

## CITS4241 Visualisation Lectures 8 & 9

### Volume Rendering (cont.)

## Volume Rendering (Part I) - Revision

- Visualising scalar quantities
- Colour mapping
  - Use a look up table
- Contouring
  - Choose contour values
    - *E.g.* interface between rock types or rock and liquid in a geological dataset
  - 2D: Contour: Line of equal values
  - 3D: Iso-surface: Surface of equal values

## Volume Rendering (Part I) - Revision

- Efficient contour determination
  - Use allowed cell topologies
    - relation to each vertex
    - topology index
    - intersecting edges
- Marching squares & cubes
  - Ambiguous cases
    - Can't determine where contour really lies
    - 2D: doesn't matter
    - 3D: there are ways to disambiguate



0101



1010

## Overview

- Volume rendering and associated algorithms
- (Briefly) A method to accelerate volume rendering

## Big picture

- **Scientific Visualisation 3D Methods**
  - **surface rendering**
  - **volume rendering**
  - **volume graphics**
  - **specialised methods for non-scalar data**
- **surface vs volume rendering**

## Surface Rendering vs Volume Rendering

Surface rendering	Volume rendering
Sensitive to scene and object complexity	Insensitive to scene and object complexity
Can only render external surface	Can render inner structures as well as surfaces
Difficult; must be performed analytically	Allow voxel aggregation, octrees etc
Measurements (eg. Distance, area, normal) are analytical, but may be complex	Discrete approximation, but simple
Memory and processing requirement are variable – depend on scene and object complexity	Large but constant
Object-space aliasing? None	Frequent
Transformation is continuous; performed on geometric definitions of objects	Transformation is discrete; performed on voxel subvolumes

## Volume Rendering: Data Sources

- **Medicine**
  - **Computer Tomography (CT),**
  - **Magnetic Resonance Imaging (MRI),**
  - **3D Ultrasound (3D US),**
  - **Positron Emission Tomography (PET),**
  - **confocal microscopy and others**
- **Geology**
- **Engineering**
- **Physics**
- **Meteorology**
- **Oceanography**
- **Finance etc.**

## Volume Rendering

- **Powerful technique for directly viewing 3D datasets**
- **Can be classified into 2 categories:**
  - **Object-order techniques, e.g.**
    - marching cubes
    - Splatting – each voxel gets “splatted” on the image
  - **Image-order techniques, e.g.**
    - ray casting – from each image pixel cast a ray into volume

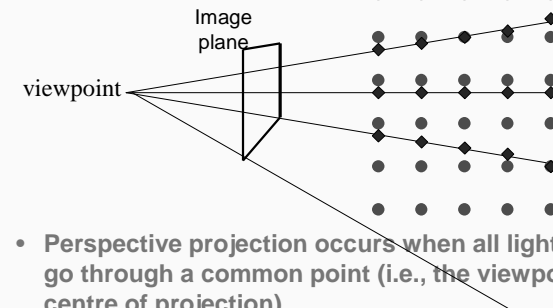
**Some techniques are combination of the two; some others do not fall into either category**
- **specialised accelerations**
  - **shear-warp method,**
  - **Fourier rendering,**
  - **Wavelet rendering,**
  - **parallel implementations.**

## Ray casting - Overview

Given a volume of scalar data, want to compute each pixel value in the image by sending a ray into the volume data:

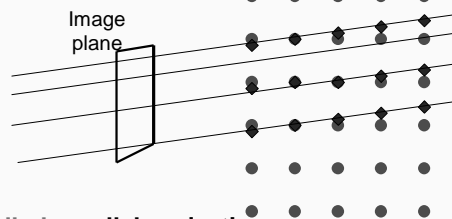
- the 3D to 2D projection can be **orthographic** or **perspective**
- rotate and translate volume *wrt* image plane
- get regions of interest
  - clip out don't care data
- need to classify volume
  - thresholds, materials, "transfer functions"
- compute Z-buffer
- for each image pixel: store in buffer result of the **compositing** ray through volume, including lighting model applied using inferred gradient
- display image

## Perspective projection



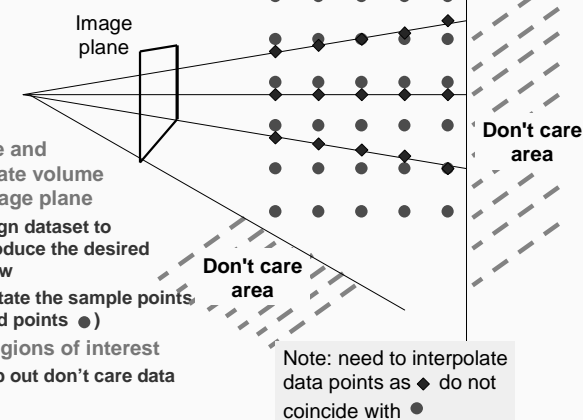
- Perspective projection occurs when all light rays go through a common point (i.e., the viewpoint or centre of projection)
- Perspective projection gives foreshortening effect

