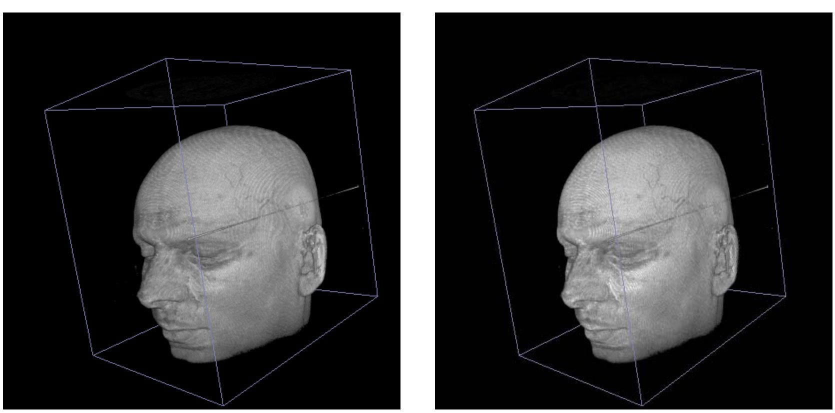
3D texture
view-aligned

6.3. Texture-Based Volume Rendering

- Stack of 2D textures:
  - Works on older graphics hardware which does not support 3D textures
  - Only bilinear interpolation within slices, no trilinear interpolation
    -> fast
    -> problems with image quality
  - Euclidean distance between slices along a light ray depends on viewing parameters
    -> sampling distance depends on viewing direction
    -> apparent brightness changes if opacity is not corrected
  - Artifacts when stack is viewed close to 45 degrees

6.3. Texture-Based Volume Rendering

- Change of brightness in 2D texture-based rendering



6.3. Texture-Based Volume Rendering

- Artifacts when stack is viewed close to 45 degrees

6.3. Texture-Based Volume Rendering

- 3D texture:
  - Needs support for 3D textures
  - Trilinear interpolation within volume
    -> slower
    -> good image quality
  - Constant Euclidean distance between slices along a light ray
  - No artifacts due to inappropriate viewing angles

6.3. Texture-Based Volume Rendering

- Render components
  - Data setup
    - Volume data representation
    - Transfer function representation
  - Data download
    - Volume textures
    - Transfer function tables
  - Per-volume setup (once per frame)
    - Blending mode configuration
    - Texture unit configuration (3D)
    - (Fragment shader configuration)
  - Per-slice setup
    - Texture unit configuration (2D)
  - Generate fragments
    - Proxy geometry rendering

6.3. Texture-Based Volume Rendering

- Representation of volume data by textures
  - Stack of 2D textures
  - 3D texture
- Typical choices for texture format:
  - Intensity
    - Scalar data, post-classification only with special
      - OpenGL extension GL_TEXTURE_COLOR_TABLE_SGI
      - dependent texture lookups (pixel shader)
  - Luminance and alpha
    - Pre-classified (pre-shaded) gray-scale volume rendering
    - Transfer function is already applied to scalar data
    - Change of transfer function requires complete redefinition of texture data
  - RGBA
    - Pre-classified (pre-shaded) colored volume rendering
    - Transfer function is already applied to scalar data

6.3. Texture-Based Volume Rendering

- Typical choices for texture format *(cont.):*
  - Paletted texture
    - Index into color and opacity table (= palette)
    - Index size = byte
    - Index is identical to scalar value
    - Pre-classified (pre-shaded) colored volume rendering
    - Transfer function applied to scalar data during runtime
    - Simple and fast change of transfer functions
    - OpenGL code for paletted 3D texture

```
glTexImage3D ( GL_TEXURE_3D, 0, GL_COLOR_INDEX8_EXT,
               size_x, size_y, GL_COLOR_INDEX,
               GL_UNSIGNED_BYTE, voldata);
```


6.3. Texture-Based Volume Rendering

- Representation of transfer function:
  - For paletted texture only
  - 1D transfer function texture = lookup table
  - OpenGL code

```
glColorTableEXT (GL_SHARED_TEXTURE_PALETTE_EXT,
                 GL_RGBA8, 256*4, GL_RGBA,
                 GL_UNSIGNED_BYTE, palette);
```

