Hardware Accelerated Volume Visualization

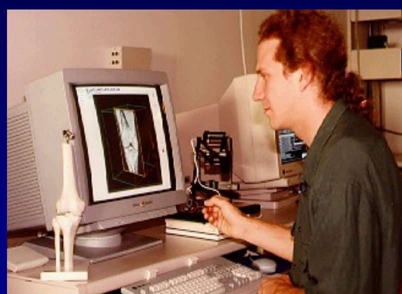
Leonid I. Dimitrov & Milos Sramek
GMI
Austrian Academy of Sciences

A Real-Time VR System

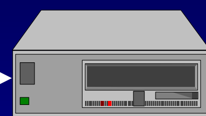
- Real-Time: 25-30 frames per second
- 4D visualization: real time input of data volumes
- High resolution data sets: 512^3, 16 bit
- High image quality: shading, transparency, depth cues
- Interactive parameter changes: lookup tables, classification

Real-Time Data Visualization

- Simulation / visualization

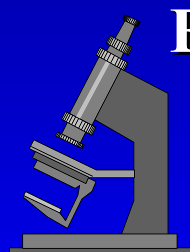


E.g., Surgical Simulation



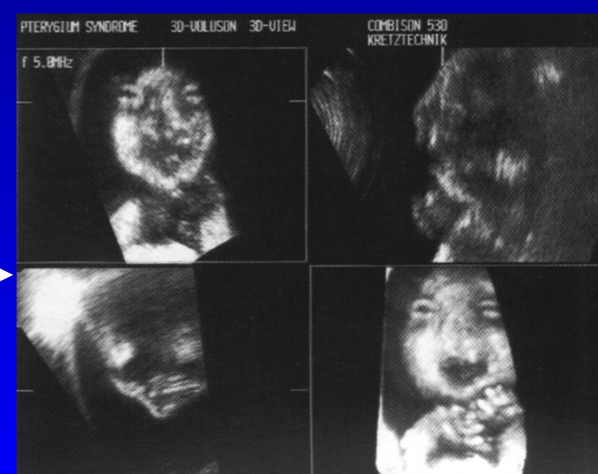
→ Render →

- Acquisition / visualization



E.g., 3D Ultrasound

→ Render →



Processing Requirements

- High demands on storage, processing, and communication of data
- E.g., a 512^3 volume:
  - 2^{24} samples $\times$ 30 instructions $\times$ 30 frames/sec
  - 500 MBytes/sec band-width between processor and memory.
  - 256 MBytes of storage
  - 120 billion instructions per second.

HW Acceleration

- General-Purpose Supercomputers
- Special Architectures
- Graphics Accelerators

General-Purpose Supercomputers

- MIMD (e.g. SGI Challenge - 16 processors, shared memory
- Performance 5-10 fps (Lacroute 1995)
- Drawbacks:
  - very expensive
  - shared among users

Special Architectures

Not specialized on volume rendering (video or polygon processing)