## Orthographic projection



- Also called **parallel projection**
- Orthographic projection occurs when all light rays are parallel (i.e., the viewpoint or centre of projection is at infinity)
- Orthographic projection gives no foreshortening effect (parallelism in 3D is preserved in 2D)

## Ray casting



- Rotate and translate volume *wrt* image plane
  - Align dataset to produce the desired view
  - Rotate the sample points (red points ●)
- Get regions of interest
  - clip out don't care data

Note: need to interpolate data points as ◆ do not coincide with ●

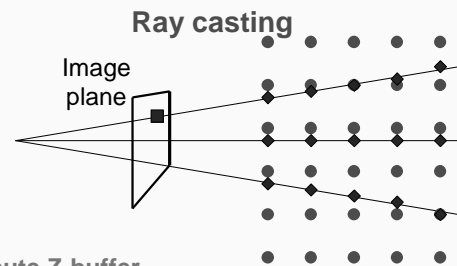
## Volume Classification

- Aim: Classify the relevant objects of interest within a dataset - Want to make “interesting features” stand out
- Need to apply a “transfer function” to assign opacity and/or colour to any point on the casting ray. Can either
  - Interpolate scalar values at grid (sampled) points, then evaluate transfer function, or
  - Evaluate transfer function at grid points then interpolate (à more accurate)
- Transfer functions can
  - make regions transparent
    - Map values to opacity = 0 (transparent)
  - select regions of interest
    - eg*  $a < V < b$      $\delta$  opacity = 1
    - $V < a$  or  $V > b$      $\delta$  opacity = 0
    - Highlights a material (*eg* bone), region of interest, ...
    - Change a, b to select a different material, ...

## Volume Classification - Transfer functions

- Transfer functions can be based on
    - Scalar values of data points alone, or
    - Scalar values *and* gradient magnitudes of data points
- The latter gives better results

*eg* The transfer function introduced in Levoy's 1988 paper



- Compute Z-buffer
- For each pixel in image
  - cast ray into volume
  - store in buffer result of compositing ray through image (including lighting model applied using inferred gradient)
- Display image

## Ray casting: traversing volume

- Alpha compositing
  - Opacities:  $\alpha = 1$  à opaque;  $\alpha = 0$  à transparent;
  - Colours: red, green, blue components
- Possible strategies
  - Maximum Intensity Projection (MIP)
    - Simple, fast
    - Effective for viewing isolated structures  
*eg* veins and arteries in an angiogram
  - Average value
  - Isosurface (threshold)
  - Distance to threshold  
*and*
  - any (sensible) combination of above

### Theoretical Model

- Physics based formula
  - low albedo = fraction of illumination reflected  
fr Latin albus = white

$$I(t_0, t_L) = \int_{t_0}^{t_L} q(\mathbf{x} + t\mathbf{s}) e^{-\int_{t_0}^t \alpha(u) du} dt$$

Light intensity,  $I$ , of a ray that traverses the volume from point  $t_0$  to  $t_L$  in the direction  $s$  with

$q(t)$  as the local illumination model function  
 $\alpha(t)$  is the opacity or absorption coefficient

### Practical model

- Discretise the physical model
- Total intensity accumulated on a single ray:

$$\sum_{i=0}^n \left( I_i \alpha_i \prod_{j=0}^{i-1} (1 - \alpha_j) \right) = I_0 \alpha_0 + I_1 \alpha_1 (1 - \alpha_0) + I_2 \alpha_2 (1 - \alpha_0)(1 - \alpha_1) + \dots$$

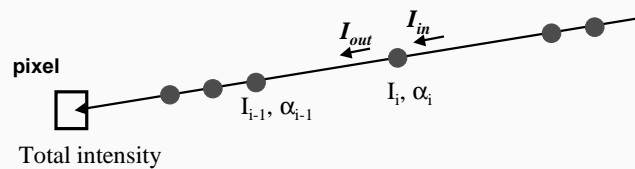
- $I(i)$  = intensity of  $i^{th}$  sample point on the ray
- $1 - \alpha_i = 1 - \text{opacity} = \text{transparency}$  of  $i^{th}$  sample point
- Intuitively ...
  - Intensity due to point  $i$  is
    - $I_i$  multiplied by product of all transparencies before it:  $j = 0$  to  $j = i - 1$
  - Sum all these intensities

### Practical model (cont.)

$$\sum_{i=0}^n \left( I_i \alpha_i \prod_{j=0}^{i-1} (1 - \alpha_j) \right) = I_0 \alpha_0 + I_1 \alpha_1 (1 - \alpha_0) + I_2 \alpha_2 (1 - \alpha_0)(1 - \alpha_1) + \dots$$

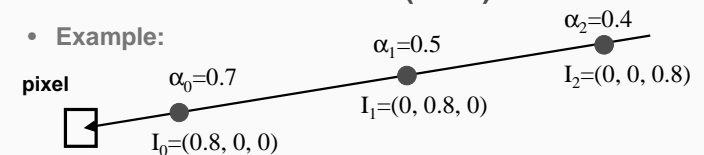
- Intensity (or colour) contributed to ray by point  $i$  is:  
 $I[i] * \alpha[i] * (1 - \alpha[0]) * (1 - \alpha[1]) * \dots * (1 - \alpha[i-1])$

Diagrammatically



### Practical model (cont.)

- Example:



Resultant colour=?