  - OpenGL extensions required

6.3. Texture-Based Volume Rendering

- Compositing:
  - Works on fragments
  - Per-fragment operations
  - After rasterization
  - Blending of fragments via over operator
  - OpenGL code for over operator

```
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- Generate fragments:
  - Render proxy geometry
  - Slice
  - Simple implementation: quadrilateral
  - More sophisticated: triangulated intersection surface between slice plane and boundary of the volume data set

6.3. Texture-Based Volume Rendering

- Advantages of texture-based rendering:
  - Supported by consumer graphics hardware
  - Fast for moderately sized data sets
  - Interactive explorations
  - Surface-based and volumetric representations can easily be combined -> mixture with opaque geometries
- Disadvantages:
  - Limited by texture memory
    -> Solution: bricking at the cost of additional texture downloads to the graphics board
  - Brute force: complete volume is represented by slices
  - No acceleration techniques like early-ray termination or space leaping
  - Rasterization speed and memory access can be problematic

6.3. Texture-Based Volume Rendering

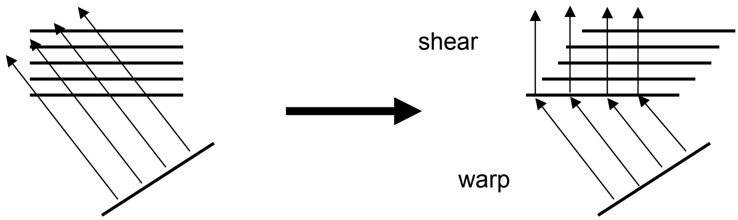
- Outlook on more advanced texture-based volume rendering techniques:
  - Exploit quite flexible per-fragment operations on modern GPUs (nVidia GeForce 3/4 or ATI Radeon 8500)
  - Post-classification (post-shading) possible
  - So-called pre-integration for very high-quality rendering
  - Decompression of compressed volumetric data sets on GPU to save texture memory

6.4. Shear-Warp Factorization

- Object-space method
- Slice-based technique
- Fast object-order rendering
- Accelerated volume visualization via shear-warp factorization [Lacroute & Levoy 1994]
- Software-based implementation

6.4. Shear-Warp Factorization

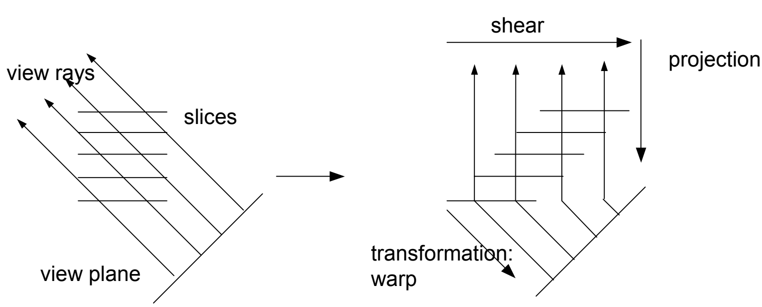
- General goal: make viewing rays parallel to each other and perpendicular to the image
- This is achieved by a simple shear



- Parallel projection (orthographic camera) is assumed
- Extension for perspective projection possible

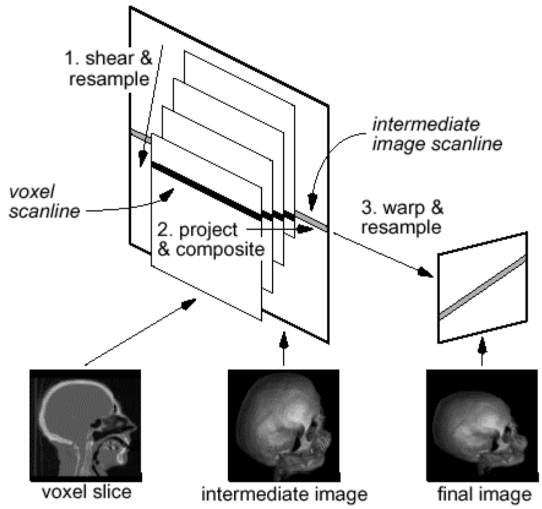
6.4. Shear-Warp Factorization

- Algorithm:
  - Shear along the volume slices
  - Projection and compositing to get intermediate image
  - Warping transformation of intermediate image to get correct result



6.4. Shear-Warp Factorization

- For one scan line



6.4. Shear-Warp Factorization

- Mathematical description of the shear-warp factorization
- Splitting the viewing transformation into separate parts

$$\mathbf{M}_{\text{view}} = \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{M}_{\text{warp}}$$

- $\mathbf{M}_{\text{view}}$ = general viewing matrix
- $\mathbf{P}$ = permutation matrix: transposes the coordinate system in order to make the *z*-axis the principal viewing axis
- $\mathbf{S}$ = transforms volume into sheared object space
- $\mathbf{M}_{\text{warp}}$ = warps sheared object coordinates into image coordinates

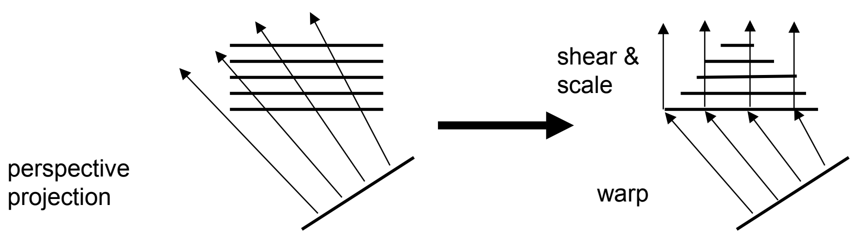
- Needs 3 stacks of the volume along 3 principal axes

