Manchester Computing

UKHEC

High-End Computing

UKHEC Technology Watch Report

Trends in Volume Visualisation of Large Data Sets

Nigel W. John, W. T. Hewitt, Jo Leng
Supercomputing, Visualisation, and e-Science

THE UNIVERSITY of MANCHESTER

MVC

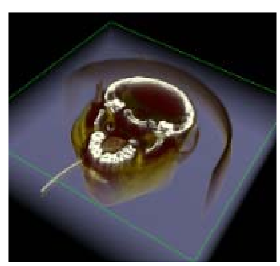
Contents

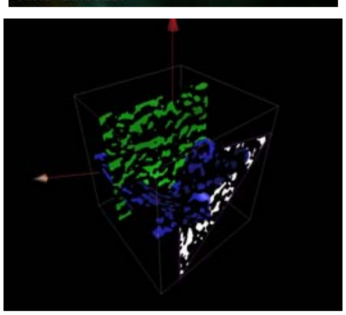
Manchester Computing

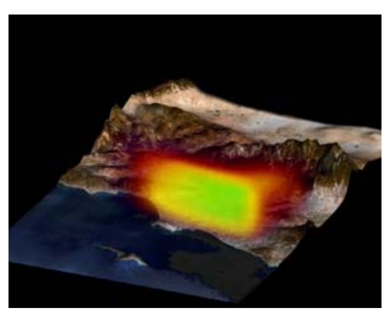
Introduction

THE UNIVERSITY of MANCHESTER

Why Volume Visualization?

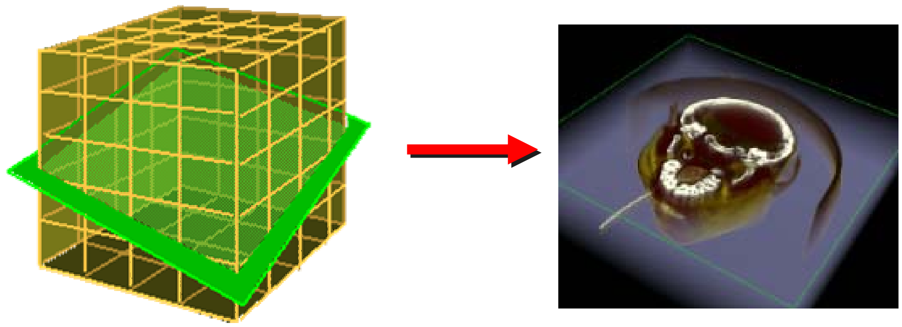






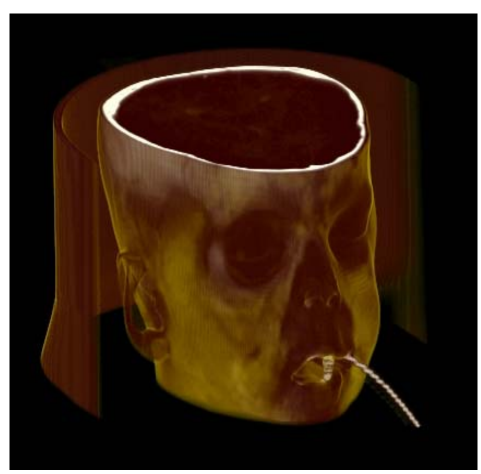
Why Volume Visualisation?

- The representation and analysis of volume data
- See internal structure/topology



Application Areas

- Medical Data
  - CT, MRI, Ultrasound, SPECT, PET, …
- Geosciences
- Environmental
- Scientific Visualisation
- Astronomy
- Confocal Microscopy
- Computational Simulation
  - CFD, FEA, ...



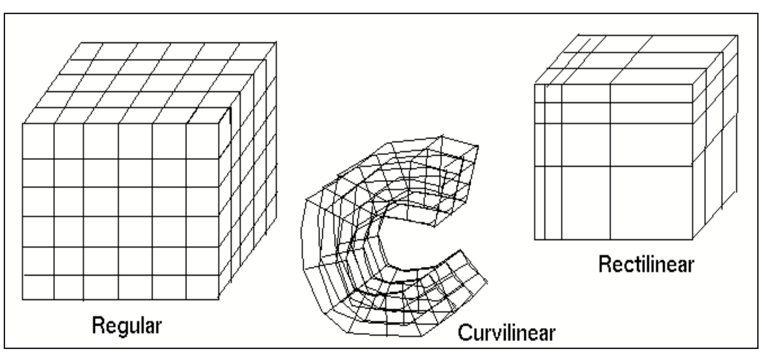
Manchester Computing

Terminology

THE UNIVERSITY *of* MANCHESTER

What is a Volume?

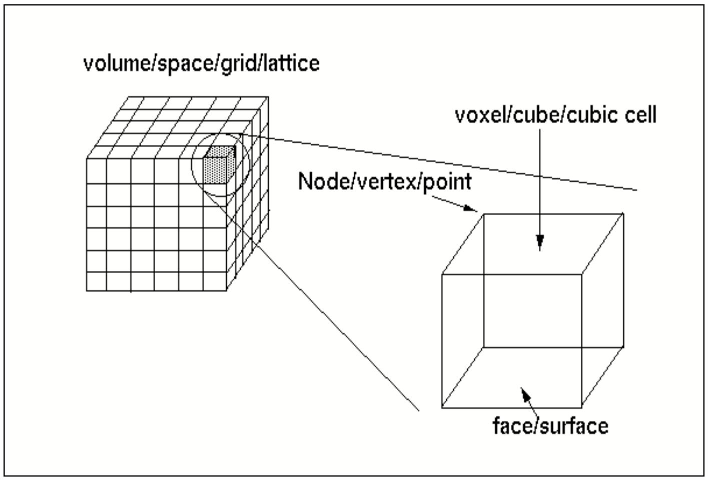
In General 3D Scalar Fields



- Regular
- Rectilinear
- Curvilinear
- Block structured
- Unstructured
- Hybrid

Voxels

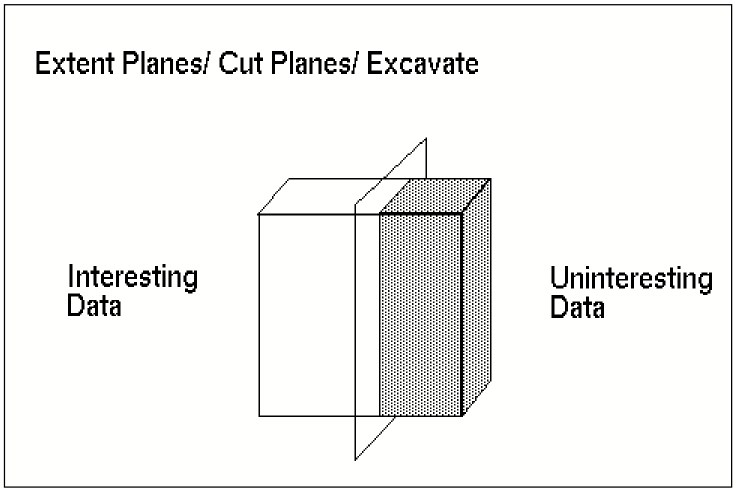
- Voxel
  - Corresponding to the *pixels* (picture elements) of a 2D image, volume elements are called *voxels* (volume elements).
- Partial Volume Effect
  - Feature may only occupy part of the voxel



Data Structures

- Binary voxel-model
  - either 1 (object) or 0 (no object)
- Grey Level voxel-model
- Generalised voxel-model
  - Intensity
  - Attributes e.g. member of
- Intelligent Volumes

Slicing



Voxelisation

- Convert geometry into a voxel model
- Spatial occupancy enumeration algorithms
  - Limited by resolution of the grid
- Distance Fields
  - The Euclidean distance to an object is calculated and stored for each discrete voxel
  - Provides smoother representation
  - Can be computationally expensive, optimisations exist

A distance field data set *D* representing a surface *S* is defined as:

$$D(p) = \mathrm{sgn}(p) \bullet \min\{|p-q| : q \notin S\}$$

$$\mathrm{sgn}(p) = \begin{cases} -1 & \text{if } p \text{ inside} \\ +1 & \text{if } p \text{ outside} \end{cases}$$

Where || is the Euclidean norm

The original surface (S) can be rendered using the level 0 distance field, S={q: D(q)=0}

Manchester Computing

Basic Techniques

Visualisation Model

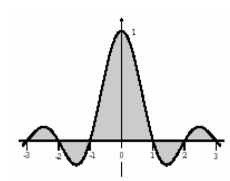
Haber & McNabb Reference Model

- Data Enhancement
  - Usually an interpolation
- Mapping
  - Geometric interpretation
- Rendering
  - Use standard computer graphics techniques to produce image

Resampling Methods: Sinc filter

- Optimum resampling is obtained from:

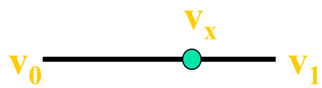
$$sinc(x) = \frac{\sin(\pi x)}{\pi x}$$



- Accuracy and efficiency depend on the size of the kernel
- Note - has unlimited extent
- In practice, simpler reconstruction filters are used

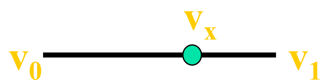
Simple Resampling Methods

Nearest Neighbour



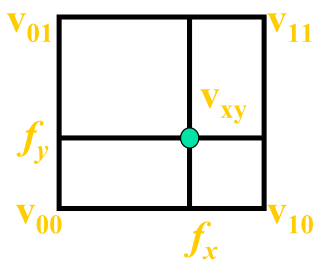
$$v_x = v_1$$

Linear Interpolation



$$v_x = (1 - f_x) v_0 + f_x v_1$$

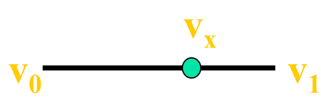
Bilinear Interpolation



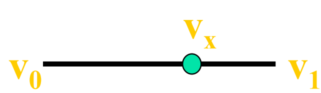
$$v_{xy} = (1 - f_x)(1 - f_y) v_{00} + (1 - f_x) f_y v_{01} + f_x (1 - f_y) v_{10} + f_x f_y v_{11}$$

Simple Resampling Methods

Nearest Neighbour

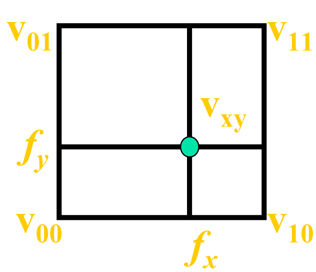


Linear Interpolation



$$v_x = v_0 + f_x (v_1 - v_0)$$

Bilinear Interpolation

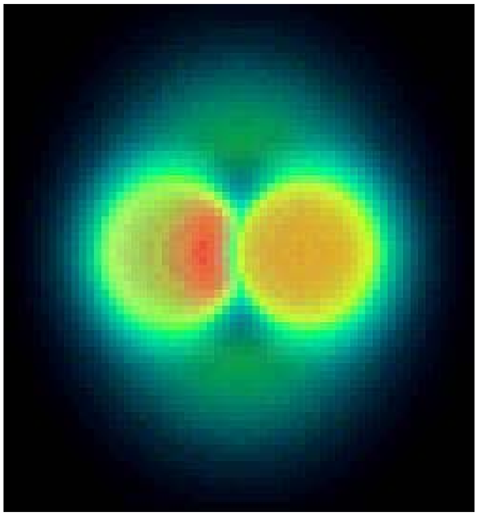


$$v_{x0} = v_{00} + f_x (v_{10} - v_{00})$$

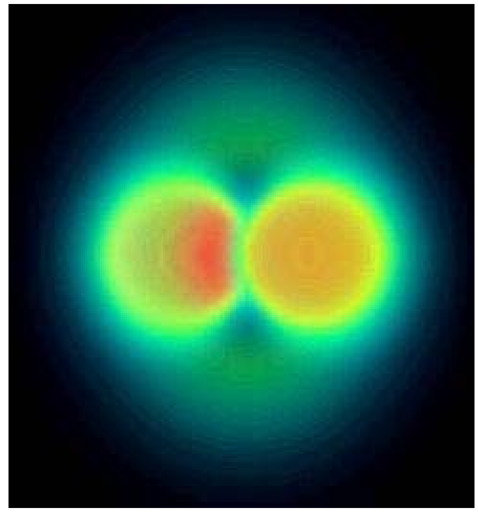
$$v_{x1} = v_{01} + f_x (v_{11} - v_{01})$$

$$v_{xy} = v_{x0} + f_y (v_{x1} - v_{x0})$$

Example



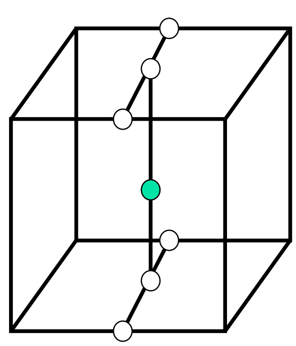
Nearest Neighbour

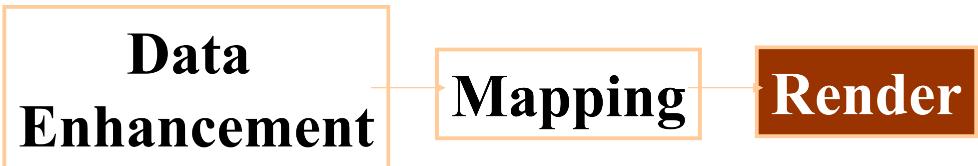


Bilinear Interpolation

Trilinear Interpolation

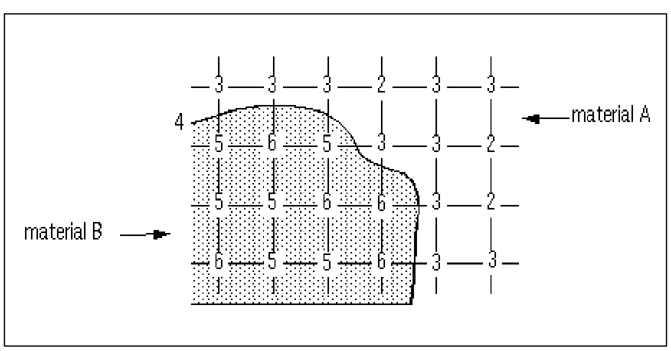
- Often performed several times for each voxel
- Involves cascading seven linear interpolations





Gradient Approximations

- Gradient of the data set is the rate of change in density values
  - Approximate surface normals, needed for the lighting model
  - Determine surface "strength"



Central Differences

$$G_x(i,j,k) = \frac{f(i+1,j,k) - f(i-1,j,k)}{\Delta x}$$

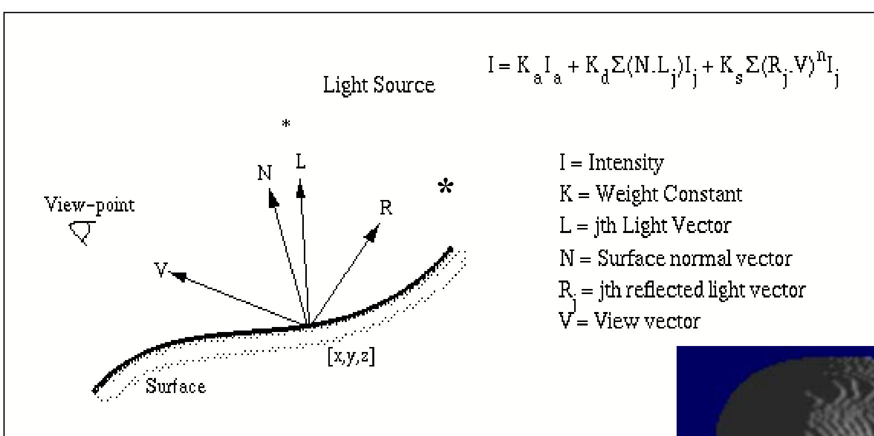
$$G_y(i,j,k) = \frac{f(i,j+1,k) - f(i,j-1,k)}{\Delta y}$$