Resultant colour at pixel =

$$0.7 * (0.8, 0, 0) + 0.5 * (0, 0.8, 0) * 0.3 + 0.4 * (0, 0, 0.8) * 0.3 * 0.5 = (0.56, 0, 0) + (0, 0.12, 0) + (0, 0, 0.048) = (0.56, 0.12, 0.048)$$

### Practical model (cont.)

$$\sum_{i=0}^n \left( I_i \alpha_i \prod_{j=0}^{i-1} (1 - \alpha_j) \right) = I_0 \alpha_0 + I_1 \alpha_1 (1 - \alpha_0) + I_2 \alpha_2 (1 - \alpha_0)(1 - \alpha_1) + \dots$$

- **Front-to-back algorithm**

```

trans = 1.0; inten = 0;
for (i=0; i <= n-1; i++) {
    inten = inten + trans*alpha[i]*I[i];
    trans = trans * (1-alpha[i]);
    if (trans == 0) break;
}
    
```

- **Note:** need to keep track of transparency
- **Advantage:** Can terminate when `trans == 0.0`
- **Disadvantage:** more housekeeping computation

### Practical model (cont.)

- **Back-to-front algorithm**

```

inten = 0;
for (i=n-1; i >= 0; i--) {
    inten = I[i] * alpha[i] + inten * (1 - alpha[i]);
}
    
```

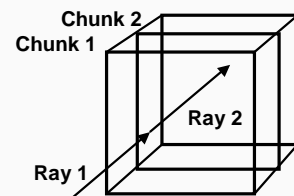
- **Note:**

- **Compute composite intensity of points from back (furthest away from viewpoint) to front**
- **Does not need to keep track of transparency**
- **Does not allow early termination of loop**

### Partial Ray Composition

- Both algorithms allow *partial ray composition*, *ie* chopping a ray in pieces and treat each one as a separate ray
- Useful if you want to work in parallel

- Ray 1 and Ray 2 can be calculated independently  
∴ *in parallel*
- Result is sum from both rays



### Lighting model

- **Gouraud and Phong Shading**
  - Interpolate colours or normals
- **Ambient light**
  - Light with same intensity everywhere
  - Non-directional
  - Doesn't depend on the angle of the light wrt to surface

$$C_o = k_a C_a O_d$$

where

- $C_o$  = resulting colour
- $k_a$  = ambient reflection coefficient :  $0 \leq k_a \leq 1$
- $C_a$  = colour of ambient light
- $O_d$  = diffuse colour of point on object
- Apply for each of the R, G, B components

## Lighting model

- Diffuse reflection
  - Point light source
  - Radiates uniformly in all directions
  - Colour of a point depends on distance and surface orientation
    - Often distance ignored (= light infinitely far away)

$$C_o = \text{ambient} + k_d C_p O_d \cos \theta$$

where

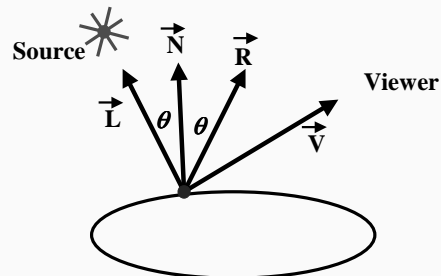
- $k_d$  = diffuse reflection coefficient :  $0 \leq k_d \leq 1$
- $C_p$  = colour of point light source
- $O_d$  = diffuse colour of point on object
- $\theta$  = angle between light source vector and surface normal =  $L \cdot N$

## Lighting model (cont.)

- Specular reflection
  - Highlights due to light source
    - ∅ Shiny white spots
  - Localised
  - Intensity falls off sharply around reflection point
  - Depends on view point
  - $C_o = \text{ambient} + \text{diffuse} + k_s C_p O_s (R \cdot V)^n$ 
    - where
      - $k_s$  = specular reflection coefficient :  $0 \leq k_s \leq 1$
      - $C_p$  = intensity of the point light source
      - $O_s$  = specular colour (sometimes white)
      - $R$  = reflection vector
      - $V$  = direction to viewer
      - $n$  = specular reflection exponent ( $\delta$  sharp cutoff)

## Lighting model (cont.)

- Specular reflection



## Final Remarks

- Volume rendering has evolved from static views to interactive visualisation
- Now possible with 2D and 3D hardware texture-capable graphics accelerators
- Temporal (time sequence) visualisation is now possible and is the way forward in medical visualisation
- Multiresolution techniques provide efficient solutions for problems in volume rendering and other visualisation algorithms

Further reading:

W. Schroeder, K. Martin, and B. Lorensen: "The Visualization Toolkit – An Object-Oriented Approach to 3D Graphics", 2<sup>nd</sup> Edition, Prentice Hall, 1998.

Sections 7.1, 7.3-7.7, 7.9, 7.11-7.13, 7.19