6.4. Shear-Warp Factorization

- Shear for parallel and perspective projections

$$S_{\text{par}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s_x & s_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S_{\text{persp}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s'_x & s'_y & 1 & s'_w \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

shear perpendicular to *z*-axis — shear and scale



6.4. Shear-Warp Factorization

- Algorithm (detailed):
  - Transform volume to sheared object space by translation and resampling
  - Project volume into 2D intermediate image in sheared object space
    - Composite resampled slices front-to-back
  - Transform intermediate image to image space using 2D warping
- In a nutshell:
  - Shear (3D)
  - Project (3D → 2D)
  - Warp (2D)

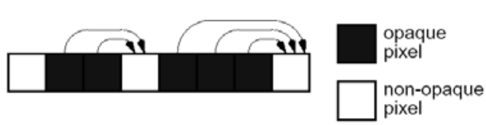
6.4. Shear-Warp Factorization

- Three properties
  - Scan lines of pixels in the intermediate image are parallel to scan lines of voxels in the volume data
  - All voxels in a given voxel slice are scaled by the same factor
  - Parallel projections only:
    Every voxel slice has the same scale factor

- Scale factor for parallel projections
  - This factor can be chosen arbitrarily
  - Choose a unity scale factor so that for a given voxel scan line there is a one-to-one mapping between voxels and intermediate image pixels

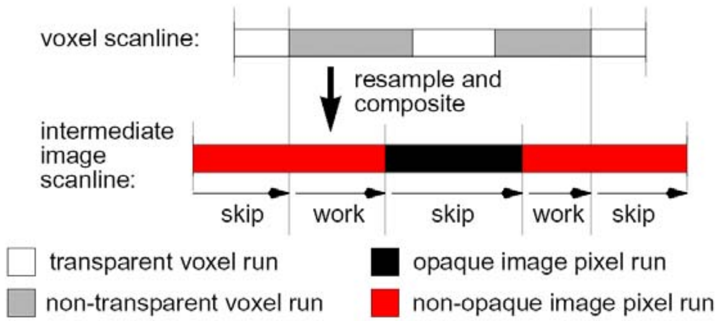
6.4. Shear-Warp Factorization

- Highly optimized algorithm for
  - Parallel projection and
  - Fixed opacity transfer function
- Optimization of volume data (voxel scan lines)
  - Run-length encoding of voxel scan lines
  - Skip runs of transparent voxels
  - Transparency and opaqueness determined by user-defined opacity threshold
- Optimization in intermediate image:
  - Skip opaque pixels in intermediate image (analogously to early-ray termination)
  - Store (in each pixel) offset to next non-opaque pixel



6.4. Shear-Warp Factorization

- Combining both ideas:
  - First property (parallel scan lines for pixels and voxels):
    Voxel scan lines in sheared volume are aligned with pixel scan lines in intermediate
  - Both can be traversed in scan line order simultaneously



6.4. Shear-Warp Factorization

- Coherence in voxel space:
  - Each slice of the volume is only translated
  - Fixed weights for bilinear interpolation within voxel slices
  - Computation of weights only once per frame
- Final warping:
  - Works on composited intermediate image
  - Warp: affine image warper with bilinear filter
  - Often done in hardware:
    render a quadrilateral with intermediate 2D image being attached as 2D texture

6.4. Shear-Warp Factorization

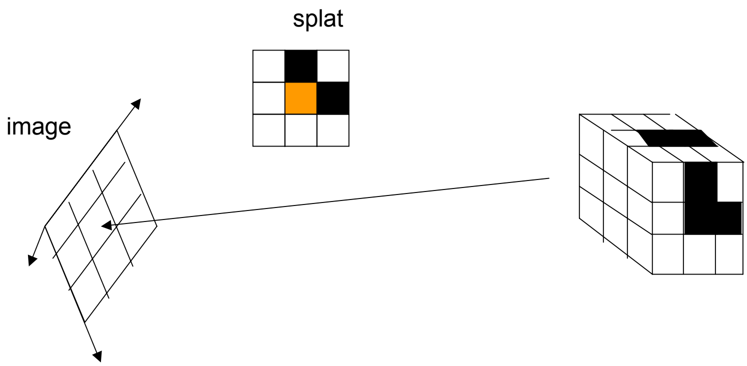
- Parallel projection:
  - Efficient reconstruction
  - Lookup table for shading
  - Lookup table for opacity correction (thickness)
  - Three RLE of the actual volume (in x, y, z)
- Perspective projection:
  - Similar to parallel projection
  - Difference: voxels need to be scaled
  - Hence more than two voxel scan lines needed for one image scan line

6.5. Splatting