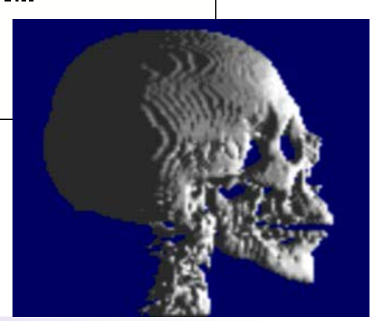
$$G_z(i,j,k) = \frac{f(i,j,k+1) - f(i,j,k-1)}{\Delta z}$$

- Surface Normal is then given by

$$\vec{N} = \frac{\vec{G}}{|\vec{G}|}$$

Lighting and Shading





Manchester Computing

Key Algorithms

THE UNIVERSITY of MANCHESTER

MVC

Data Enhancement → **Mapping** → Render

THE UNIVERSITY of MANCHESTER

Basic Mapping Algorithms

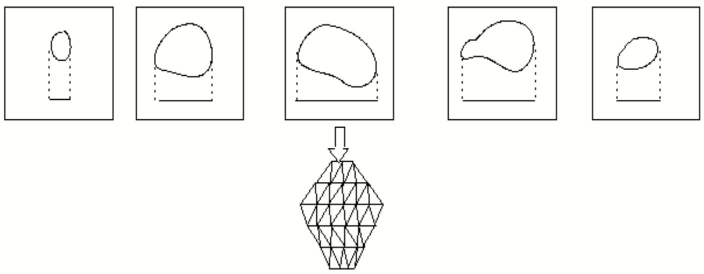
- Surface Extraction
  - Extract single value (iso-) surface from volume
  - Convert to geometric primitives (triangles)
- Direct Volume Rendering
  - Render without extracting geometry
  - Generally CPU intensive

Surface Extraction Methods

- Contour Tracking
- Marching Cubes
- Dividing Cubes
- Marching Tetrahedra
- Point Rendering

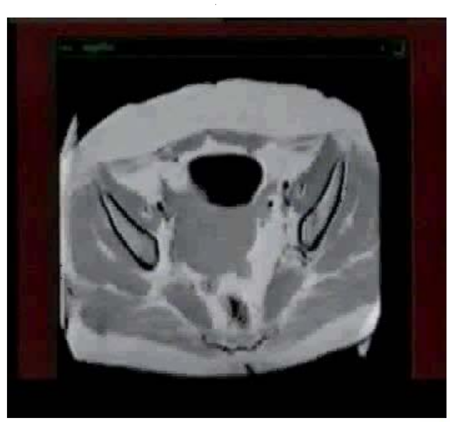
Contour Tracking

- Trace isovalued contour on 2D slice
- Construct triangular mesh between slices
- Render the triangles

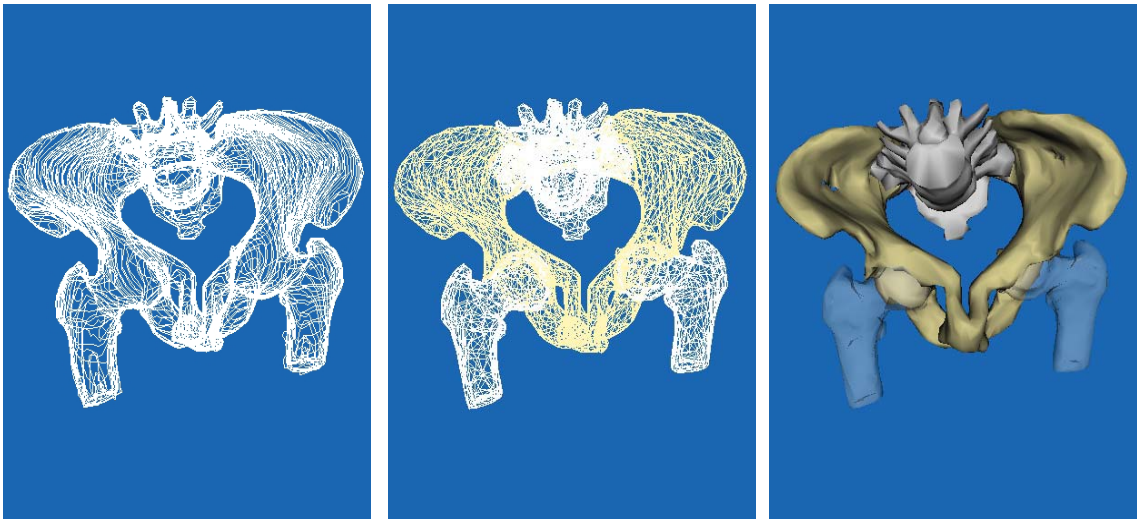


Contour Tracking

- Inaccurate
- Handles small features poorly
- Can't handle branching
- Requires binary Classification
- BUT Quick to Render



Example



Marching Cubes

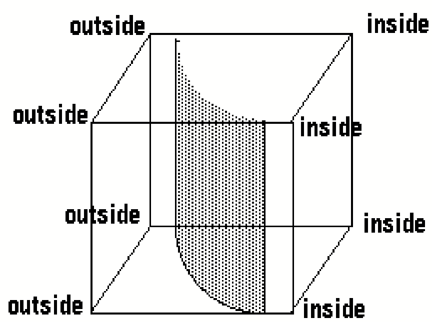
- For each voxel the surface passes through:
  – create small polygons to approximate the surface
- Surface intersects edges whose vertices bracket the surface value
- Assumptions:
  – maximum of one intersection per edge
  – maximum of four triangles per voxel

Algorithm

- Classify each grid point as being inside / outside the surface
- Build index for each voxel
- Create table of intersecting edges
- Interpolate intersection points
- Calculate & interpolate surface normals

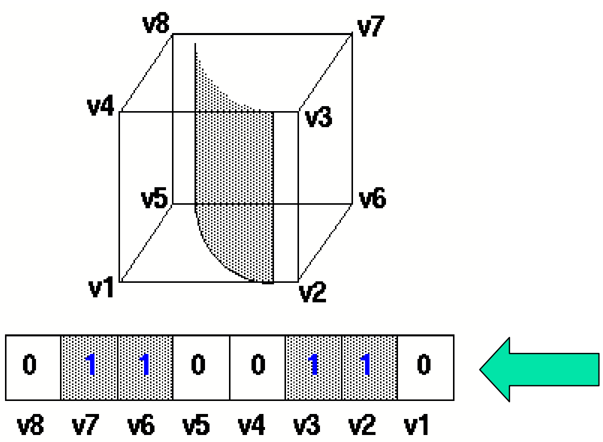
Classify each Vertex

Outside (0) if vertex value < surface value
Inside (1) if vertex value >= surface value



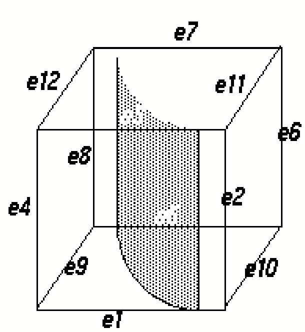
Build an Index

Binary Labelling



8-bit Index

Edge Table



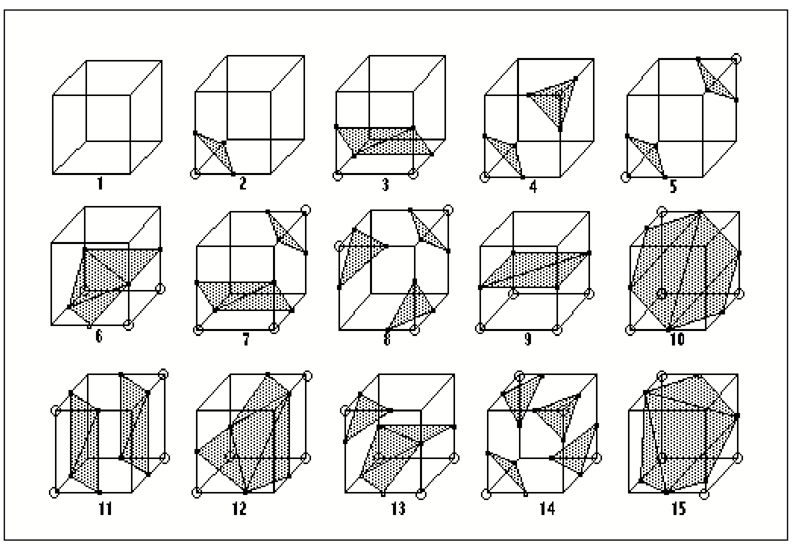
Index	Edge List
00000000	.
00000001	e1 e4 e9
00000010	e1 e2 e10
00000011	.
.	.
.	.
.	.
.	.
.	
.	
11111111	

Table will have 256 entries (8-bit index)

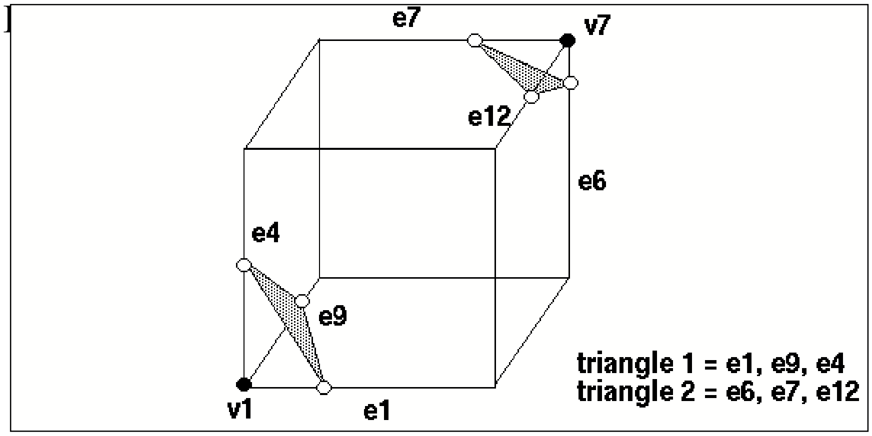
Edge Intersections

- There are 256 ways in which a surface can intersect a cube
- All 256 can be generated by 15 base cases:
  - By complimentary cases (swap 0's and 1's), reduces to 128
  - By rotational symmetry (reduces to 15)
- Thus edge intersections can be defined by the 15 cases.

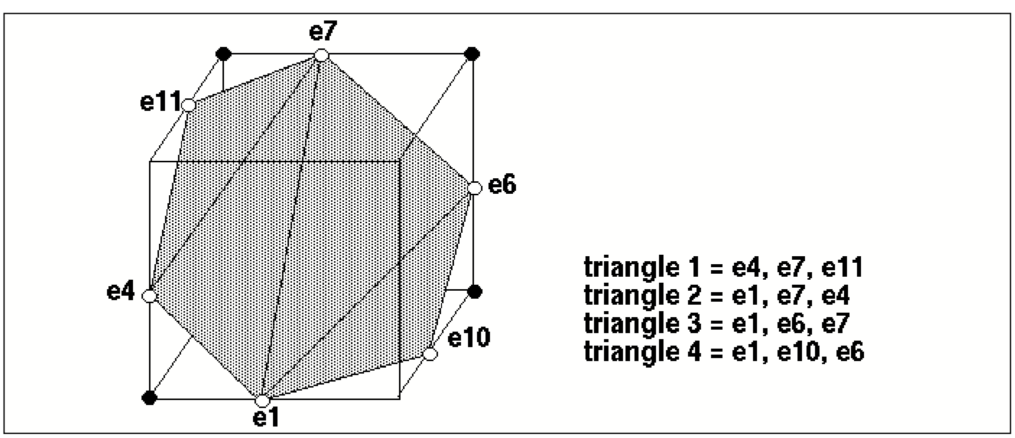
15 Canonical Configurations



Case 5

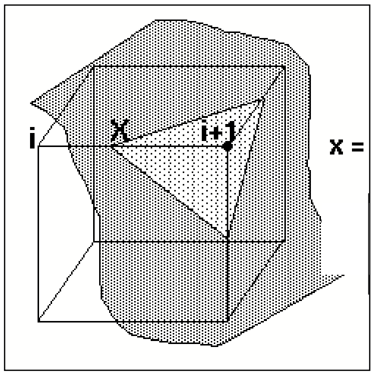

e7
v7
e12
e6
e4
e9
v1
e1
triangle 1 = e1, e9, e4
triangle 2 = e6, e7, e12

Case 9


e7
e11
e6
e4
e10
e1
triangle 1 = e4, e7, e11
triangle 2 = e1, e7, e4
triangle 3 = e1, e6, e7
triangle 4 = e1, e10, e6

Interpolate Triangles

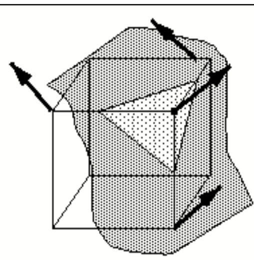
- For each triangle, find vertex by linear interpolation of the sample values



$$x = i + \frac{isovalue - D(i)}{D(i+1) - D(i)}$$

Calculate Normals

- For each triangle edge, find normals at the cube vertices from gradient of density data
- Interpolate normal at the point of intersection



$$G_x = D(i+1,j,k) - D(i-1,j,k)$$

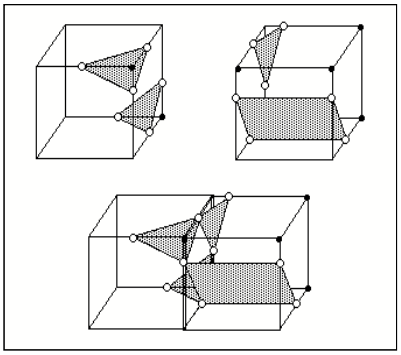
$$G_y = D(i,j+1,k) - D(i,j-1,k)$$

$$G_z = D(i,j,k+1) - D(i,j,k-1)$$

$$\vec{N} = \frac{\vec{G}}{|\vec{G}|}$$

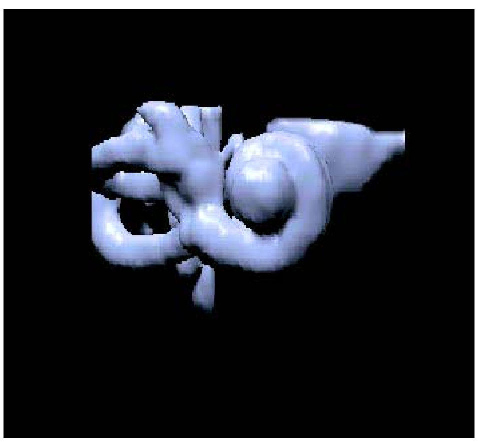
Ambiguities

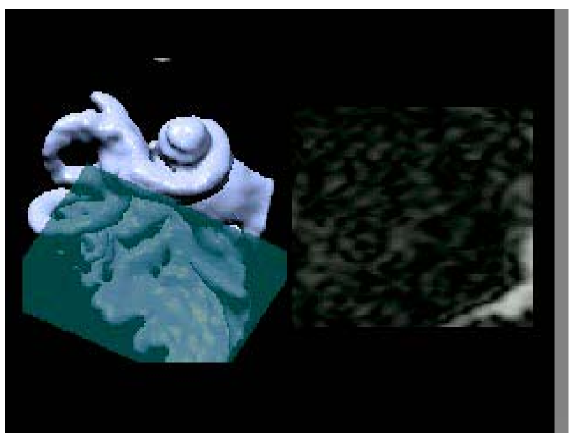
- There are six ambiguous cases
  - cause “holes”



- Solutions are available

Examples





Dividing Cubes

- Observation: many polygons produced by Marching Cubes are smaller than pixel size
- Basic idea:
  - Classify each voxel as being inside, outside or on the surface
  - Project surface voxels to the image plane
  - If it projects as a single pixel or smaller render it as a point
  - Else subdivide it into surfaces with the Marching Cubes Algorithm

