Direct Volume Rendering

Lecture 9

The story so far…

- So far we have only examined 2D data slice and surface visualisation techniques.
- Even when dealing with 3D data we have reduced it to a 2D (surface) problem.
- Now we consider viewing volume data without constructing “artifical” surfaces/meshes from spatial information and data values.

Direct Volume Rendering (DVR)

- This is a significant sub-branch of Visualisation
- The science of projecting a 3D (or higher) volume into 2D to explore its internal structure.
- Benefits areas in which rigid thin surfaces are a bad representation.
  - Eg amorphous materials, slowly varying density.

Direct Volume Rendering ...

- Key differences from previous surface techniques include:
  - No surface mesh is constructed
  - Entire volume (ie subsurface data) may contribute to appearance
  - Computationally intensive
  - Essentially a 2D rendering process
  - Each position adjustment requires complete recalculation

Data Classification

- Using previous techniques data was identified and classified by creating geometric surfaces and iso-surfaces etc.
- With DVR all the data is available.
- To help identify different portions, colour is used to classify data based on the values.
  - Eg High density might be bone and lower values tissue in a CAT scan.
- Data value based transparency is used to hide less interesting regions.

Voxels

- 3D analogue to a pixel. A voxel is the smallest unit of a data volume.
- Just as a pixel contains one data value (shown as colour), so a voxel contains (typically) one data value (also shown as colour).
- Voxels are usually hexahedrons, typically cubes.

DVR Techniques

- A range of DVR algorithms exist including:
  - Projection Methods (Object Order)
    - V-Buffer
    - Splatting
  - Image Order Methods
    - Ray-casting
    - Sabella
    - Cell integration
- **Note**: There is a non DVR technique (technically a surface algorithm using opaque voxels, called *Cuberille*)

DVR Techniques…

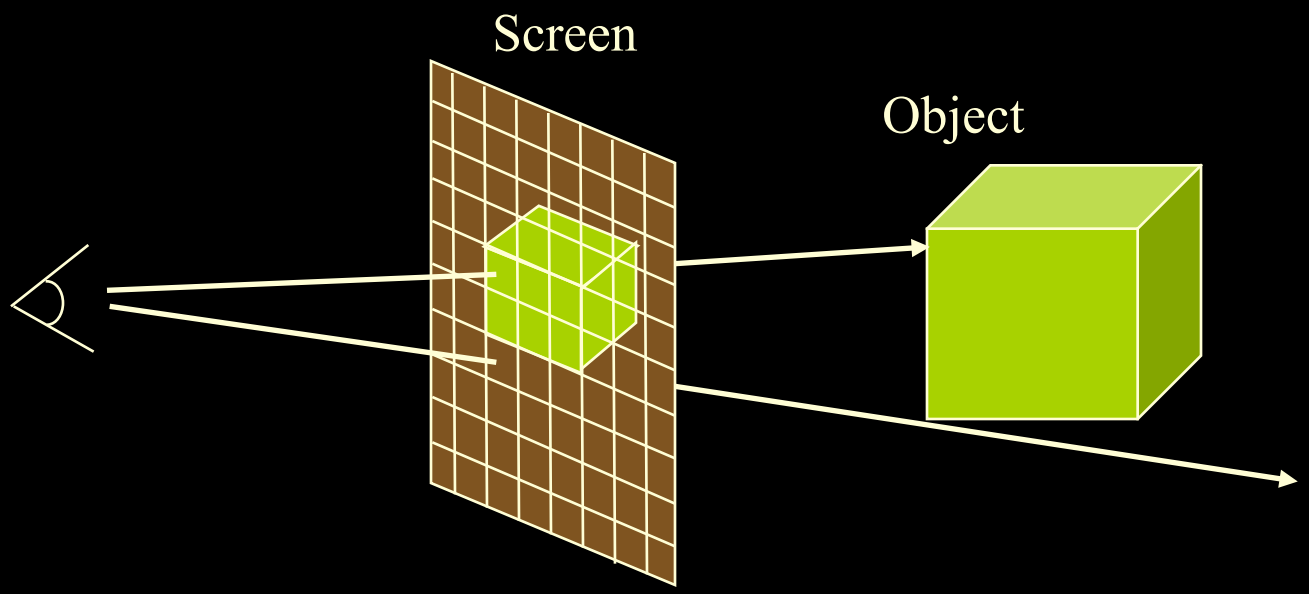
- Two broad ways to treat data:
  - **voxel based** - cube of data is constant valued (or simple weight function from central value to edge)
  - **cell based** - data points at cell vertices. Interpolate to obtain value
- Images generally created using one of the following algorithm categories:
  - **Image-order** pixel traversal (ray casting)
  - **Object-order** voxel traversal

Ray Casting

- Image order traversal
  - Imagine looking at a digitized photograph. Each pixel has a particular colour based on the colour of light that entered the camera at that point from the object being photographed.
  - Imagine looking at an object through a sheet of glass with a very fine grid on it. Each grid cell has a colour based on the ray of light that passes though that grid cell (pixel) into your eye.