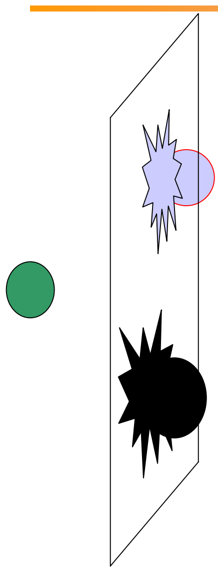
- Splatting [Westover 1990]
- Object-order method
- Original method: fast, poor quality
- Many improvements since then

6.5. Splatting

- Project each sample (voxel) from the volume into the image plane



6.5. Splatting



6.5. Splatting

- Ideally we would reconstruct the continuous volume (cloud) using the interpolation kernel *w* (spherically symmetric):

$$f_r(v) = \sum_k w(v - v_k) f(v_k)$$

- Analytic integral along a ray *r* for intensity (emission):

$$I(p) = \int f_r(p+r)\,dr = \int \sum_k w(p+r-v_k) f(v_k)\,dr$$

- Rewrite:

$$I(p) = \sum_k f(v_k) \cdot \boxed{\int w(p+r-v_k)\,dr}$$

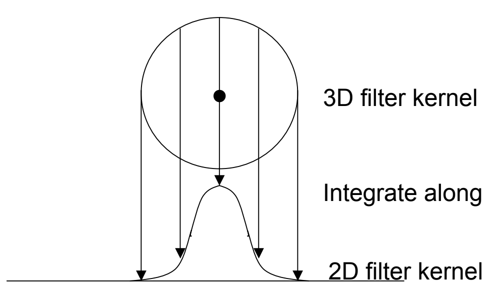
splatting kernel (= "splat")

6.5. Splatting

- Discretization via 2D splats

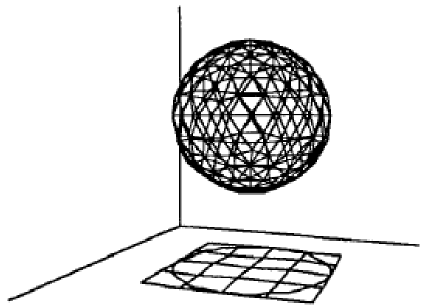
$$\text{Splat}(x,y) = \int w(x,y,z)\,dz$$

from the original 3D kernel
- The 3D rotationally symmetric filter kernel is integrated to produce a 2D filter kernel



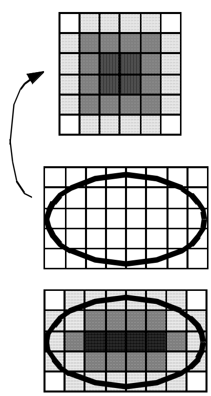
6.5. Splatting

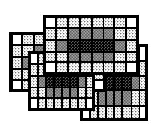
- Draw each voxel as a cloud of points (footprint) that spreads the voxel contribution across multiple pixels
- Footprint: splatted (integrated) kernel
- Approximate the 3D kernel $h(x,y,z)$ extent by a sphere



6.5. Splatting

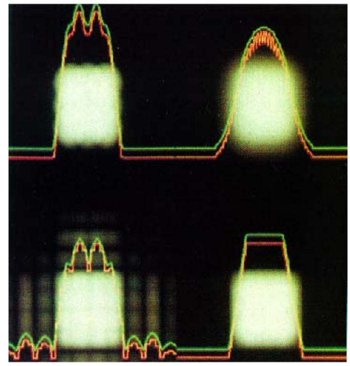
- Larger footprint increases blurring and used for high pixel-to-voxel ratio
- Footprint geometry
  - Orthographic projection: footprint is independent of the view point
  - Perspective projection: footprint is elliptical
- Pre-integration of footprint
- For perspective projection: additional computation of the orientation of the ellipse

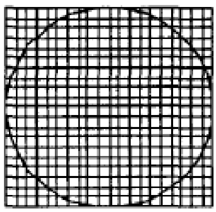


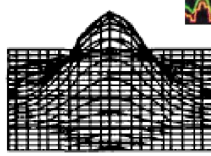


6.5. Splatting

- The choice of kernel can affect the quality of the image
- Examples are cone, Gaussian, sinc, and bilinear function
- Effects of kernel function



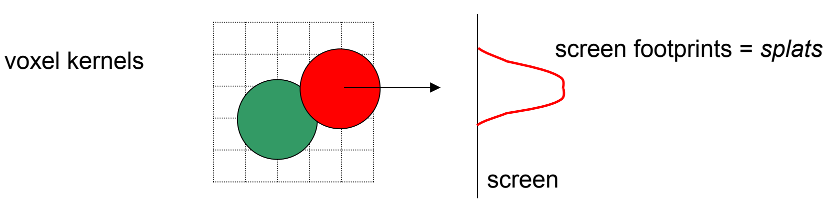




1D integration of 3D Gaussian is a 2D Gaussian

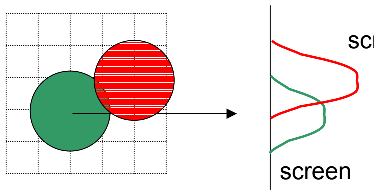
6.5. Splatting

- Volume = field of 3D interpolation kernels
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
- Weighted footprints accumulate into image



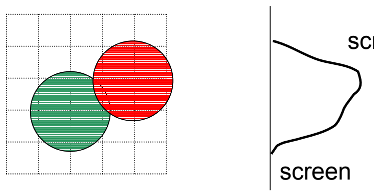
6.5. Splatting

- Volume = field of 3D interpolation kernels
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
- Weighted footprints accumulate into image



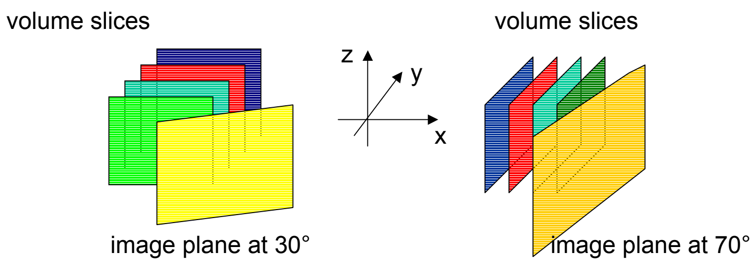
6.5. Splatting

- Volume = field of 3D interpolation kernels
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
- Weighted footprints accumulate into image



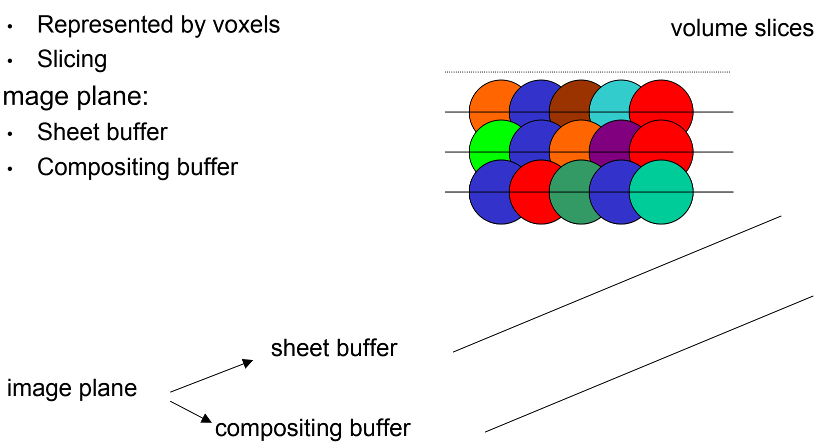
6.5. Splatting

- Voxel kernels are added within sheets
- Sheets are composited front-to-back
- Sheets = volume slices most perpendicular to the image plane (analogously to stack of slices)



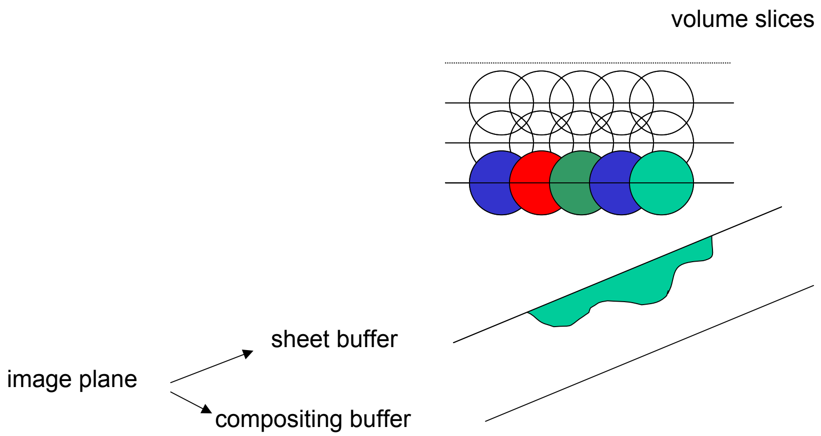
6.5. Splatting

- Core algorithm for splatting
- Volume
  - Represented by voxels
  - Slicing
- Image plane:
  - Sheet buffer
  - Compositing buffer



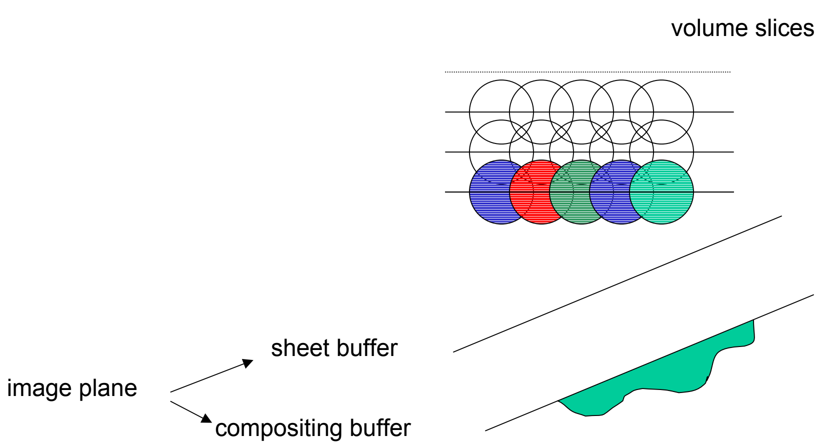
6.5. Splatting