Classify Voxels

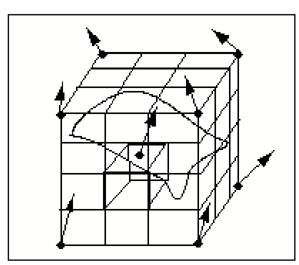
- Interior if all data values are above surface value
- Exterior if all data values are below surface value
- Otherwise surface intersects the voxel

Subdivide Surface Voxels

- In x, y, and z create small voxels that correspond to final image resolution
- Resample the intensities for the new voxels using trilinear interpolation
- For each new voxel, test if surface voxel as above

Calculate Normals

- For each surface voxel, interpolate gradient vectors for surface intersections (central differences)
- Calculate surface normal at each surface point
- Output points with normal

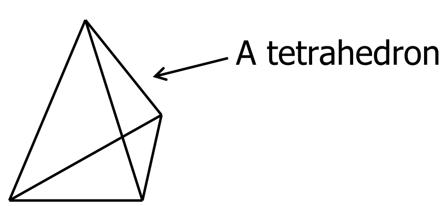


Comparison

- Marching Cubes
  - Creates triangles
  - Models rendered using conventional graphics techniques
  - Efficient for small volumes
- Dividing Cubes
  - Creates points with normals
  - Efficient for larger volumes

Marching Tetrahedra

- Observation: Tetrahedra are the fundamental unit of discrete space like a triangle is the fundamental unit of a discrete surface SO all polyhedral cells can be decomposed into a whole number of tetrahedra
- Basic Idea
  - Decompose all data cells into tetrahedrons
  - March through the tetrahedral cells in a similar way to Marching cubes



- Each tetrahedra can have a maximum of 6 edge intersections
- Requires only 3 unique entries in the table
- Results in a Maximum of 2 triangles/tetrahedron
- A rectangular cell/voxel can be subdivided into 5, 6 or 24 tetrahedra
- 5 tetrahedra case requires flipping to remove surface ambiguity

Comparison

- Marching Cubes
  - Less triangles
- Marching Tetrahedra
  - More robust, has no ambiguous solutions

Point Rendering

- High resolution datasets typically produce high resolution surfaces
- High resolution triangle based surfaces typically project so 1 triangle is less than a pixel
- If there are enough points there is implicit connectivity
- Triangular surfaces have vertices, edges, faces and connectivity - interpolation
- Point-based surfaces are simpler and faster – no interpolation
- Point rendering is implemented in hardware

Visibility Splatting

- If points are too sparse or the viewer too close there may be gaps in the surface
- Method mainly used for 3D models produced from laser scans
- Visibility Splatting uses surfels (surface elements) to fill in the gaps
  1. A surfel (disc perpendicular to the surface normal) is project into the Z-buffer.
  2. Holes in the Z-buffer are used to determine the size of the surfel

Splat Shape

- Kernel shape used for a point effects image quality and speed of rendering

1. Simplest to use is a non-antialiased OpenGL point (a square shape)
2. Next an opaque circle rendered as a small group of triangles or less expensively in OpenGL as a single texture mapped polygon
3. Most realistic is a fuzzy spot with an alpha that falls off radially with a Gaussian or some other approximation

- Get slower in order 1-3
- For 3 splat order is important

Comparison

- Marching Cubes
  - Creates a geometry
- Point Rendering
  - Anti-aliasing/interpolation solved
  - Volume rendering can also be solved with splatting

Surface Ext. Summary

- Advantages
  - Uses common rendering methods
  - Changing views/lights only requires re-rendering of geometry
  - Can be done in hardware
- Disadvantages
  - Can lead to false positives, or false negatives
  - Throws away data between surfaces
  - Cannot image interior of volume

Direct Rendering

- To render the volume directly without recourse to intermediate geometry.
- To allow the display of weak and fuzzy surfaces.
- Can relax the condition that a surface is either present or not.

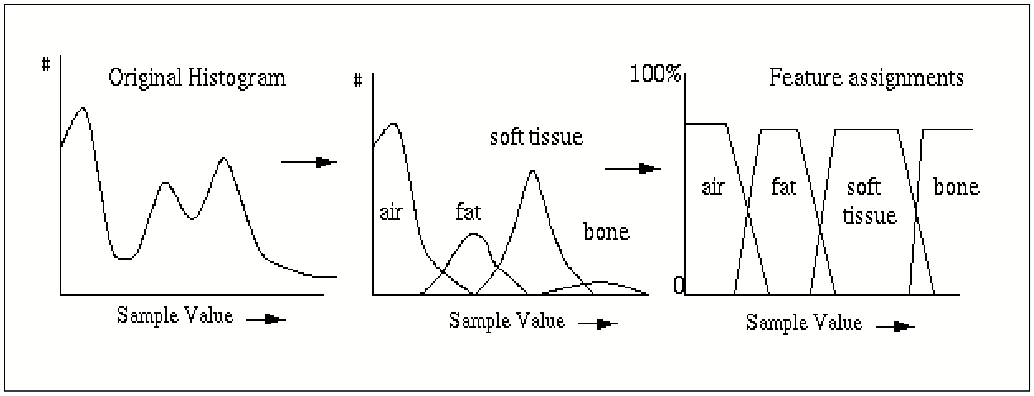
How is it Done?

- Software: Direct Rendering Algorithms
  - Raycasting, Splatting
- Hardware
  - Special Purpose e.g. pixel-flow, cube4
  - Graphics Pipeline: texture mapping hardware
- APIs
  - OpenGL Volumizer
  - Voxelator - HP's OpenGL extensions

Segmentation

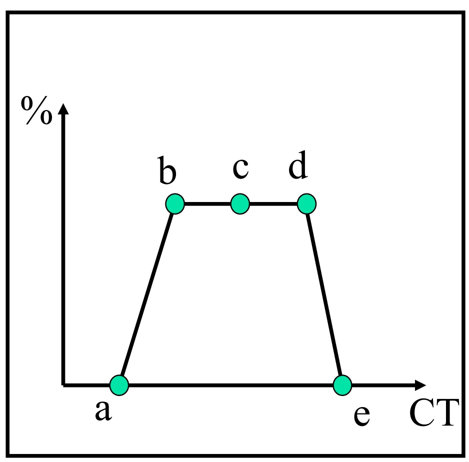
- Point-Based
  - A voxel is classified only dependent on its intensity
  - Thresholding
- Edge-based
  - Detect intensity discontinuities in a grey level volume e.g. detect maxima of the first derivative of the 3D intensity function (Canny)
- Region-based
  - Consider properties of regions e.g. size, shape, location, variance of grey levels.

Data Classification

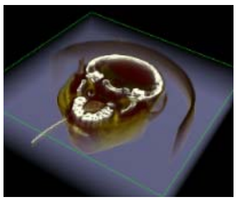


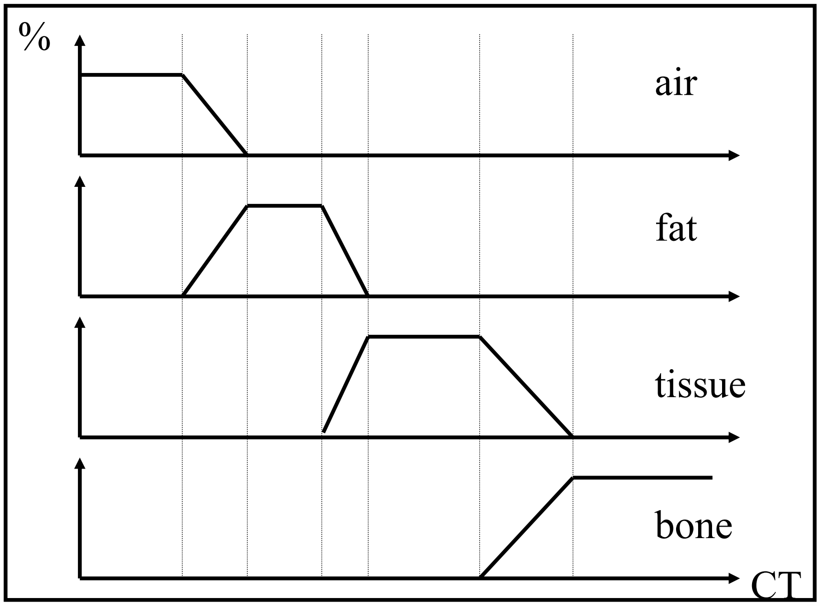
Trapezoid Classification Function

- There will exist a particular CT value that is most likely to represent the material at point c
- Points b and d represent the maximum deviation in CT number from point c that is still considered to be the same material
- Any CT number that is less than b or greater than d, and contained within the limits defined a and e, is defined to be a mixture of neighbouring materials



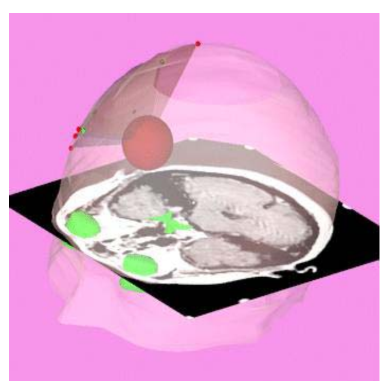
4 Classification Functions Needed



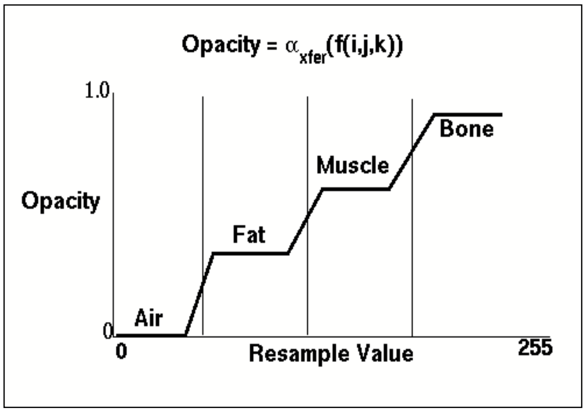


Transparency

- Opacity
  - All light is prevented from passing through the material (α=1)
- Transparency
  - All light passes through the material (α=0)
- Translucency, semi-transparency
  - Graded or blurred transparency ($0<\alpha>1$)



Opacity Classification



Volume Rendering Integral

For any ray, amount of light received at the image plane is:

$$I_\lambda = \int_0^L C_\lambda(s)\mu(s)e^{-\int_0^s \mu(t)dt} ds$$

λ = wavelength
μ = density
L = length of ray
$C_\lambda(s)$ = light of wavelength λ reflected at s in ray direction

Simple Riemann Sum Approximation

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i\Delta s)\mu(i\Delta s)\Delta s \prod_{j=0}^{i-1} \exp(-\mu(j\Delta s)\Delta s))$$

Where n is the number of sample steps along the ray

Make further approximations …

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i\Delta s)\alpha(i\Delta s) \prod_{j=0}^{i-1} (1-\alpha(j\Delta s))$$

Compositing Formula

If using unit spacing:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i)\alpha(i) \prod_{j=0}^{i-1} (1-\alpha(j)) \quad *$$

Where α is opacity

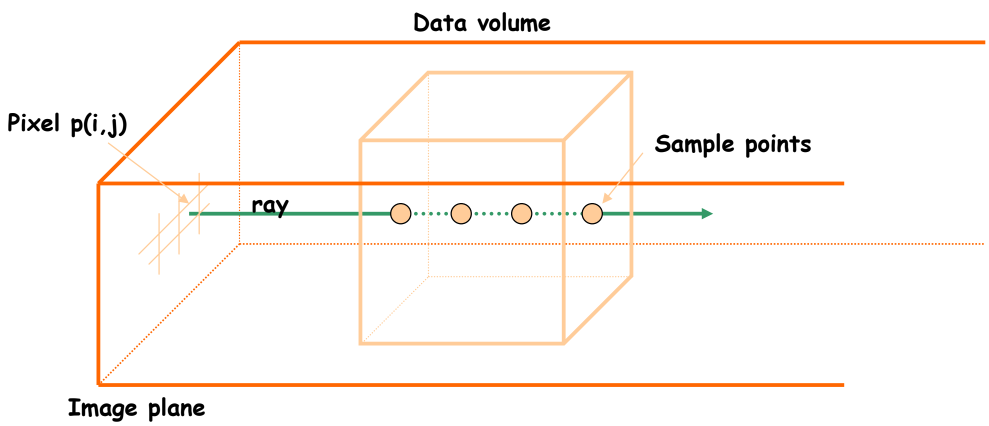
In practice, computation is done for RGB Components separately.

Traversing the Volume

- Feed Forward Projection / Object Order Traversal
  - The data volume is traversed and each voxel in the volume is projected onto the image plane.
- Feed Backward Projection / Image Order Traversal
  - The pixels in the image plane are traversed and imaginary rays are cast through each pixel into the volume. The path of the ray determines the pixel value.

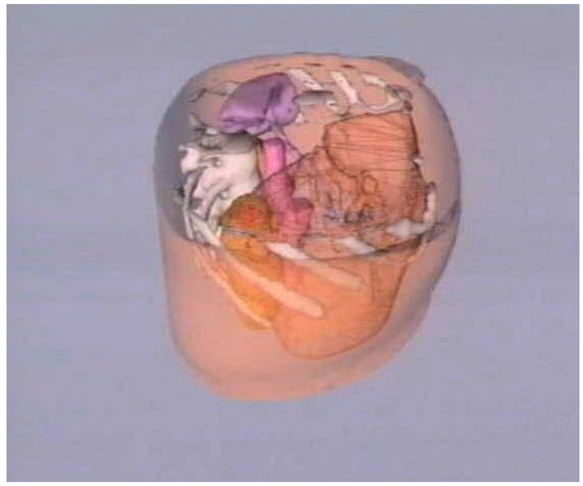
Ray Casting

- Shade the acquired data to obtain a volume of colour values.
- Classify the data to obtain a volume of opacity values.
- Cast a ray through both volumes and take samples along the ray.



Ray Casting

- At each sample point calculate the colour and opacity using tri-linear interpolation.
- Composite the colour and opacity samples along the ray to produce a total colour for the ray.