Ray Casting

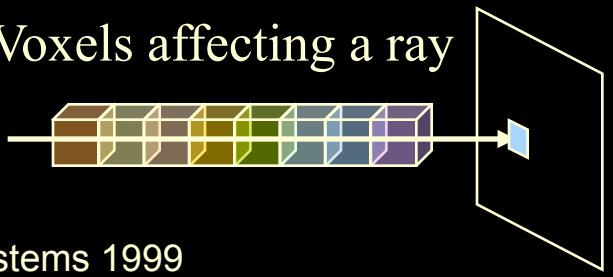
- Reverse the direction and cast a ray through each pixel
- Find colour where it intercepts an object
- Colour pixel appropriately
- Note: Different to Ray Tracing, nor reflections/shadows!



Viewing Volume Internals

- We don't actually stop casting the ray when we hit the first voxel. Instead the ray passes though the whole volume.
- Calculate the actual colour to fill the pixel by working out the contribution of each voxel along the ray's path.
- If the voxels are made semi-transparent then internal voxel will contribute to the colour.

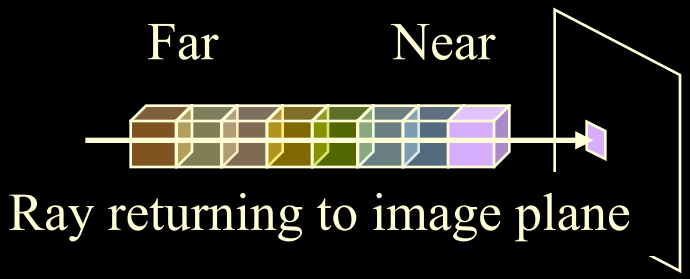
Voxels affecting a ray



Ray accumulates colours based on opacity as it returns through the volume to the image plane/eye

Voxel Colour and Transparency

- A 2D array of scalar data values may be turned into a colour image by using a colourmap (LUT).
- Similarly a voxel may be coloured using a colourmap. In this case however each voxel has colour and *transparency*.
- An opaque voxel will obscure any voxels behind it in ray order.
- Voxels may also be shaded based on (central difference) gradient calculations



Interpolation Issues

- Ray lines typically do not pass though voxel centres.
- Interpolation between voxels values is often done.
- Since the internal structure is not known mathematically, care must be taken with interpolation.
- For example, a bone tissue interface is discontinuous. Interpolation here would create a smoother transition.
- Simple tri-linear interpolation is usually the safest.

Object Order Traversal

- Calculation moves though the volume a voxel at a time.
- Calculates the projection onto screen of each voxel and adds its contribution into the image.
- Object order algorithms include the V-Buffer and Splatting methods.
- Object Order Traversal may proceed though the volume
  - *Front-to-Back* or
  - *Back-to-Front*

Splatting

- Based on the science of throwing snowballs at windows.
- Each snowball will “splat” on a window with a solid central impact region and snow thinning towards the edge of the splat region.
- Each voxel ejects a coloured snowball projectile to splat on the viewing plane.
- The faces/voxel slices closest to the viewing plane are splatted first. The general splatting order being based on voxel distance from the viewing plane.

Splatting…

- Thus closest voxel splats are in front of (and tend to obscure) splats from voxels further back. (Remember we are painting on the back of a window so the first layer painted dominates.)
- Once colour, transparency and shading have been determined the next step is to draw the splat on the image plane.

Splatting Kernels

- A *reconstruction kernel* is used to calculate the contribution of the splat to each pixel in the image plane, after which the incoming splat pixel is *alpha blended* with the existing pixel's colour *and transparency/opacity*.
- Once the opacity has reached 1.0 at any pixel, further splats on that pixel have no effect.
- The projection of the kernel onto the image plane is termed the *footprint*.
- Scaling the footprint appropriately makes the volume fill the desired image size.

Splatting Viewing Projections

- For an orthogonal image projection, the reconstruction kernel is typically a simple circularly symmetric gaussian.
- In perspective mode however an elliptical kernel is required and furthermore is voxel position dependent.
- As a consequence of the extra complexity in perspective mode, many software programs only implement orthogonal projection mode.

DVR CPU Requirements

- Key points:
  - Computationally Intensive
  - CPU speed of key importance
  - Fairly memory hungry
  - Graphics hardware generally of little benefit

    Some DVR software may use accelerated texure memory.

Some Representative Timings

- To give an indication of just how intensive the process is here are a few timings taken from the Splatting program produces by San Diego Super Computer Center
- 256 x 256 x 90 volume of the brain (12Mb)
- Output: 200x200 pixel image

  - SGI Crimson 1 CPU ~15 minutes
  - CRAY YMP 8 CPU ~3 minutes
  - nCUBE 32 CPU ~2 minutes

Interactive Rendering Speeds

- Systems which try to achieve near real-time rendering performance use super-computers or computer clusters to perform the calculations and send the results to a workstation for display.
- Some of the algorithms are particularly suited to parallelization, where many CPUs each work on part of the image or part of the dataset.

Progressive Refinement

- Due to the long times for even relatively small datasets, some of the slower techniques display a low resolution version of the dataset first (fewer rays) and then subdivide onto a progressively finer grid.
- This technique is termed *progressive refinement*.

So what does it look like ?

One example...