- Add voxel kernels within first sheet



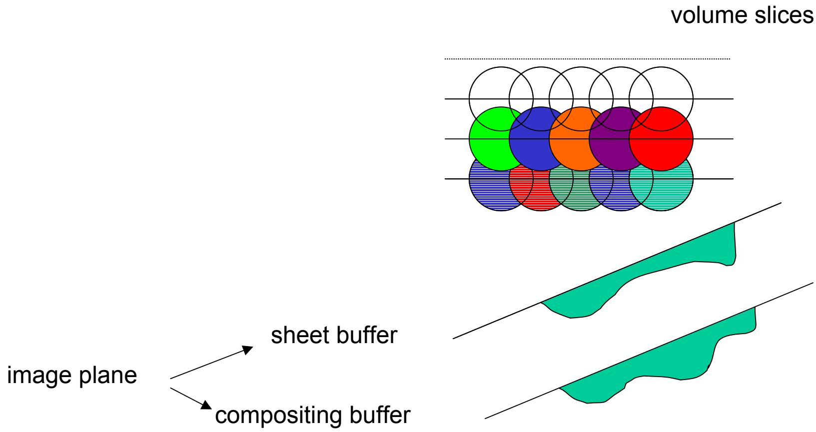
6.5. Splatting

- Transfer to compositing buffer



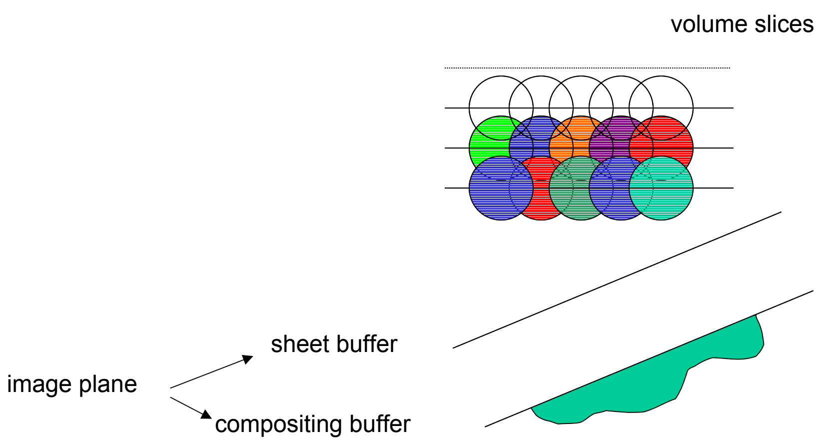
6.5. Splatting

- Add voxel kernels within second sheet



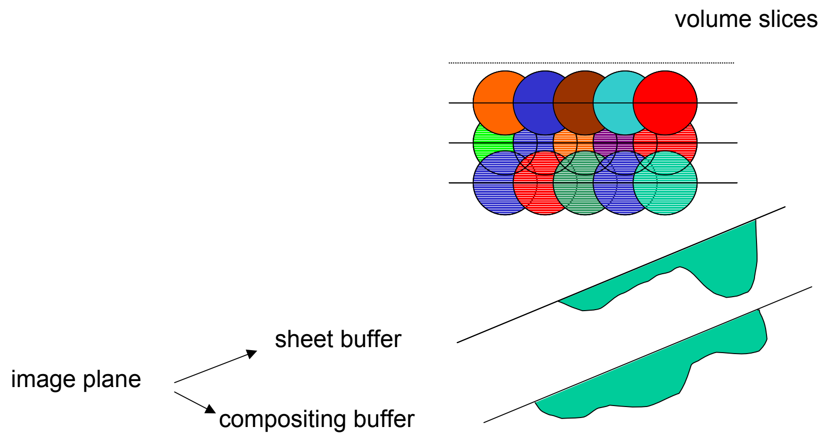
6.5. Splatting

- Composite sheet with compositing buffer



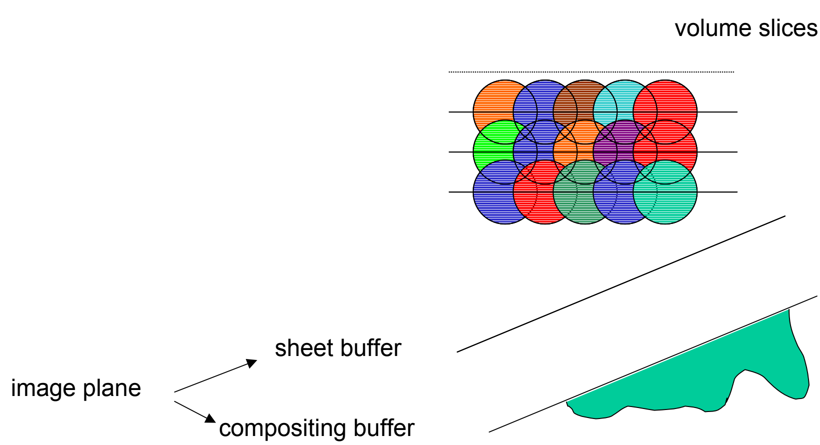
6.5. Splatting

- Add voxel kernels within third sheet



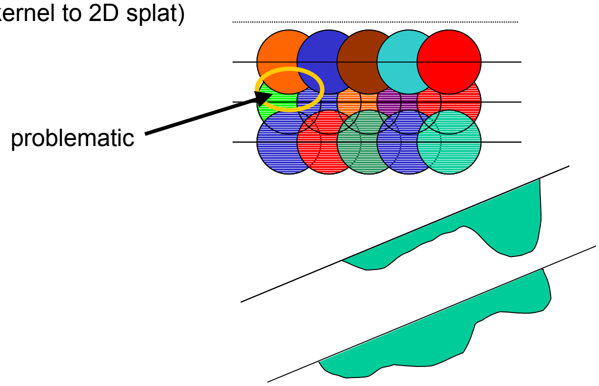
6.5. Splatting

- Composite sheet with compositing buffer



6.5. Splatting

- Inaccurate compositing
- Problems when splats overlap
- Incorrect mixture of
  - Integration (3D kernel to 2D splat) and
  - Compositing

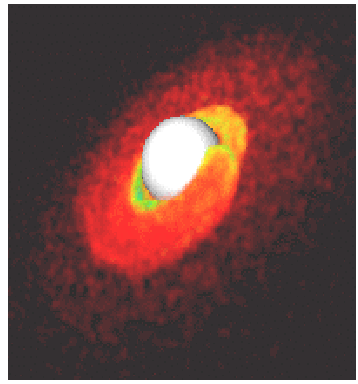


6.5. Splatting

- Advantages:
  - Footprints can be pre-integrated
  - Quite fast: voxel interpolation is in 2D on screen
  - Simple static parallel decomposition
  - Acceleration approach: only relevant voxels must be projected
- Disadvantages:
  - Blurry images for zoomed views
  - High fill-rate for zoomed splats
  - Reconstruction and integration to be performed on a per-splat basis
  - Dilemma when splats overlap

6.5. Splatting

- Simple extension to volume data without grids
  - Scattered data with kernels
  - Example: SPH (smooth particle hydrodynamics)
  - Needs sorting of sample points



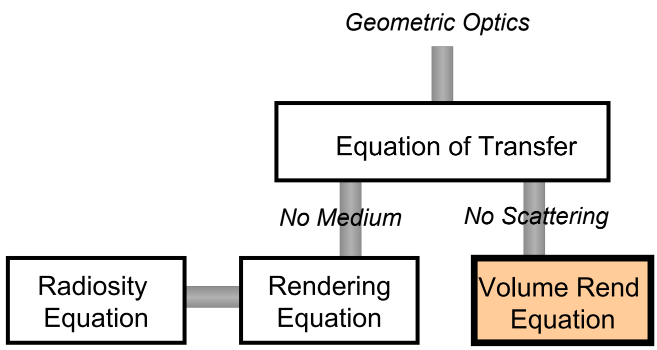
6.6. Equation of Transfer for Light

- Goal: physical model for volume rendering
  - Emission-absorption model
  - Density-emitter model [Sabella 1988]

- More general approach:
  - Linear transport theory
  - Equation of transfer for radiation
  - Basis for all rendering methods
- Important aspects:
  - Absorption
  - Emission
  - Scattering
  - Participating medium

6.6. Equation of Transfer for Light

- The Grand Scheme

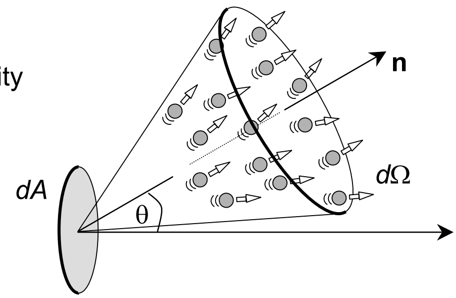


6.6. Equation of Transfer for Light

- Assumptions:
  - Based on a physical model for radiation
  - Geometrical optics
- Neglect:
  - Diffraction
  - Interference
  - Wave-character
  - Polarization
- Interaction of light with matter at the macroscopic scale
  - Describes the changes of specific intensity due to absorption, emission, and scattering
- Based on energy conservation
- Expressed by equation of transfer

6.6. Equation of Transfer for Light

- Basic quantity of light: radiance *I*
- Sometimes called specific intensity

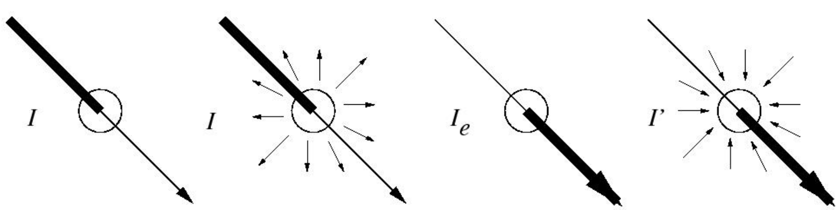


$$\delta E = I(\mathbf{x},\mathbf{n},\nu)\overbrace{\cos\theta\, dA}^{\text{projected area element}}\, d\Omega\, d\nu\, dt$$

radiative energy
radiance
position
direction
frequency
solid angle
time

6.6. Equation of Transfer for Light

- Contributions to radiation at a single position:
  - Absorption
  - Emission
  - Scattering

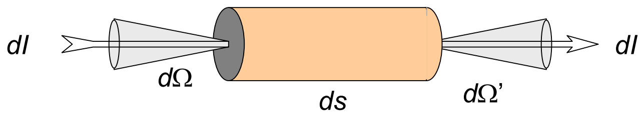


Absorption outscattering emission inscattering

6.6. Equation of Transfer for Light

- Absorption
- Total absorption coefficient or total extinction coefficient $\chi(\mathbf{x},\mathbf{n},\nu)$
- Loss of radiative energy through a cylindrical volume element:

$$\delta E^{(ab)} = \chi(\mathbf{x},\mathbf{n},\nu)\, I(\mathbf{x},\mathbf{n},\nu)\, ds\, dA\, d\Omega\, d\nu\, dt$$

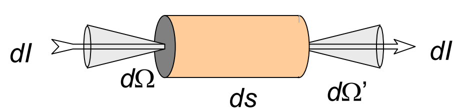


- $1/\chi$ is called mean free path

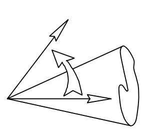
6.6. Equation of Transfer for Light

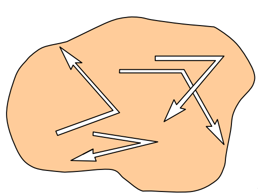
- Total absorption coefficient consists of:
  - True absorption coefficient $\kappa(\mathbf{x},\mathbf{n},\nu)$
  - Scattering coefficient $\sigma(\mathbf{x},\mathbf{n},\nu)$

$$\chi = \kappa + \sigma$$



removal of radiative energy by true absorption (conversion to thermal energy)





scattering out of solid angle $d\Omega$

6.6. Equation of Transfer for Light

- Emission
- Emission coefficient $\eta(\mathbf{x},\mathbf{n},\nu)$
- Emission of radiative energy within a cylindrical volume element:

$$\delta E^{(\mathrm{em})} = \eta(\mathbf{x},\mathbf{n},\nu)\, ds\, dA\, d\Omega\, d\nu\, dt$$

- Consists of two parts:
  - Thermal part or source term $q(\mathbf{x},\mathbf{n},\nu)$
  - Scattering part $j(\mathbf{x},\mathbf{n},\nu)$

$$\eta = q + j$$

6.6. Equation of Transfer for Light

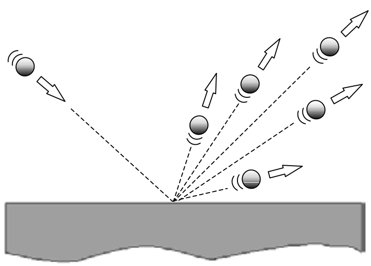
- Scattering
- Described by phase function $p(\mathbf{x},\mathbf{n},\mathbf{n}',\nu,\nu')$

$$\delta E^{(\mathrm{scat})} = \sigma I\, ds\, dA\, d\Omega\, d\nu\, dt \cdot \frac{1}{4\pi} p(\mathbf{x},\mathbf{n},\mathbf{n}',\nu,\nu')\, d\Omega'\, d\nu'$$

- Elastic scattering:
  - No change of frequency
  - $p(\mathbf{x},\mathbf{n},\mathbf{n}')$

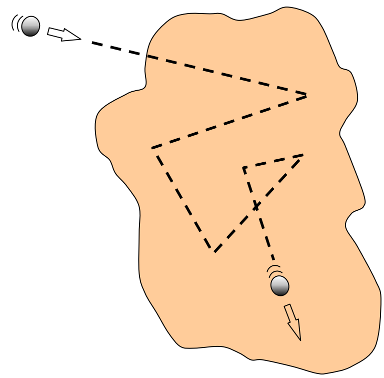
6.6. Equation of Transfer for Light

surface scattering



interface between materials

volume scattering



participating medium

6.6. Equation of Transfer for Light

- Conservation of energy: difference of radiative energy along light ray must be equal to difference between effects of emission and absorption

$$\{I(\mathbf{x},\mathbf{n},\nu) - I(\mathbf{x}+d\mathbf{x},\mathbf{n},\nu)\}\, dA\, d\Omega\, d\nu\, dt$$
$$= \{-\chi(\mathbf{x},\mathbf{n},\nu) I(\mathbf{x},\mathbf{n},\nu) + \eta(\mathbf{x},\mathbf{n},\nu)\}\, ds\, dA\, d\Omega\, d\nu\, dt$$

$\Rightarrow$

$$\boxed{\mathbf{n} \bullet \nabla I = -\chi I + \eta}$$

time-independent equation of transfer

6.6. Equation of Transfer for Light

- Note: scattering part contains I
- Writing out the equation of transfer:

$$\mathbf{n} \bullet \nabla I = -(\kappa + \sigma) I + q + \frac{1}{4\pi} \iint \sigma(\mathbf{x}, \mathbf{n}', \nu') p(\mathbf{x}, \mathbf{n}', \mathbf{n}, \nu', \nu) I(\mathbf{x}, \mathbf{n}', \nu') \, d\Omega' \, d\nu'$$

- Without frequency-dependency and without inelastic scattering:

$$\boxed{\mathbf{n} \bullet \nabla I = -(\kappa + \sigma) I + q + \frac{1}{4\pi} \int \sigma(\mathbf{x}, \mathbf{n}') p(\mathbf{x}, \mathbf{n}', \mathbf{n}) I(\mathbf{x}, \mathbf{n}') \, d\Omega'}$$

integro-differential equation: time-independent equation of transfer

6.6. Equation of Transfer for Light

- Boundary conditions for equation of transfer
  - Required for complete description of the problem
  - Explicit boundary condition (emission of light at boundary)
  - Implicit boundary condition (reflection)