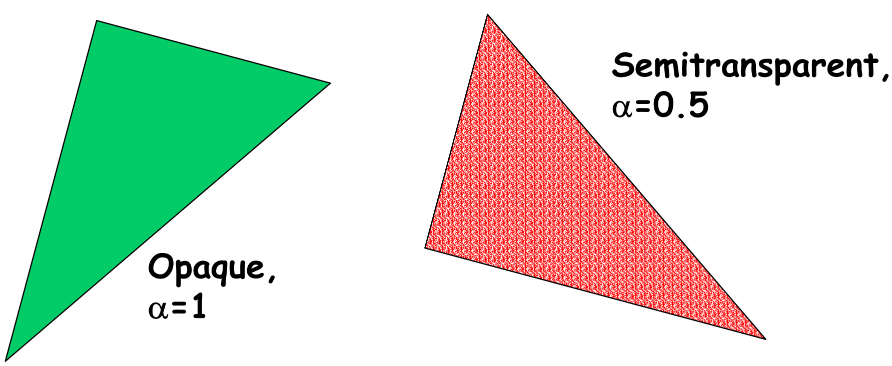


THE UNIVERSITY of MANCHESTER

Simple Example

I(x,y,z) = Intensity value of a voxel

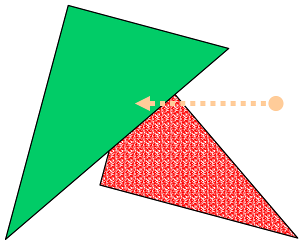
α(x,y,z) = Opacity value



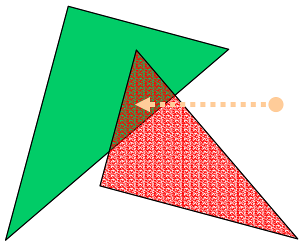
THE UNIVERSITY of MANCHESTER

Blending - "Over" Operator

M over N: $I = (1-\alpha_M)I_N + \alpha_M I_M$

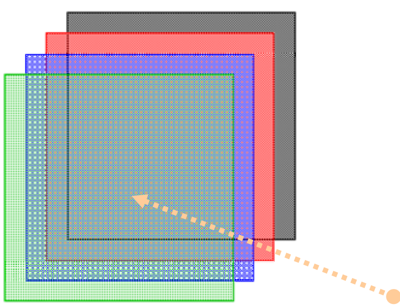


Intensity = $(1 - 1)I_{red} + 1 \times I_{green}$



Intensity = $(1 - 0.5)I_{green} + 0.5 \times I_{red}$

Compositing



$I_2 = (1 - \alpha_2)I_1 + \alpha_2 I_2$

$I_3 = (1 - \alpha_3)I_2 + \alpha_3 I_3$

$I_n = (1 - \alpha_n)I_{n-1} + \alpha_n I_n$

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i)\alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j)) \quad *$$

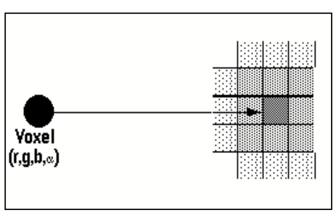
Serial Optimisation

- Early Ray Termination
- Hierarchical Spatial Enumeration
- Adaptive Screen Sampling
- Adaptive Ray Sampling
- Templates
- Space Leaping

But still compute intensive - too slow for real-time

Splatting

- Determine in what order to traverse the volume
- Classify and shade the voxels in each slice
- Project or splat each slice onto the image plane



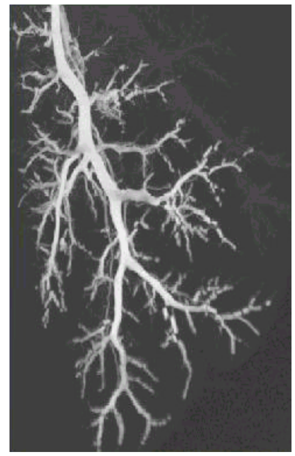
- Determine the extent of the splat's contribution and attenuate according to some filter
- Merge the attenuated splats to calculate the pixel contributions

Can be likened to throwing a snowball onto a glass plate

Other Lighting Models

- Maximum Intensity Projection (MIP)
- Minimum Intensity Projection
- X-Ray Projection

No alpha channel needed



MIP image showing Bile Ducts in a Rat's Liver

Direct Rendering Summary

- Advantages
  - Non-binary classification
  - Shows structure between surfaces
  - Displays small and poorly defined features
  - Readily parallelisable
- Disadvantages
  - Expensive. Cost is proportional to volume size $O(n^3)$
  - Combining volume and geometry data is difficult

Usability

- 3 issues effect usability

1. Speed of processing data
2. Speed of interaction (25 frames per second for film)
3. Ease of determining optimal isovalue or colour/transparency map

Speed

- The quality of the image and the speed it is rendered at is a property of:

1. The size of the data
2. Structure of data
3. Angle and distance it is view from
4. Available memory
5. Graphics hardware

Presorting

- Put the data in a structure normally a tree so that particular cells are easily found:
  - cell value for isosurfacing or transparency maps
  - location for view dependant algorithms
- If the data is presorted on data value the number of cells that need to be tested against the isovalue is reduced and speed up is gained each time a new isosurface is generated.
- To do this the tree must be organised so that a test at a high level eliminates a large number of cells.
- The “multi resolution” or “level of detail” approach uses a similar tree structure but one that also allows less detailed isosurfaces to be generated. Speed up can be gained for isosurface generation and for isosurface manipulation.

Presorting

- Presorting may speed up interactive processing and rendering but comes at a cost:
  - Time
  - Computational resources
- Data can be presorted once before use and then stored on disc in its presorted format
- More importantly a presorted tree takes up more memory than the original data
- If data is too large to fit into main memory then use out of core algorithms. Need an on disc data structure that allows optimal I/O performance for the paging of data to and from disc.

Meaningful Isosurfaces and Transfer Functions

- Any data value can be used to give an isosurface
- User can spend too much time finding meaningful isosurfaces
- In the same way for direct rendering it is the transfer function that makes meaningful features visible

Finding Material Boundaries

- Use image processing techniques to identify the boundaries of different features of interest
- In volume data obtained from a imaging system like a medical scan then the transitions between different materials correspond to intensity transitions
- Detect the major intensity transitions and you have meaningful isovalues or a data value that should lie on the steepest slope of a opacity transfer function

Benefits

- Automatic generations of isosurfaces or transfer functions
- Semi-automatic generation with better user interfaces allows the user to explore the features in the data more fully and quickly
- More complex transfer functions (2D or 3D) can be generated and manipulated in a meaningful way

Manchester Computing

Parallelisation Strategies

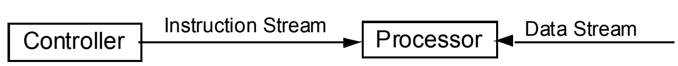
THE UNIVERSITY of MANCHESTER

Overview of Parallel Architectures

- A classification scheme for architectures
  - Flynn in 1972
- Outdated, however still widely used
- Four categories:
  - SISD - Single Instruction stream, Single Data stream
  - MISD - Multiple Instruction stream, Single Data stream
  - SIMD - Single Instruction stream, Multiple Data stream
  - MIMD - Multiple Instruction stream, Multiple Data Stream

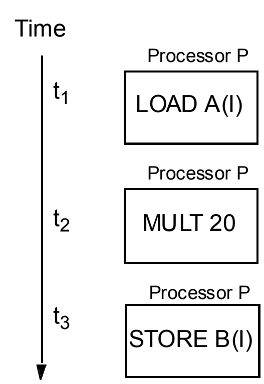
Single Instruction, Single Data Stream

- Conventional Uni-processor architecture



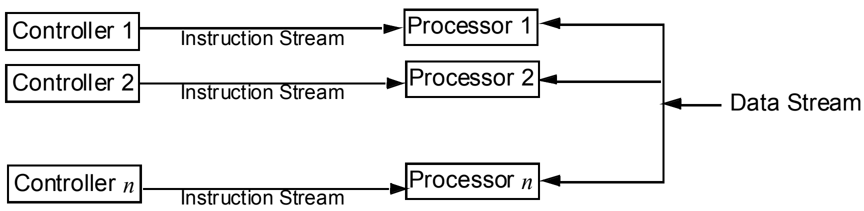
B(I)=A(I)*20

LOAD A(I)
MULT 20
STORE B(I)

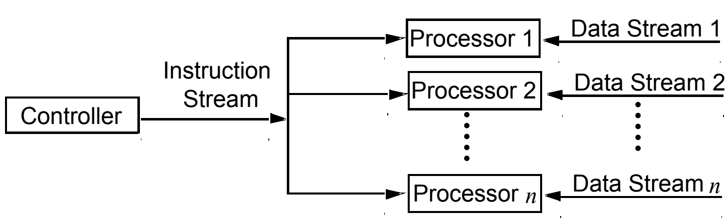


Multiple Instruction, Single Data stream

- Several processors simultaneously execute different instructions on one data stream
- Used in pipelines, e.g graphics hardware



Single Instruction, Multiple Data Stream

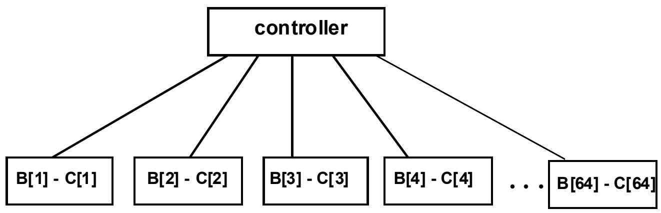


- Single instruction executed on several data streams simultaneously in lock step manner.
- In general processors are very simple and act like mindless clones.
- Used in processor arrays like the DAP, Connection Machine CM2. Also in Pixel Planes machine

SIMD Example

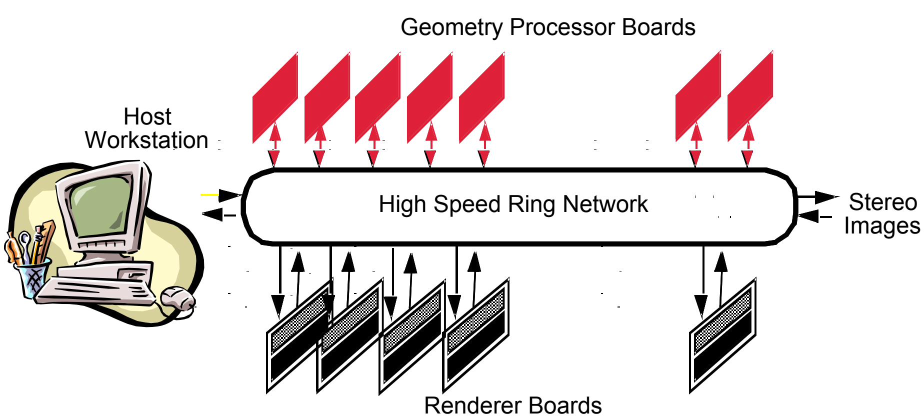
- Each element of the array is calculated simultaneously

```
for i = 1, 64 do
      answer[i] := b[i] - c[i];
```



Example Machine — Pixel Planes 6

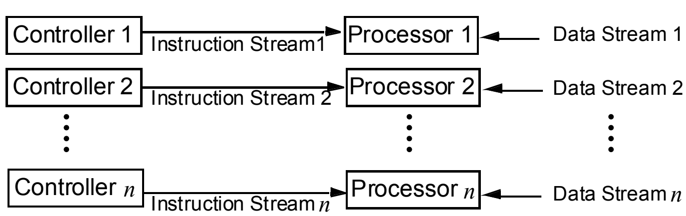
- Developed by University of North Carolina
- Marketed by Division Ltd (now defunct).



Example Machine — Pixel Planes 6

- Each Geometry Processor (GP) contains two processors each with 32MBytes of local memory. A subset of the graphics database is distributed to each GP.
- All GP's operate in parallel to transform the graphics database and send rendering commands to the Rendering Processors (RPs)
- Each Rendering Processor board has 64 custom chips each with 256 processors, giving 16,384 processors per board
- The RP's work in a lock-step mode rendering up to 1 GPixels per second per board.

Multiple Instruction, Multiple Data Stream



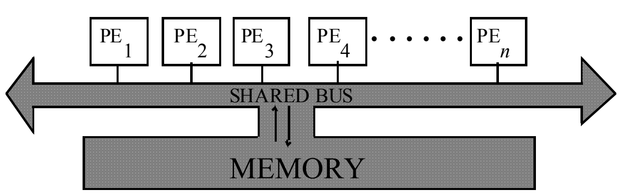
- Essentially separate computers working together to solve a problem
- Includes networks of workstations
- All other classes are sub-classes of MIMD
- We will concentrate on MIMD

Other Architectural Issues
Connection to Memory

- Shared Memory
  - Bus based
  - Interconnection network
- Distributed Memory
  - Message passing
- Virtual shared memory
  - DM but looks like SM

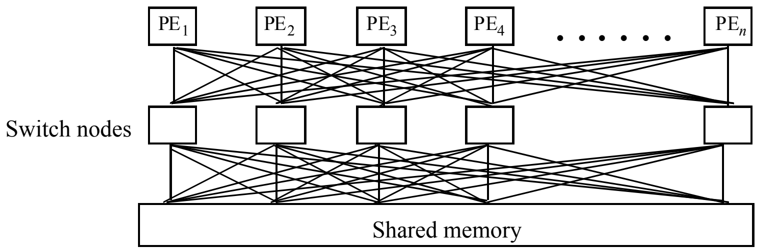
Shared Memory

- One common memory block shared between all processors
- Connection either by shared bus or inter-connection network
- Since bus has limited bandwidth, number of processors which can be used is limited to a few tens of processors
- Examples include Encore, Sequent, SG Power series



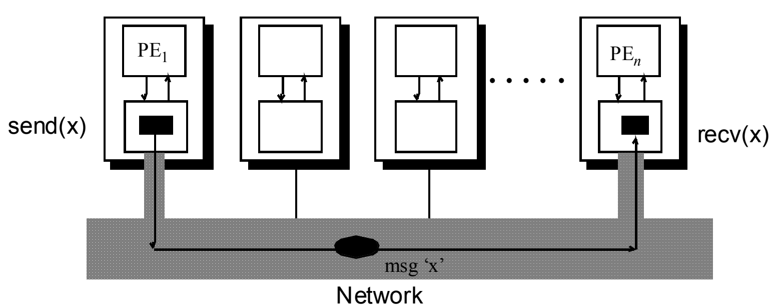
Switch based shared memory

- Uses network between processors and memory modules.
- May use multi-stage networks
- Increases bandwidth to memory over bus-based systems
- Every processor still has access to global store
- In general provide Non-Uniform Memory Access



Distributed Memory

- Message Passing. Memory physically distributed throughout the machine. Each processor has private memory
- Private memory can only be accessed by that processor. If required by another then it must be sent explicitly.

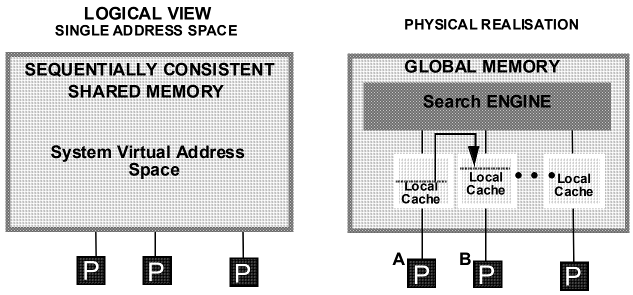


Distributed Memory

- No memory bottleneck
- Machines scale to hundreds (thousands) of processors
- Considered difficult to program due to message passing and difficulty to debug.
- Examples include Intel Paragon, Meiko CS2 and Cray T3DT3E, IBM SP2.

Virtual Shared Memory

- Objective: scalability of distributed memory with the programmability of shared memory
- Global address space mapped to physically distributed memory
- Data moves between processors "on demand", i.e as it is accessed. Examples include KSR, DASH, SGI Origin 2000



Issues in Parallel Programming

- Load Balancing
- Levels of Granularity
- Nature of Parallelism
- Data Coherence
- Data Access
- Scalability

Load Balancing

- To encourage an equal distribution of work throughout the processors
- Each processor is used as effectively as its neighbours
- Equal amounts of work mean that all processors finish their work at the same time
- Typically address by task partitioning:
  - Static assignment of large tasks to processor
  - Dynamic assignment of smaller tasks

Static Task Assignment

- Typically the number of tasks is equal to the number of processors
- All tasks are estimated to take approximately the same amount of time
- Advantages:
  - Communication overhead tends to be smaller due to longer tasks
  - Task startup cost is minimized and scheduling overhead reduced
- Disadvantages:
  - Often require pre-processing to ensure that tasks are roughly the same size
  - Limited number of applications

Dynamic Task Assignment

- Number of tasks T >> number of processors P
- Processors are assigned tasks from a pool of tasks waiting to be executed.
- Processors work on tasks until the task pool is empty.
- Processors then idle until all processors complete.

Dynamic Task Assignment

- Advantages
  - Task size does not need to be known in advance
  - Idle time is usually small and has minimal impact
  - Load balancing solved dynamically
- Disadvantages
  - Communication overhead tends to be higher.
  - Task initialization and termination time can be significant.
  - If task *granularity* is too small then may lead to excessive communication

Levels of Granularity

- Granularity — A measure of the size of an individual task to be executed on a parallel processor

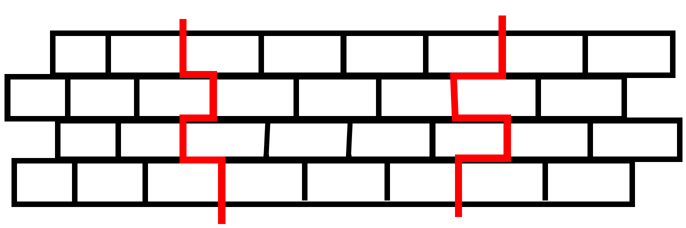
  - Stone, "High Performance Computer Architectures"
- Coarse
  - Execution of P modules in parallel on P processors i.e, large tasks
- Medium
  - Execution of N modules on P processors in parallel where N>>P
- Fine
  - Parallel computations of loop iterations in parallel.
    e.g., for each pixel in an image

Nature of Parallelism

- Principal types
  - Data Parallelism (Geometric)
  - Functional Parallelism (Procedural)
  - Farm Parallelism

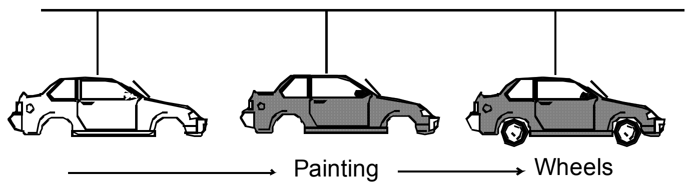
Data Parallelism

- Divide up data amongst processors
- Process different data segments in parallel
- Maybe requirement for communication at borders
- Maybe inefficient if data access patterns are unknown



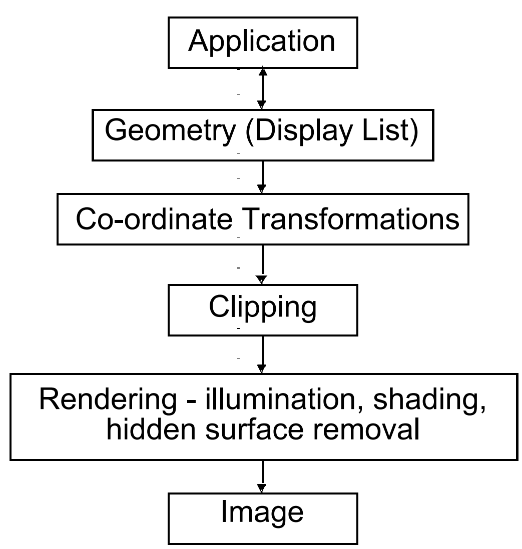
Functional Parallelism

- Different threads of control
- Decompose the algorithm into different sections, assigned to different processors
- Pipelining is a form of functional parallelism



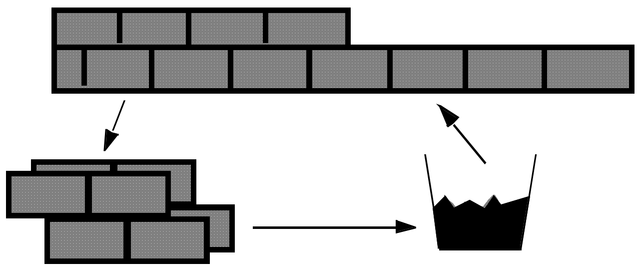
Functional Parallelism

- E.g. Graphics Pipeline



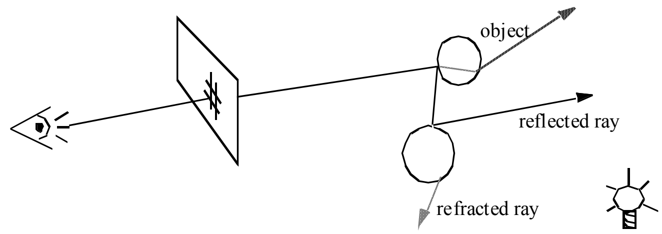
Farm Parallelism

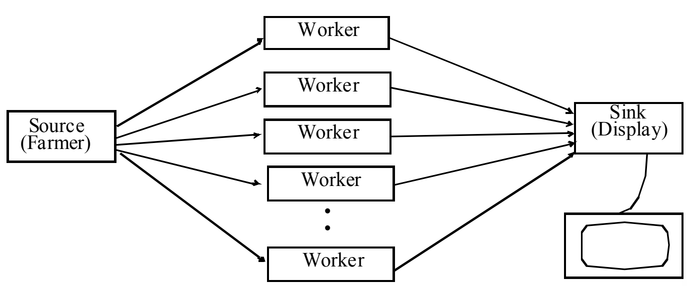
- Usually used with dynamic task assignment. Similar tasks are generated by a “source” process and maintained in a pool
- “Worker” Processors repeatedly take tasks from the pool and perform the calculation and pass them onto a “sink”



Farm Parallelism continued

- Example use in ray tracing





Data Access

- Concerned with movement of data between processors
- Try to avoid access to data on remote processors where possible. Differences in access times between local and remote data can be dramatic, particularly on networks of workstations
- Exploit data coherence where possible

Data Coherence

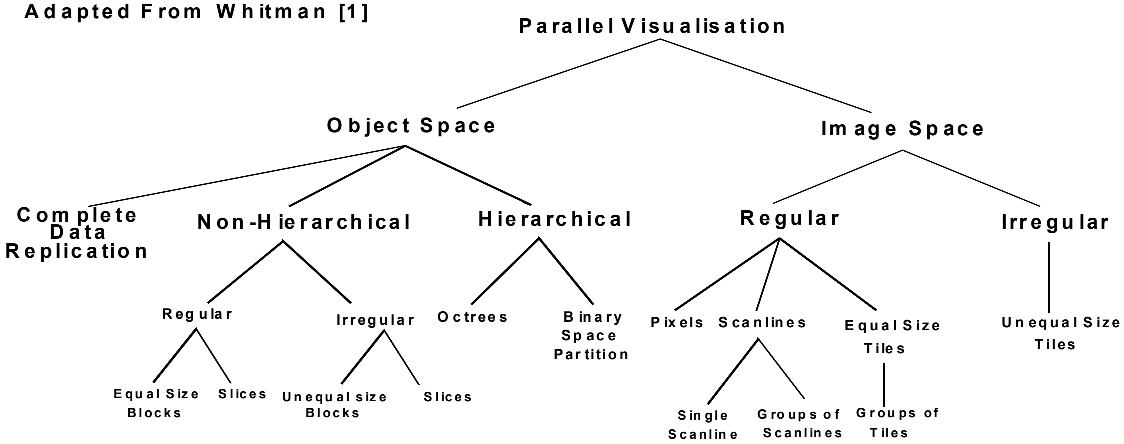
- Concerned with exploiting coherence in data to avoid re-computations and number of remote accesses.
- Pixel level coherence
  - High probability that neighbouring pixels have similar values in an image
- Area level coherence
  - Areas of pixel are likely to have similar values in an image, and will therefore use similar data.
  - Maybe groups of scan lines
- Frame level/Temporal coherence
  - e.g. small changes between frames in an animation sequence
  - Some algorithms/machines can exploit this data coherency between frames

Parallelism in volume visualization

- Volume visualization exhibits high degree of parallelism
- Two major partitioning schemes:
  - Object partitioning
  - Image space partitioning
- Many ways of partitioning either of these.

Taxonomy of Parallel Visualization Decompositions

Adapted From Whitman [1]

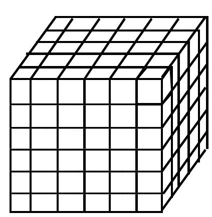


Object Space
Complete Data Replication

- All data is held locally at each processor
- Simple parallelization since multiple instances of the same sequential algorithm
- No communication during compute phase
- Wasteful of memory. Does not scale with data set size
- Impractical on massively parallel machines due to high cost of initial data distribution
- Only useful for read-only data

Object Space
Non-Hierarchical

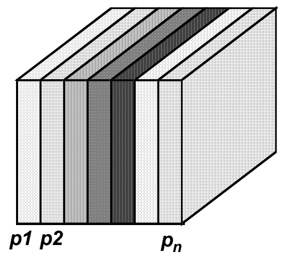
- regular decomposition — equal size blocks
  - Break down the data set into regular 3D regions
  - Each processor works on separate 3D region(s)

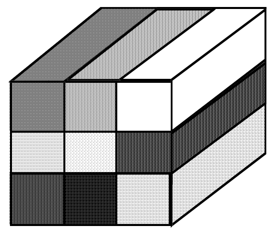


  - Can lead to poor performance if load balancing is not handled correctly

Object Space Non-Hierarchical

- Regular Decomposition — Slices, Slabs or Shafts



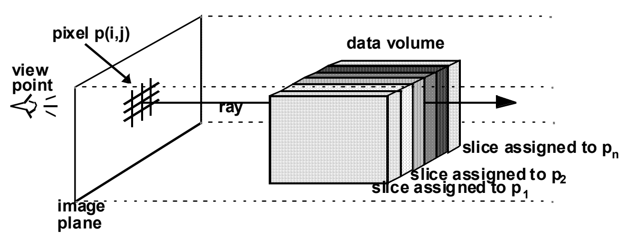


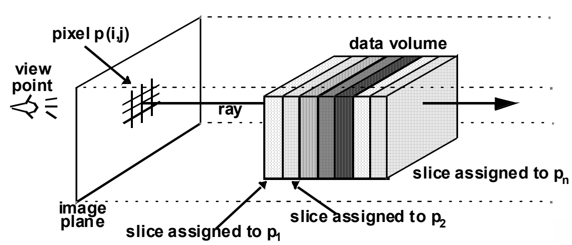
  - Suited to architectures which are ring based or chains since only nearest

THE UNIVERSITY of MANCHESTER

Problems

- Means that for some algorithms performance is view-dependent e.g., ray casting

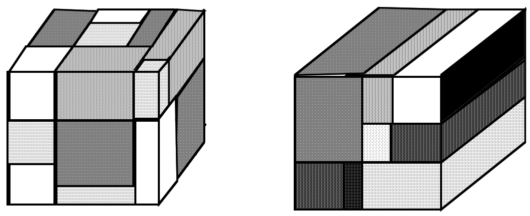




THE UNIVERSITY of MANCHESTER

Object Space Non-Hierarchical

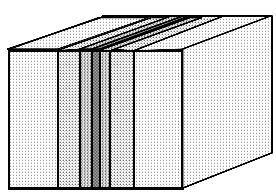
- Irregular Decomposition — Unequal sized blocks
  - Break the data set up so that each block represents similar amounts of work



  - Hard to estimate this, requires pre-processing

Object Space Non-Hierarchical

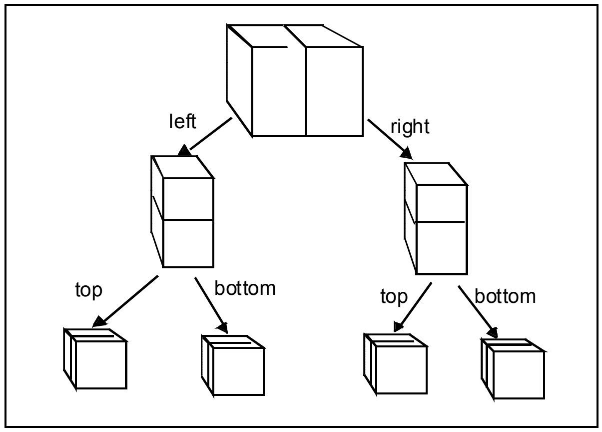
- Irregular Decomposition — Slices or Slabs



  - Slices may be different widths to aid load balancing
  - Again requires pre-processing to determine slice width

Object Space Hierarchical Approaches

- Kd Tree Data Partitioning



Object Space Hierarchical Approaches

- Octree Subdivision

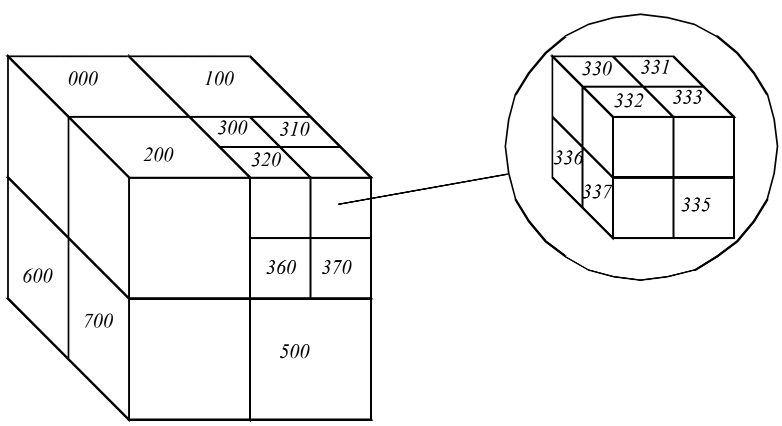
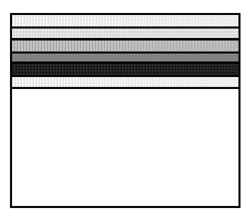


Image Space Regular

- Processor per pixel
  - very fine grained parallelism
  - In general used on SIMD machines
  - Time to render one pixel may be several orders of magnitude different to time to render another
  - Should be load balanced if many more pixels than processors (i.e task pool)

Image Space Regular



- Scanlines
  - Single
  - Groups
- Assumptions
  - Distribute Scan lines interleaved in round-robin fashion
  - Some pixels on a scan line are computed trivially, while others take longer to compute
  - Hence, should be intrinsically load balanced

Image Space Regular

- Regular Spaced Pixel tiles
  - Straightforward to implement
  - Exploits area coherency in neighbouring pixels
  - Each block is generally independent of the other blocks, so it is not necessary for any communication between blocks to take place
  - Can vary the size of the blocks to suit the granularity of the machine
  - Often implemented using Task-Farm parallelism

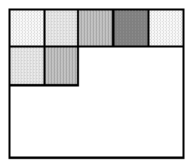
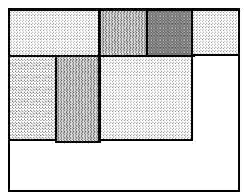
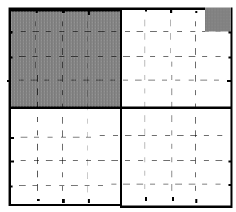


Image Space Irregular

- Again exploits area coherency
- Uses pre-processing to calculate load for each pixel tile
  - Subsample the image plane, and keep a note of the time taken to calculate the subsampled pixels
  - Cluster tiles together to form new tiles with roughly equal times to render
  - Should be better load balanced



- Groups of tiles
  - Each processor works on a block of square tiles held in its queue
  - When a processor finishes rendering the tiles in its queue, it steals tiles from other processor queues



Manchester Computing

Parallelisation Implementation

THE UNIVERSITY of MANCHESTER

Parallel Methods

- Parallel Surface Extraction
  - Marching Cubes on a SIMD machine
  - Marching Cubes on a DM-MIMD machine
  - Marching Cubes on a VSM-MIMD machine
- Parallel Direct Methods
  - Ray casting on a network of workstations
  - Ray casting on DM-MIMD
  - Ray casting on VSM-MIMD
  - Splatting on DM-MIMD
- Parallel compositing

Parallel Surface Extraction

- Recall Marching Cubes Algorithm:
  - Create a bit index by classifying each voxel as being inside or outside the surface
  - Look up edge intersection in a pre-computed table
  - Interpolate edge intersection and gradients
  - Construct Triangles
- "Marching Cubes on a Connection Machine CM2" Hansen et al. [1]
- Target Architecture CM2
  - SIMD machine with 64,000 processors. Each processor can simulate a number of virtual processors (VPs). Lock step parallelism
  - Supports fast nearest neighbour communication
  - Each processor has only 256k of memory

Parallel Isosurface Extraction SIMD

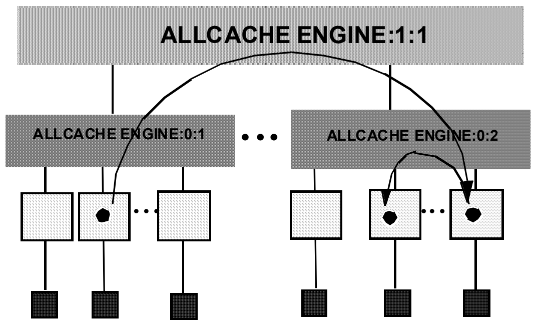
- Parallelization Strategy
  - Each voxel is distributed to a different virtual processor - essentially one voxel per processor - very fine grained
  - Each virtual processor then performs communication with its nearest neighbours to obtain the neighbouring voxels required for edge intersection and gradient calculations
  - Edge intersections are calculated locally using trilinear interpolation. Edge intersections are then communicated to neighbouring virtual processors until a complete triangle list is produced.