- Combination of explicit and implicit boundary conditions on boundary surface:

$$I(\mathbf{x}, \mathbf{n}, \nu) = \underbrace{E(\mathbf{x}, \mathbf{n}, \nu)}_{\text{emission}} + \iint \underbrace{k(\mathbf{x}, \mathbf{n}', \mathbf{n}, \nu', \nu)}_{\text{surface scattering kernel}} \underbrace{I^{(\mathrm{in})}(\mathbf{x}, \mathbf{n}', \nu')}_{\text{incoming radiation}} \, d\Omega' \, d\nu'$$

6.6. Equation of Transfer for Light

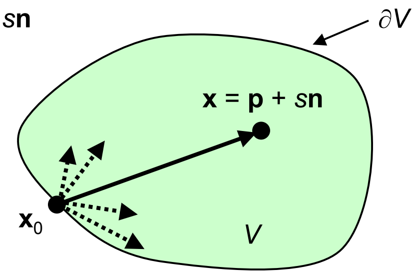
- Rewriting the equation of transfer

$$\mathbf{n} \bullet \nabla I(\mathbf{x},\mathbf{n},\nu) = -\chi(\mathbf{x},\mathbf{n},\nu) I(\mathbf{x},\mathbf{n},\nu) + \eta(\mathbf{x},\mathbf{n},\nu)$$

  yields

$$\frac{\partial}{\partial s} I(\mathbf{x},\mathbf{n},\nu) = -\chi(\mathbf{x},\mathbf{n},\nu) I(\mathbf{x},\mathbf{n},\nu) + \eta(\mathbf{x},\mathbf{n},\nu)$$

  for derivative along a line **x** = **p** + *s***n**
- Arbitrary reference point **p**



6.6. Equation of Transfer for Light

- Optical depth between 2 points $\mathbf{x}_1 = \mathbf{p} + s_1\mathbf{n}$ and $\mathbf{x}_2 = \mathbf{p} + s_2\mathbf{n}$ is

$$\tau_\nu(\mathbf{x}_1,\mathbf{x}_2) = \int_{s_1}^{s_2} \chi(\mathbf{p} + s'\mathbf{n},\mathbf{n},\nu)\, ds'$$

- Optical depth serves as integrating factor for equation of transfer:

$$\frac{\partial}{\partial s}\left(I(\mathbf{x},\mathbf{n},\nu) \cdot e^{\tau_\nu(\mathbf{x}_0,\mathbf{x})} \right) = \eta(\mathbf{x},\mathbf{n},\nu) \cdot e^{\tau_\nu(\mathbf{x}_0,\mathbf{x})}$$

  $\Rightarrow$

$$I(\mathbf{x},\mathbf{n},\nu) \cdot e^{\tau_\nu(\mathbf{x}_0,\mathbf{x})} - I(\mathbf{x}_0,\mathbf{n},\nu) = \int_{s_0}^{s} \eta(\mathbf{x}',\mathbf{n},\nu) \cdot e^{\tau_\nu(\mathbf{x}_0,\mathbf{x}')}\, ds'$$

  with $\mathbf{x}_0$ on the boundary surface

6.6. Equation of Transfer for Light

- Integral form of the equation of transfer

$$I(\mathbf{x},\mathbf{n},\nu) = I(\mathbf{x}_0,\mathbf{n},\nu)\cdot e^{-\tau_\nu(\mathbf{x}_0,\mathbf{x})} + \int_{s_0}^{s} \eta(\mathbf{x}',\mathbf{n},\nu)\cdot e^{-\tau_\nu(\mathbf{x}',\mathbf{x})}\, ds'$$

- Integral equation because η contains *I*
- Interpretation: Radiation consists of
  - Sum of photons emitted from all points along the line segment,
  - Attenuated by the integrated absorptivity of the intervening medium, and
  - Additional, attenuated contribution from radiation entering the boundary surface

6.6. Equation of Transfer for Light

- Special case: vacuum condition
  - No emission, absorption, or scattering
  - Except on surfaces
  - Frequency-dependency is usually neglected (no inelastic effects)

  - Equation of transfer (inside the volume) is greatly simplified:

$$I(\mathbf{x},\mathbf{n}) = I(\mathbf{x}_0,\mathbf{n})$$

  - Rays incident on surface at **x** are traced back to some other surface element at **x'**:

$$I^{(\mathrm{in})}(\mathbf{x},\mathbf{n}') = I(\mathbf{x}',\mathbf{n}')$$

6.6. Equation of Transfer for Light

- Special case: vacuum condition *(cont.)*
  - Generic boundary condition

$$I(\mathbf{x},\mathbf{n},\nu) = E(\mathbf{x},\mathbf{n},\nu) + \iint k(\mathbf{x},\mathbf{n}',\mathbf{n},\nu',\nu)\, I^{(\mathrm{in})}(\mathbf{x},\mathbf{n}',\nu')\, d\Omega'\, d\nu'$$