Parallel Isosurface Extraction SIMD

- Implementation
  - The full 256 case look up table is used rather than 15 base cases to avoid handling special cases in lock-step manner. All VPs need access to the table but don't have sufficient memory to hold it
  - On CM2 groups of 32 processors can share memory. Therefore the table is stored once for every 32 processors. This allows the table to be accessed in parallel by the 32 processors
- Observations
  - The algorithm is not portable to other machine as it exploits explicit characteristics of the CM2
  - The lock step nature of the machine means that performance is bounded by the number of triangles in a given voxel, rather than the total number of triangles

Parallel Isosurface Extraction VSM

- "Marching Cubes on a Virtual Shared Memory Architecture", Grant et al. [2]
- KSR Architecture



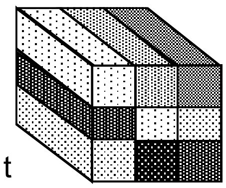
Data Location	Latency
PE on a different AE:0	30 μs
Different PE same AE:0	8.7 μs
Local-cache	0.9 μs
sub-cache	0.1 μs

Parallel Isosurface Extraction VSM

- KSR Architecture
  - Each processor has a 32 Mbyte cache and a 0.5 Mbyte subcache
  - Data is moved on demand when it is referenced. To distribute data around the machine, distribute the iteration space rather than use explicit message passing
  - The mapping between an address and a physical location is handled by the hardware
  - The unit of data movement is a subpage of 128 bytes. i.e when an address is referenced then that address and the surrounding subpage are fetched

Parallel Isosurface Extraction VSM

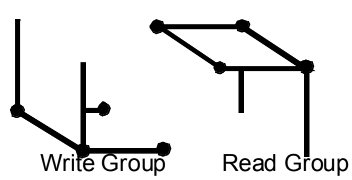
- Parallel Design
  - Crucial on this machine to exploit data coherency to minimize the amount of non-local communication
  - Need to avoid writing to shared data structures where possible
  - Split the data volume into shafts along the first dimension - this takes advantage of data coherency since neighbouring voxels in the shaft are neighbours in contiguous memory blocks.



  - Each processor is responsible for calculating the t shaft

Parallel Isosurface Extraction VSM

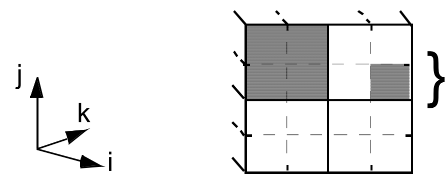
- Implementation
  - Each voxel shares edges with other voxels and so at the edges of the shafts there is a need to exchange edge information.
  - In reality each voxel has a "read" and a "write" group, where triangles are only calculated for its write group. Dummy references are maintained for the other edges.



  - Each processor will maintain a triangle list for its subvolume, which has some dummy references. In the final stage these lists are combined and the references resolved.

Parallel Isosurface Extraction VSM

- Optimizations
  - Since the vertices in each write group are stored contiguously, then when the corresponding read group references the first true vertex, all the vertex values will be fetched in the 128 byte subpage. This decreases amount of data referencing
  - Distributed task queue.



  - Each processor assigned a queue of subvolumes. Processors take subvolumes from their own queue until empty and then steal from other processor's queues
  - Grouping like this means that more data references will be local

Parallel Ray Casting Workstation Clusters

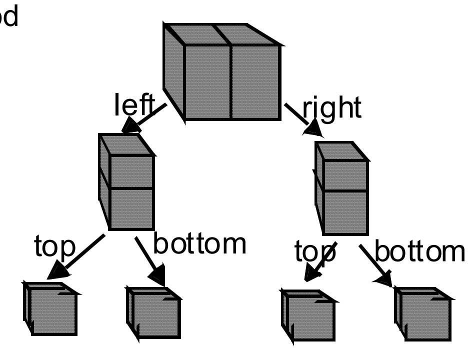
- "A distributed parallel algorithm for ray traced volume rendering" Ma et al. [3]
  Environment
  - A network of high performance heterogeneous workstations connected by Ethernet
  - non-dedicated computing cycles
  - message passing communication using the PVM libraries
- Motivation
  - Datasets often too large for a single workstation
  - Clusters are prevalent and much cheaper than MPPs

Parallel Ray Casting Workstation Clusters

- Recall the Ray casting algorithm
  - Shade the acquired data to obtain a volume of colour values
  - Classify the data to obtain a volume of opacity values
  - Cast a ray through both volumes and takes samples along the ray
  - At each sample point calculate the colour and opacity using tri-linear interpolation
  - Composite the colour and opacity samples along the ray to produce a total colour for the ray.
- Parallelization Strategy
  - Distribute data amongst processors using a hierarchical subdivision method
  - Distribute viewing information and classification tables
  - Each processor calculates a partial image for its local subvolume
  - Partial images can then be merged in parallel, ensuring load balancing

Parallel Ray Casting Workstation Clusters

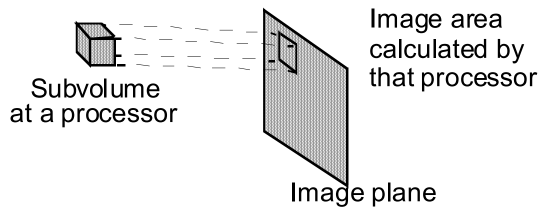
- Data Distribution
  - Need to establish an unambiguous back-to-front ordering, to ensure correct compositing
  - Uses the Kd Tree Method



  - Difficult to ensure load balancing, since each subvolume may not represent equivalent amounts of work

Parallel Ray Casting Workstation Clusters

- Calculation of the partial images
  - Each processor has information on the view position and orientation of the image plane. Only rays within the image region corresponding to the subvolume are cast.



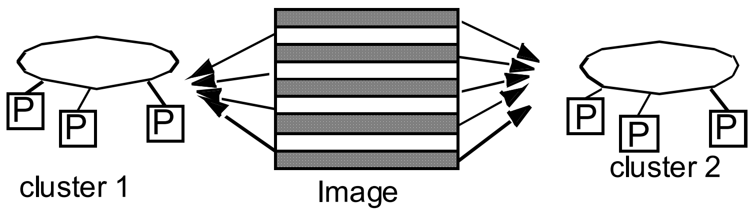
  - Ray casting takes place independently for each subvolume, so that no communication is required during this phase. End result is that each processor holds a partial image
  - Use a parallel compositing technique (see later) to reconstruct the total image from the partial images

Parallel Ray Casting Workstation Clusters

- Summary
  - Workstation clusters can provide a useful environment when large datasets are to be rendered
  - Achieving real time in this kind of environment is difficult due to high latency communications
  - Algorithm is scalable due to the distribution and parallel compositing method.
  - Algorithm is also applicable to MPP systems
  - Load balancing is difficult using this kind of decomposition. Alternatives using static or dynamic partitioning can lead to either more communication or more pre-processing.

Parallel Ray Casting DM-MIMD

- "Volume rendering on a distributed memory parallel computer" Montani et al. [4]
  - Implementation on a nCube 2 with 128 nodes, 4Mbytes of memory per node
- Parallelization Strategy
  - Group processors into clusters. Replicate the volume in each cluster
  - Divide the image space by scanlines, so that each cluster is responsible for rendering a subset of the scanlines



Parallel Ray Casting DM-MIMD

- Data Decomposition
  - A slice strategy is used to decompose the copies of the data volume within each cluster
  - The justification is that the partitioning is straightforward and the communication patterns are simple
- Ray Dataflow
  - Rays are cast into the data volume. When a ray reaches the bounds of its slice partition it must be packaged and communicated to adjacent nodes
  - The identifier of a processor within a cluster provides a unique address for the slice partitions it holds

Parallel Ray Casting DM-MIMD

- Load Balancing
  - A static load balancing scheme is used.
  - Initially each processor has a uniform number of slices. A subset of rays are calculated and the load redistributed according to the partial results
- Observations
  - Load balancing scheme requires pre-processing step
  - Since a slice strategy is use the performance of the algorithm is view dependent
  - As more processors are added slice partitions become narrower, and rays reach their bounds sooner. This means more ray communication

Parallel Ray Casting DM-MIMD

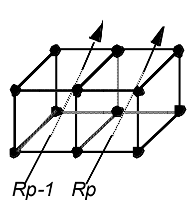
- "Parallel Volume Rendering and Data Coherence" Corrie et al. [5]
- Environment
  - Fujitsu AP1000
  - DM-MIMD message passing architecture with up to 1024 nodes.
  - Wormhole routed 2D mesh network
- Parallelization Strategy
  - Image space task partition
  - Implementation of distributed shared memory for data distribution

Parallel Ray Casting DM-MIMD

- Image space task partitioning
  - Uses square pixel tiles to exploit data coherence. More tiles than processors
  - To improve load balancing implement a work item timeout period. When a processor is struggling the work is re-distributed to other processors
- Data distribution using DSM
  - Each processor maintains two lists - persistent and cache
  - Persistent list - Use to hold part of the volume which it serves to other processors
  - Cache list - An LRU cache which is used to hold data which is fetched from other processors persistent lists

Parallel Ray Casting DM-MIMD

- Data coherency
  - Since pixels in a neighbourhood of an image will use the same data value then once the data is cached for the first ray it should be present for the second ray



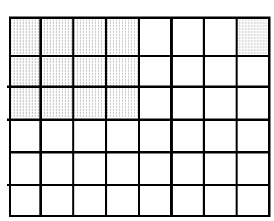
  - If the LRU cache is too small can lead to thrashing
- Performance
  - Efficiencies of between 80 and 95% obtained on up to 128 processors
  - Overhead of using DSM less than 20%

Parallel Ray Casting VSM-MIMD

- "Volume rendering on a scalable shared memory MIMD Architecture" Nieh et al. [6]
- Environment
  - Stanford DASH machine. Up to 64 high performance RISC processors
  - Physically distributed memory providing a single address space logical view for the programmer. Processors are grouped into clusters
  - Each processor has a local cache which forms part of the larger global memory. If memory requests can't be handled locally they are referred to other processors in the local cluster and then to other clusters
  - The DASH supports explicit placement of data unlike the KSR

Parallel Ray Casting VSM-MIMD

- Parallel Implementation
  - Distribute pages of the data in a round robin fashion amongst the processors. This ensures no data hotspots
  - Adopt a task queue image partition scheme. Each processor assigned a block of image tiles. When its tiles are completed it can grab tiles from other processors



  - Since corner pixels may be required by two different processors a scheme is adopted to avoid duplicating the computation of the corner pixels.

Parallel Ray Casting VSM-MIMD

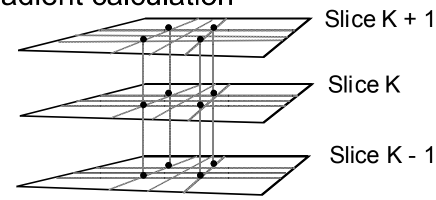
- Summary
  - The scheme shows good parallel efficiencies (over 80% on 48 processors) and is well load balanced
  - It takes advantage of both image coherency and volume coherency
  - For animation sequences inter-frame temporal coherency is also exploited, since neighbouring frames use broadly the same data values which are already in local caches

Parallel Splatting DM-MIMD

- "Volume Rendering on a Distributed Memory Parallel Computer" Elvins et al. [7]
  - Distributed Splatting algorithm on distributed memory nCube2 machine
  - nCube2: 128 processors, 16 Mbytes per processor, host/node approach
  - Recall sequential splatting
    - **Determine in what order the volume will be traversed**
    - **Classify each voxel according to colour and opacity look up tables**
    - **Project (splat) each voxel into image space. Use a reconstruction filter to determine extent of the splat's contribution (footprint)**
    - **Attenuate the colour-opacity tuple with the reconstruction filter**
    - **Composite the attenuated tuples into an image buffer**

Parallel Splatting

- Parallel strategy
  - Object space data decomposition using master-slave parallelism
- Master Process
  - Read dataset from disk and distribute slices to slave processors in round-robin fashion.
  - Actually need to distribute 3 slices for each slice that is to be splatted, since this data is required for the gradient calculation



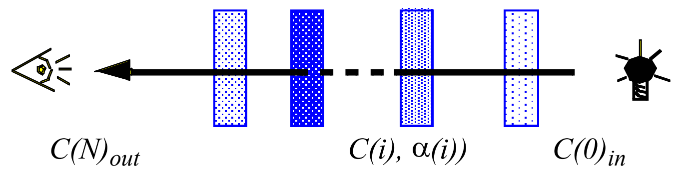
  - Collects Image contributions from each slave and composite in correct order

Parallel Splatting

- Slave Processors
  - Slaves perform the splatting operation on the slices assigned to them
  - Image contributions are then passed back to the master for composition
  - If more slices still to be rendered then obtain more slices from master
- Performance Optimizations
  - Split the distribution and composition roles of the master to avoid a bottleneck
  - *Image Coherency*. Each slave calculates the non-black sections of the image contributions it generates and just passes these back. Saves on communication and reduces the composition task
  - *Distribute groups of slices*. Instead of sending out groups of 3 slices, send out larger groups depending on memory capacity. Reduces short communications and also unchokes the image contribution bottleneck.

Parallel Compositing

- Porter and Duff "Over" operator widely used for compositing, or "alpha blending"

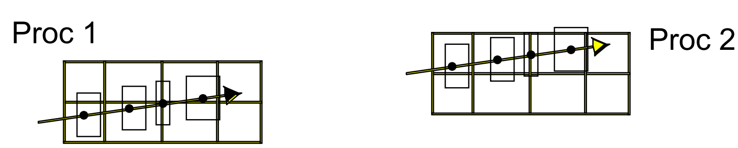


- For a sample location S(i), with colour C(i) and opacity a(i), then

S(i)over S(j) = S(I) + (1- alpha(I))S(j)

"Over" Operator

- The "over" operator is associative
- This means that samples which have been computed by different processors can be composited if the correct order is maintained

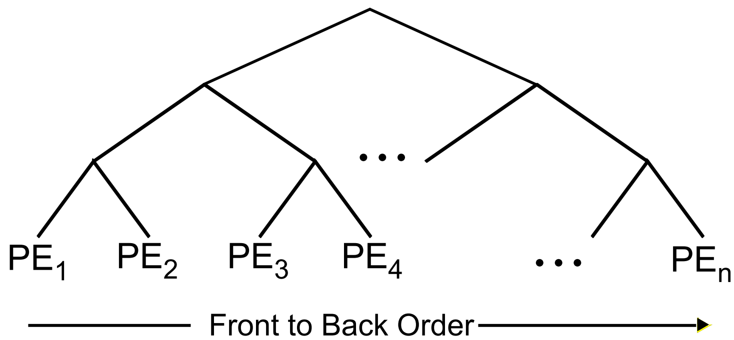


$S(P_1) = S(1)over\ ...S(k)$ $S(P_2) = S(K+1)over\ ...S(N)$

$S(P) = S(P_1)\ over\ S(P_2)$

Binary Compositing

- Data required for a particular section of the image is distributed amongst the processors
- Data must be re-composed in the correct order



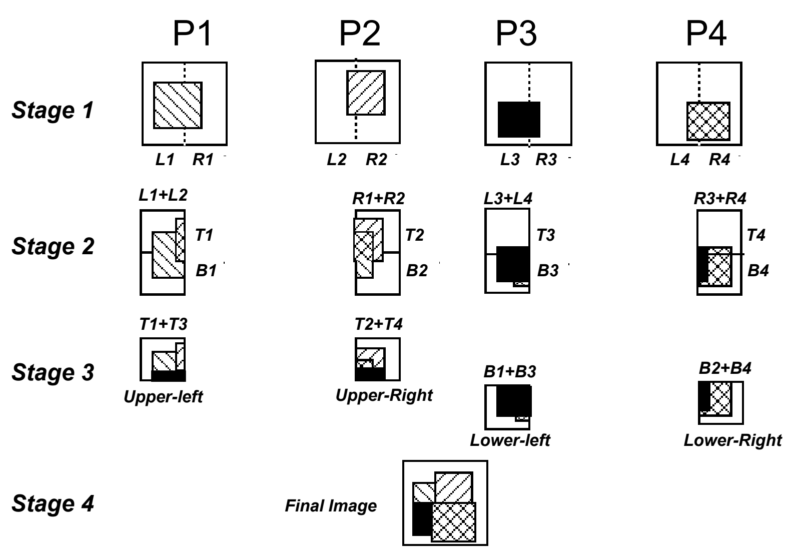
Binary Compositing

- Composite data at each stage of the tree starting at bottom
- At each stage half the processors become idle after they pass their data on.
  - Poor load balancing
- Final compositing step is done by one processor
  - a significant bottleneck
- Better to adopt a parallel approach, where each processor is kept busy until the final stage. e.g. Painter and Hansen's Binary Swap Compositing[8]

Binary Swap Method [8]

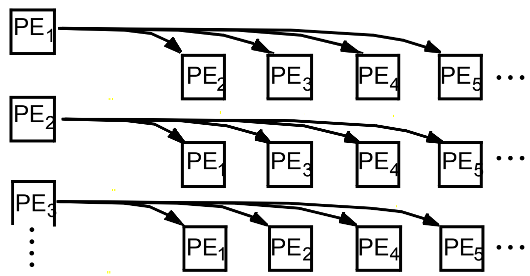
- At each stage of the compositing tree, swap half of the PE's image with a partner
- At each stage the size of the image at each PE becomes smaller
- All processors are active throughout the composite phase

Binary Swap Compositing



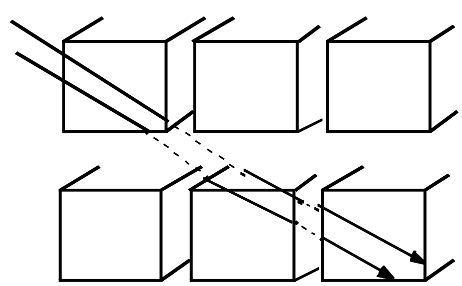
Direct Send Compositing[9]

- Split the tasks of rendering and compositing. For each rendering processor allocate a compositing processor.
  - These may or may not be the same processors that are doing the rendering
  - Subdivide the image
  - After the rendering process completes all compositing is performed in parallel
- Maintains load balance



Ray Dataflow Approach

- If ray dataflow has been used as the parallelization method, then composite as the rays progresses



- Alternatively wait until all processors have rendered their sub images and then propagate a ray to do the compositing as a post process

Ray Dataflow Approach

- Observations
  - Integrated with the rendering stage
  - If done as a post process, then a relatively small number of messages
- Disadvantages
  - Same as ray dataflow approaches i.e not scalable
  - means that some processors are idle waiting for rays to progress to them

Manchester Computing

Hardware Volume Rendering

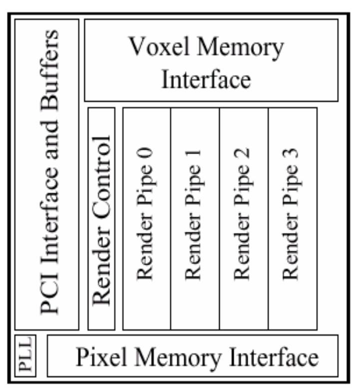
THE UNIVERSITY of MANCHESTER

Making it go faster

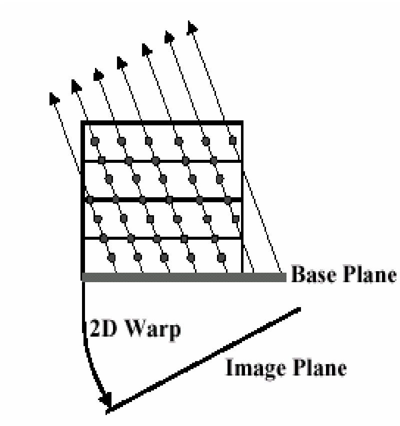
- Hardware Implementations include:
  - VolumePro Card
  - Texture Mapping
  - Programmable Consumable Graphics Hardware
    - NVIDIA GeForce4
    - ATI Radeon 8500

MERL VolumePro 500

- PC PCI Card
- Renders a 256^3 Volume in Real Time
- 128 MB Volume Memory
- On-fly gradient estimation
- 500 million interpolated, Phong-lit, composited samples/s
- Cost: $2,995

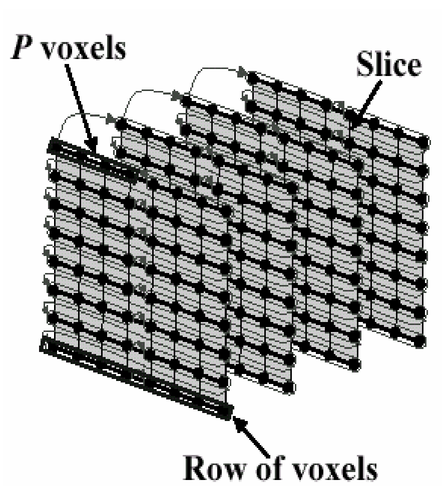


Object Order Raycasting



- Cast Rays from base plane (not image plane)
- Resample dataset on slices perpendicular to main viewing direction
- Final 2D warp onto image plane

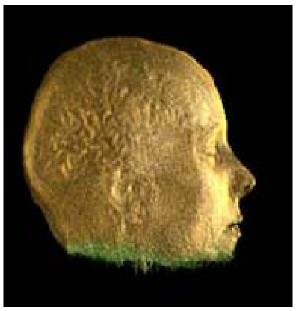
Slice by Slice Processing



- Voxels are processed a slice at a time
- The slices are parallel to the base-plane
- Within a slice, voxels are processed a row at a time
- Within a row, *P* voxels are processed in parallel, where P is the level of parallelism

Advantages

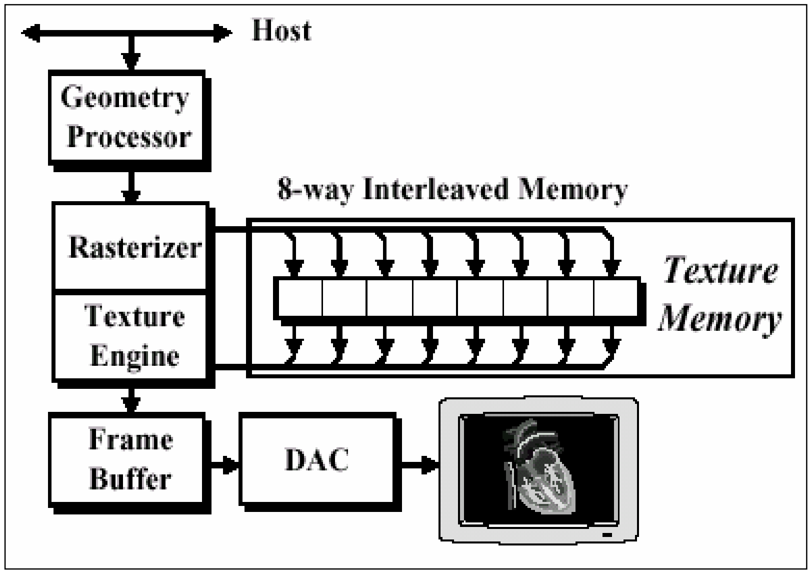
- High image quality, real-time performance
- Full ray-casting pipeline in hardware
- Compact, inexpensive design



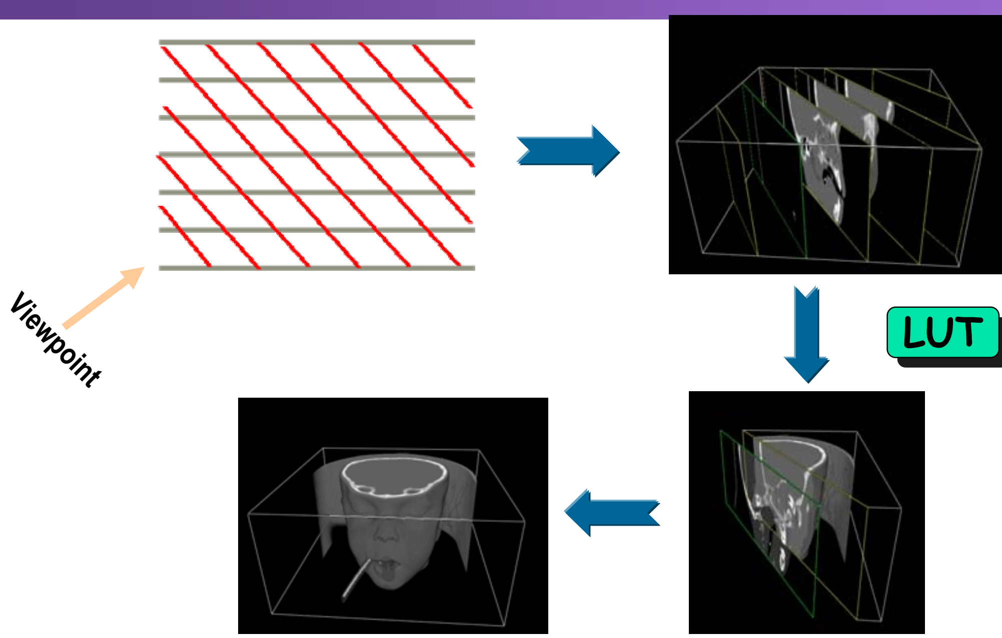
Disadvantages

- No integration with polygons (yet)
- No perspective projections (yet)
- Narrowly focused on volume graphics
- Consumer graphics cards are catching up.

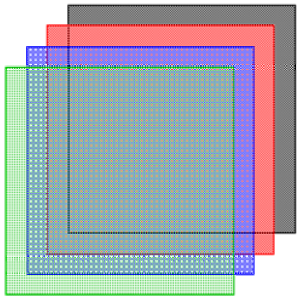
3D Texture Hardware



3D Texture Mapping



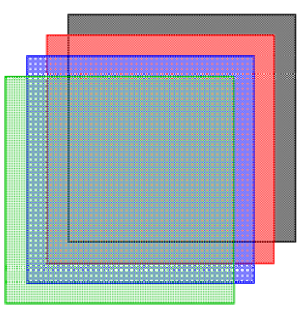
Compositing in OpenGL



Draw I_1
Draw I_2 & Blend with I1
Draw I_3 & Blend with Framebuffer

Draw I_N & Blend with Framebuffer

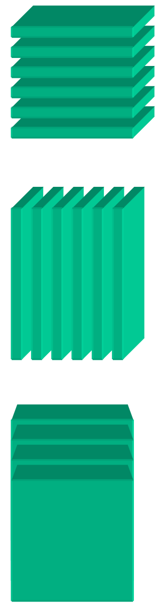
OpenGL Source



```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
            GL_ONE_MINUS_SRC_ALPHA);
glColor4f(1.0, 0.0, 0.0, 0.75);
glRectf(0.0, 0.0, 1.0, 1.0);
glColor4f(0.0, 0.0, 1.0, 0.6);
glRectf(0.0, 0.0, 1.0, 1.0);
```

2D Texture Mapping

- Need 3 copies of the data.
- Viewpoint position determines which set to use.



XY Slices

YZ Slices

XZ Slices

In OpenGL

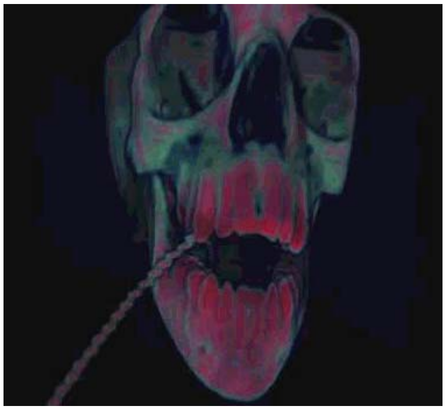
```
for  ( t=0;  t < depth ;  t++ )   {
      glBindTextureEXT (GL_TEXTURE_2D,t);
      glBegin (GL_QUADS);
           glTexCoord2f (x0, y0) ; glVertex3f (x, -y, -z);
           glTexCoord2f (x1, y0) ; glVertex3f (x, -y, z);
           glTexCoord2f (x1, y1) ; glVertex3f (x, y, z);
           glTexCoord2f (x0, y1) ; glVertex3f (x, y, -z);
      glEnd ();
      x += step;
}
```

Implementation Notes

- Brick Sets
  - Sub-divide voxel data to fit into texture memory
  - application controls paging
- Perspective Projection
  - Sample with concentric spherical shells

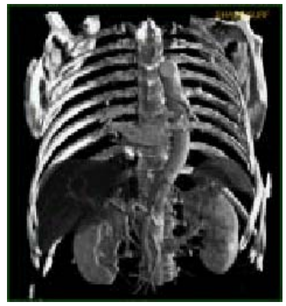
Advantages

- Graphics Hardware gives interactivity
- Can mix with geometric models
- Moving to the PC Domain



Disadvantages

- Images can be "fuzzy"
- Some things are still missing
  - Gradient calculations in hardware



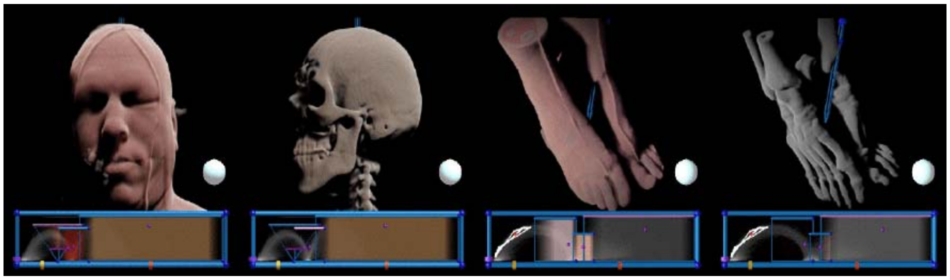
No Shading
1 pass (11 frames/s)

Central Differences
10 pass (1 frames/s)

Consumer Programmable Graphics

- Latest generation graphics cards provide a programmable pipeline for:
  - Vertex processing stage
  - Rasterisation stage – can be exploited for volume rendering
- Fragment shaders
  - Written in assembler and downloaded to hardware
  - NVIDIA use two stages:
    - Texture shaders
    - Register Combiners
  - ATI combine texture fetch and colour combination in one fragment shader

Examples

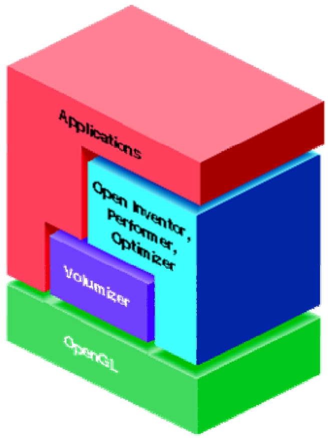


Limitations

- Precision of current PC graphics hardware is limited
  - Typically 8 or 16 bit precision only
  - Results in artifacts
- Limited programmability restricts number of operations in one pass of the graphics pipeline
  - Often need to use multiple rendering passes
- Texture memory available is not enough
- Low bandwidth of graphics bus
- Difficult to support all graphics boards

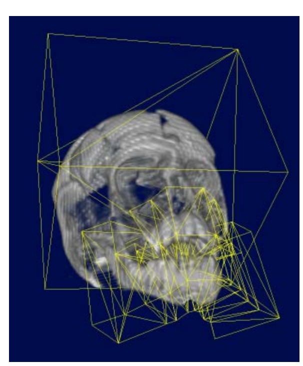
OpenGL Volumizer

- Layered above OpenGL
- Uses Voxels and Tetrahedrons as volume primitives
- Better modeling capabilities:
  - Volume deformation
  - Co-registration
  - Arbitrary VOI Shape
- May be cross platform



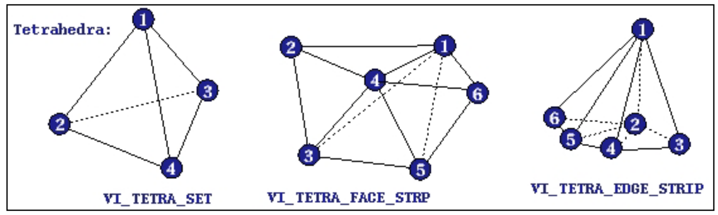
Algorithm Overview

- Subdivide volume into bricks;
- Define volume geometry using tetrahedrons;
- Polygonize geometry (slice volume);
- Render resulting face sets.