  becomes

$$\boxed{I(\mathbf{x},\mathbf{n}) = E(\mathbf{x},\mathbf{n}) + \int k(\mathbf{x},\mathbf{n}',\mathbf{n})\, I(\mathbf{x}',\mathbf{n}')\, d\Omega' \quad , \quad \mathbf{x} \in S}$$

  - Rendering equation [Kajiya 1986]
  - Standard form via BRDF (bidirectional reflection distribution function)

$$k(\mathbf{x},\mathbf{n}',\mathbf{n}) = f_r(\mathbf{x},\mathbf{n}',\mathbf{n}) \cos\theta_i$$

6.6. Equation of Transfer for Light

- Special case for most volume rendering approaches:
  - Emission-absorption model
  - Density-emitter model [Sabella 1988]
  - Volume filled with light-emitting particles
  - Particles described by density function
- Simplifications:
  - No scattering
  - Emission coefficient consists of source term only: $\eta = q$
  - Absorption coefficient consists of true absorption only: $\chi = \kappa$
  - No mixing between frequencies (no inelastic effects)

6.6. Equation of Transfer for Light

- Volume rendering equation

$$\boxed{I(s) = I(s_0) \cdot e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(s')e^{-\tau(s',s)}\, ds'}$$

with optical depth

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s')\, ds'$$

6.6. Equation of Transfer for Light

- Discretization of volume rendering equation
  - Discrete steps s_k
  - Often equidistant

$$I(s_k) = I(s_{k-1})e^{-\tau(s_{k-1},s_k)} + \int_{s_{k-1}}^{s_k} q(s)e^{-\tau(s,s_k)}\, ds$$

6.6. Equation of Transfer for Light

- Discretization of volume rendering equation *(cont.)*
- Define:
  - Transparency part $\theta_k = e^{-\tau(s_{k-1}, s_k)}$
  - Emission part $b_k = \int_{s_{k-1}}^{s_k} q(s) e^{-\tau(s, s_k)} ds$
- Discretized volume integral:

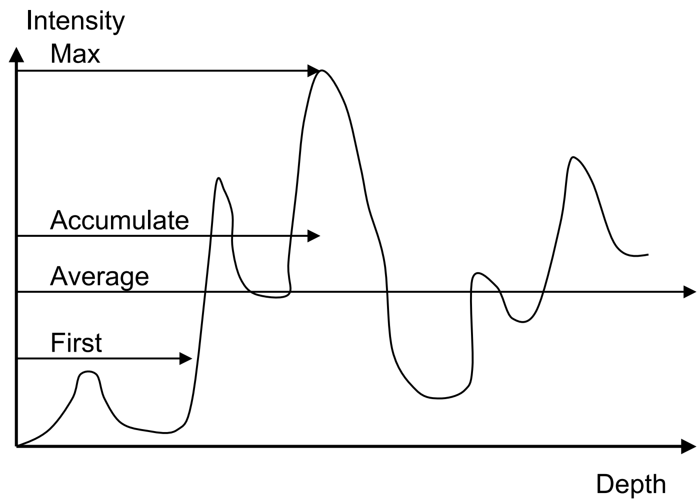
$$\boxed{\begin{aligned} I(s_n) &= I(s_{n-1})\theta_n + b_n = \sum_{k=0}^{n} \left(b_k \prod_{j=k+1}^{n} \theta_j \right) \\ &= I(s_{n-1}) \cdot (1 - \alpha_n) + b_n \end{aligned}}$$

over operator
with opacity

6.7. Compositing Schemes

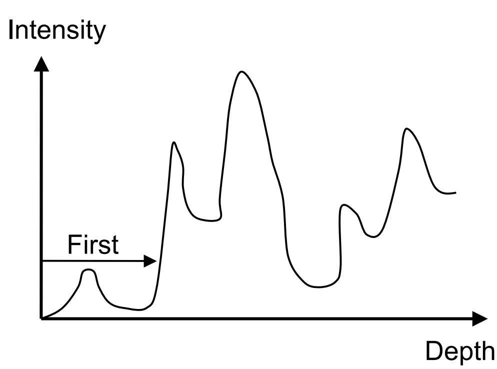
- Variations of composition schemes
  - First
  - Average
  - Maximum intensity projection
  - Accumulate

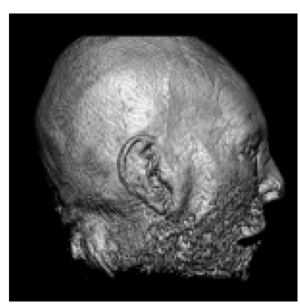
6.7. Compositing Schemes



6.7. Compositing Schemes

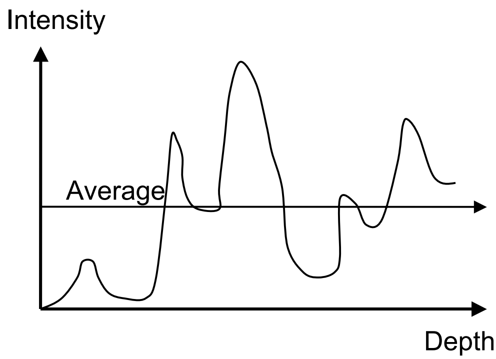
- Compositing: First
- Extracts isosurfaces

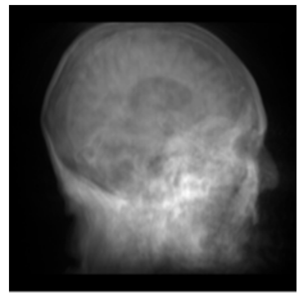




6.7. Compositing Schemes

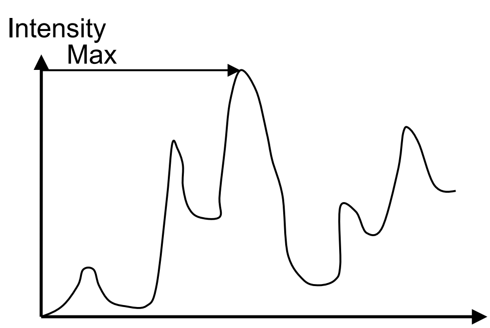
- Compositing: Average
- Produces basically an X-ray picture

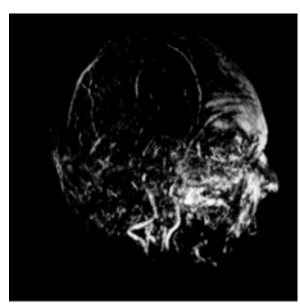




6.7. Compositing Schemes

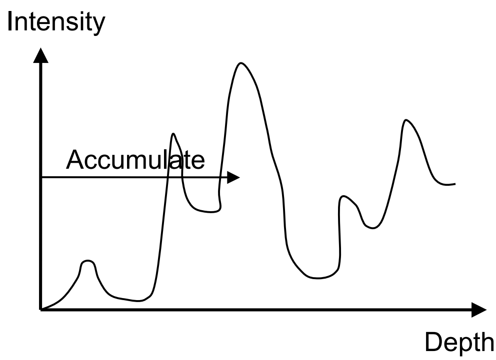
- Maximum Intensity Projection (MIP)
- Often used for magnetic resonance angiograms
- Good to extract vessel structures

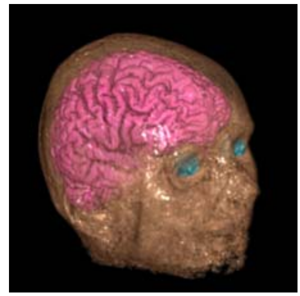




6.7. Compositing Schemes

- Compositing: Accumulate
- Emission-absorption model
- Make transparent layers visible (see volume classification)





6.8. What Else?

- Non-uniform grids:
  - Resampling approaches, adaptive mesh refinement (AMR)
  - Cell projection for unstructured (tetrahedral) grids
  - Shirley-Tuchman [1990]