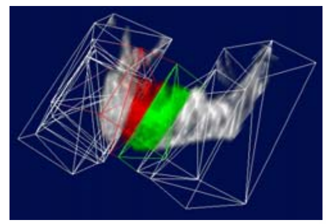


Define Geometry

- Use Tetra Sets

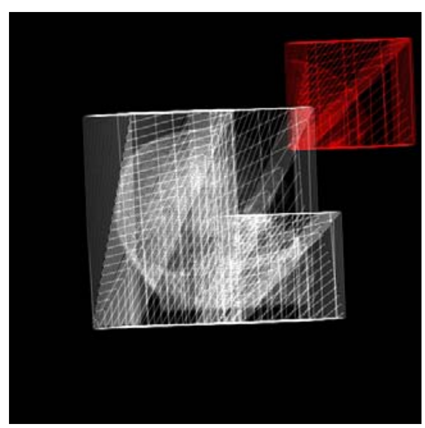


- Define Voxel Coordinates
- Define Texture/Colour/User Coordinates
- Define Indices
- Define IndexSet



Polygonize (Slice)

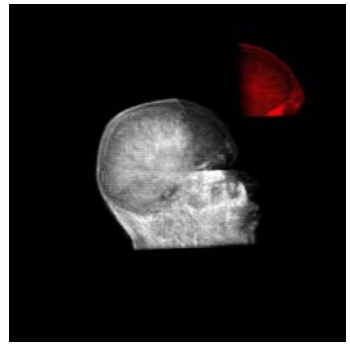
- Set samplingMode
- Set samplesNumbers
- Set samplingSpace



```
int polygonize(
        voIndexedTetraSet *aTetraSet,
        voBrickSet *aBrickSet,
        GLdouble modelMatrix[16],
        GLdouble projMatrix[16],
        voSamplingMode smaplingMode,
        voSamplingSpace samplingSpace,
        int &samplesNumber,
        float samplingPeriod,
        voIndexedFaceSet ***aIndexedFaceSet
);
```

Render Face Sets

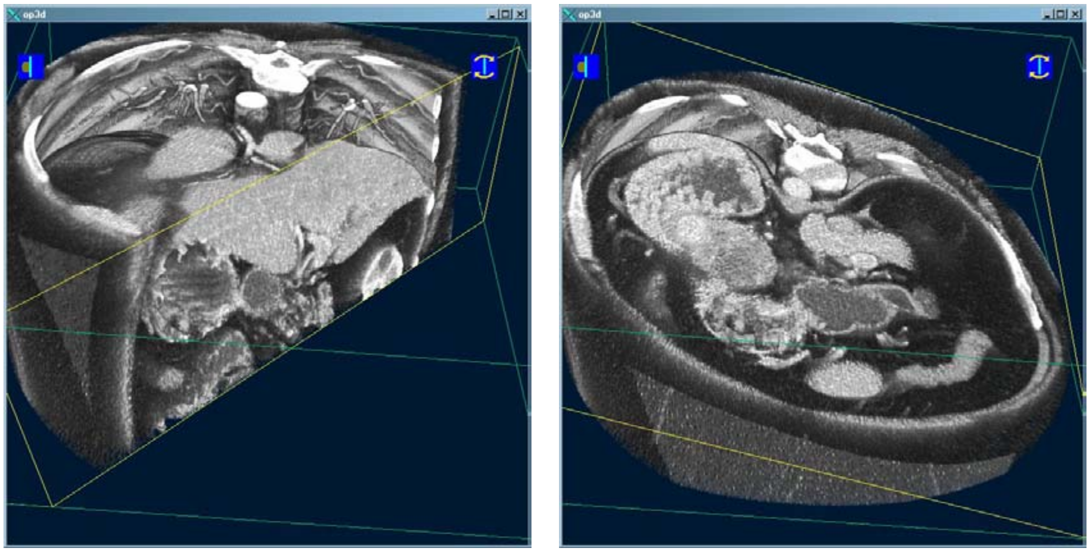
- Enable texture mapping
- Set blend function
- Loop over sampling planes:



```
voGeometryActions::draw(
        aPolygonSetArray[brickSortedNo][binNo],
        interleavedArrayFormat );
```


Op3D Project

OpenGL Volumizer in the Operating Theatre



CT Data: 512 x 512 x 203 slices

THE UNIVERSITY of MANCHESTER

Manchester Computing

Detailed Example

THE UNIVERSITY of MANCHESTER

Visualisation of Orion Nebula

Courtesy of San Diego Supercomputing Centre

Orion Nebula

Hubble Telescope image of the Orion Nebula

- Enormous cloud of dust and gas 1500 light years from earth
- Clumps of higher density gas form and grow – birthplace of stars (proplyds)
- High energy photons from stars ionize gas atoms, knocking electrons from their orbit. Electrons emit light as they return to former orbit.

3D Structure of Orion Nebula

- Ionization front, derived from infrared and visible light observation
- Surface glows from the influence of the proplyds above
- Ionization model is extrapolated outward to include surrounding regions

Voxelisation

- Distance Fields used
- To create a soft gaseous layer, the fields distances were used to vary opacity
- A 3D procedural turbulence function was applied to give the gaseous layer a rougher, more turbulent look

Compositing Formula:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i)\alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j))$$

Where α is opacity. If $\alpha=1$, light is blocked from further samples along the ray.

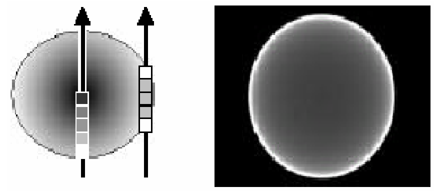
Modified version:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i)\alpha(i)\prod_{j=0}^{i-1}(1-\beta(j))$$

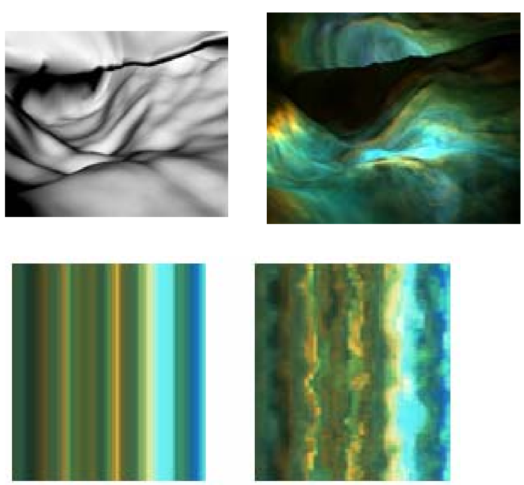
Where α is emissivity, ß is opacity (absorptivity). Models radiative transfer – suitable for glowing gas.

Opacity and Emissivity

- Voxels in open space outside of the nebula are transparent and empty of glowing gas.
- Voxels within the glowing parts of the nebula are given low opacity and high emissivity
  - If surrounding a dark opaque core, such a shape appears to have a halo
- Dark dust is given high opacity and low emissivity



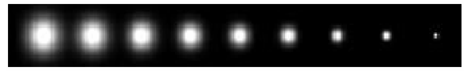
Shading the Nebula



- Hubble images encode both colour and intensity
- Texture mapping used instead of calculating lighting effects
  - Projected through volume on a vector from Earth
- Texture coordinates are jittered (using procedural noise) to prevent striping effects

Visualising the Stars

- Stars modelled as Gaussian spots
  - Radius and brightness varied with stars apparent magnitude

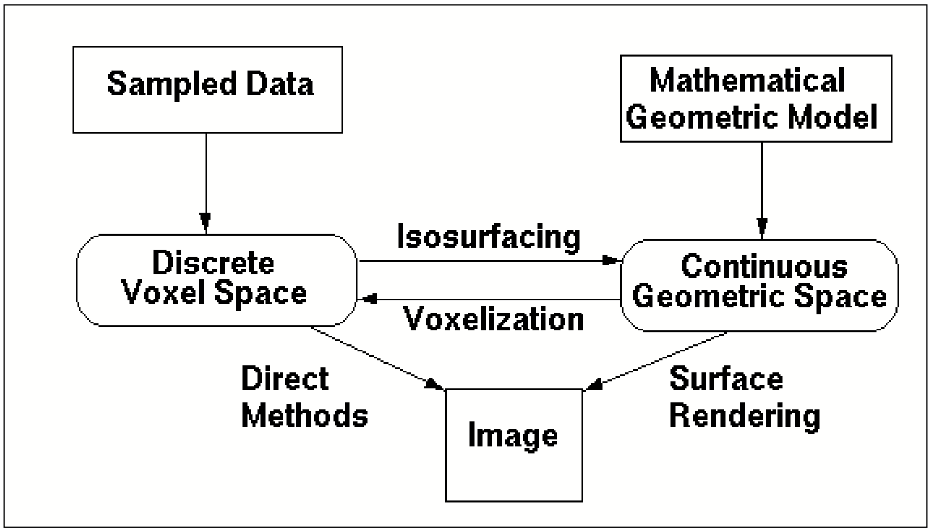


More Information

Visualizing Stars and Emission Nebulae
David R. Nadeau, Jon D. Genetti, Steve Napear Bernard Pailthorpe, Carter Emmart, Erik Wesselak, Dennis Davidson, Eurographics 2000 Short Presentations

A 2½ minute fly through animation was produces for the Haydon Planetarium, New York. Seven projectors are used to seamlessly cover the interor of the planetarium's dome.

Summary



Manchester Computing

References

THE UNIVERSITY of MANCHESTER

References

[1] J. Bentley, "Multidimensional Binary Search Trees used for Associative Searching", Comm. ACM 18,8 (Sept. 1975)

[2] K. Brodlie, J. Wood, “Recent Advances in Volume Visualization”, Computer Graphics Forum, 2001

[3] G. Cameron, P. Underill, "Rendering Volumetric medical Image data on a SIMD Architecture Computer", Proc. of the Third Eurographics workshop on Rendering, 1992

[4] J. Challinger, “Parallel Volume Rendering on a Shared Memory Multiprocessor", Technical report UCSC-CRL-91 -23, University of California, Santa Cruz,1992

[5] B. Corrie, P. Mackerras, "Parallel Volume Rendering and Data Coherence", IEEE Proceedings of 1993 Parallel Rendering Symposium

[6] L. Doctor et al., "Display techniques for Octree-Encoded Objects", IEEE CG&A, July 1981

[7] A. Doi, A. Koide, “An Efficient Method of Triangulation Equi-Valued Surfaces by Using Tetrahedral Cells”, ICE Trans Commun. Elec Inf. Syst., E-74(1):214-224, 1991

[8] T. Elvins, “Volume Rendering on a Distributed Memory Parallel Computer", Proc.Vis’92, Boston

[9]T. Ford, A.Grant, "Volume Rendering on the Computing Surface", Proceedings of the Parallel Computing & Transputer Applications Conference, Barcelona, 1992

[10] I. Foster, "Designing and Building Parallel Programs", Addison-Wesley, 1994

[11] A.Grant, W. Haslam, "Marching Cubes using Virtual Shared Memory", Technical Report MVC, University of Manchester

[12] A.Grant, M.Zuffo, "Approaches to Direct Volume Rendering on Distributed Memory and Virtual Shared Memory Parallel Computers", Proceedings of BCS Conference on Parallel Processing for Graphics and Scientific Visualization, Edinburgh, May 1993

[13] S. Green, "Parallel Processing for Computer Graphics", Pitman, London 1991

[14] S. Green, D. Paddon, "Exploiting Coherence for Multiprocessor Ray Tracing", IEEE CG&A 9(6), 1989

[15] M. Gross, K. Joy, R. Hammersley, A. Hubeli, H. Lu, H. Pfister, “Tutorial 4: Multiresolution Techniques for Surfaces and Volumes” IEEE Visualization 2001

[16] R. B. Haber and D. A. McNabb, “Visualization Idioms: A conceptual model for scientific visualisation systems.”, In B Shriver, G M Nielson and L J Rosenblum, editors, Visualisation in Scientific Computing, pp74-93, IEEE, 1990.

[17] C. Hansen, P. Hinker, "Massively Parallel Isosurface Extraction", Proceedings of Vis’92

[18] G. Kindlmann, J. Durkin, “Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering”, Proceedings of IEEE Visualization 2001

[19] J. Kniss, G. Kindlmann, C. Hansen, “Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets”, Proceedings of IEEE 1998 Symposium on Volume Visualization

[20] M. Levoy, "Efficient ray Tracing for volume data", ACM Transactions of Computer Graphics 9(3), July 1990

[21] K. Ma, J. Painter et al., "A Data Distributed, Parallel Algorithm for ray Traced Volume Rendering", Proceedings of the 1993 Parallel Rendering Symposium, San Jose, IEEE

[22] M. Meissner, J. Huang, D. Bartz, K. Mueller and R. Crawfis, “A Practical Evaluation of Four Popular Volume Rendering Algorithms”, ACM Symposium on Volume Visualization, 2000

[23] C. Montani, R. Perego et al., "Parallel Volume Visualization on a Hypercube Architecture", Proceedings of the 1992 Workshop on Volume Visualization, 9-16, Boston

[24] U. Neumann, "Parallel Volume Rendering Algorithm Performance on Mesh Connected Multicomputers", Proceedings of the 1993 IEEE Parallel Rendering Symposium, San Jose 1993

[25] J Nieh, M Levoy "Volume Rendering on Scalable Shared Memory MIMD Architectures", Proceedings of the 1992 Workshop on Volume Visualization

[26] V. Pekar, R. Wiemker, "Fast Detection of Meaningful Isosurfaces for Volume Data Visualization", Proceedings of IEEE Visualization 2001

[27] H. Pfister, M Zicker, J van Baar, M Gross, "Surfels: Surface Elements as Rendering Primitives", SIGGRAPH 2000

[28] J. Rowland, et al., "A Distributed, Parallel, Interactive Volume Rendering Package", Proceedings of IEEE Vis'94

[29] S. Rusinkiewicz, M. Levoy, "QSplat: A Multiresolution Point Rendering System for Large Meshes", SIGGRAPH 2000

[30] H. Stone "High Performance Computer Architectures", Addison-Wesley, 1993

[31] N.A. Thacker, A. Jackson, D. Moriarty, B. Vokurka, "Renormalised SINC Interpolation", Proceedings MIUA 33-36, Leeds, 1998

[32] S. Whitman, "Multiprocessor Methods for Computer Graphics Rendering", Jones and Bartlett, Boston 1992

[33] M. Zicker, H. Pfister, J. van Baar, M. Gross, "Surface Splatting", SIGGRAPH 2001

[34] M.K.Zuffo, A.J.Grant, "RTV: A Package for the Visualization of three dimensional medical data", Proceedings of SIBGRAPI'93, Pernambuco, Brazil,1993.

THE UNIVERSITY of MANCHESTER

Manchester Computing

Manchester Computing

Europe's Premier Academic Computing Service

www.mc.man.ac.uk
mc-info@man.ac.uk

THE UNIVERSITY of MANCHESTER