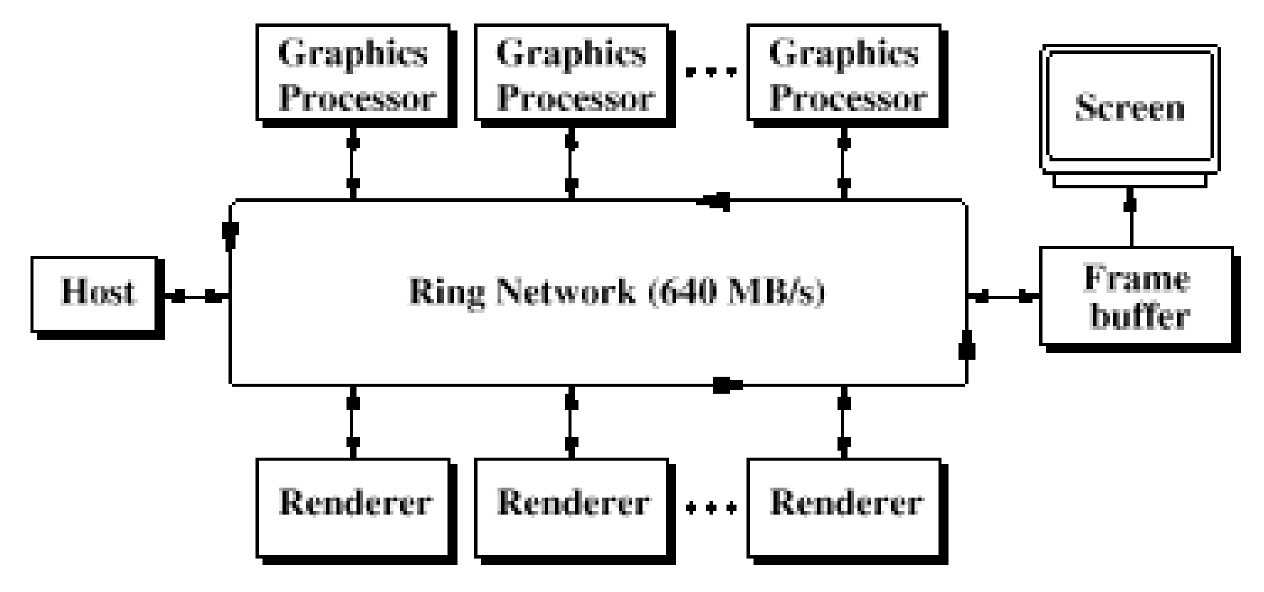
- The PIXAR and PIXAR II Image Computer (1984)
- Pixel-Planes 5 (Fuchs 1989)

PIXAR

- Primary purpose: Visual effects in film industry
- Used for volume rendering (Drebin 1988)
  - Fast volume rotation by shearing.
  - Accumulation along volume rows

Pixel-Planes 5

- Multipurpose system (ray casting, splatting)
- Graphics processors (20) and Renderers (8)
  - 192×192×128 data sets at 11 frames per second
  - Problems with bandwidth

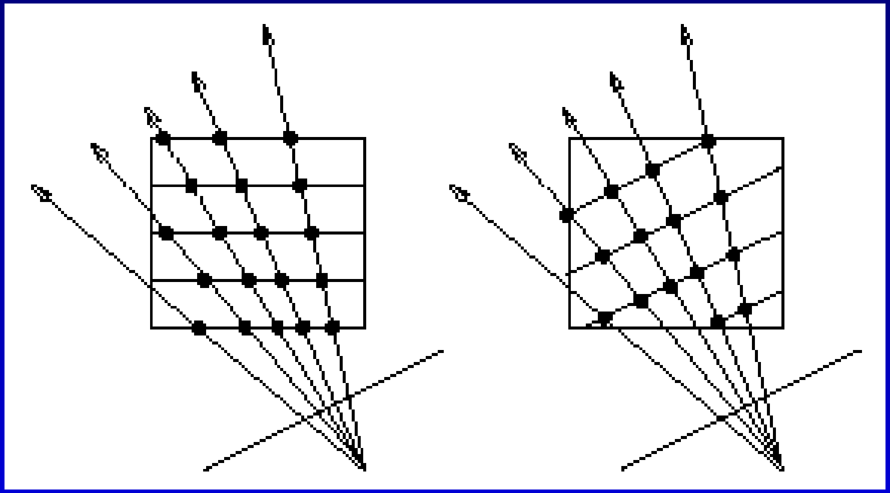


Volume Rendering Accelerators

- PARCUM (Jackel 1985)
  - parallel ray casting
  - 512^3 in about a minute
- The Voxel Processor (Goldwasser 1983)
  - octree scene subdivision
  - hierarchy of rendering an display processors
  - back-to-front rendering of binary data, image-space shading
  - 256^3, 25 fps
- Never built

SGI Reality Engine

- Texture mapping of polygons through 3D texture memory
- Multiple *Raster Manager* boards (16MB textures each)
- Rendering technique: *planar texture resampling*
- 10 fps (512×512×64) Cabral 1994
- No shading



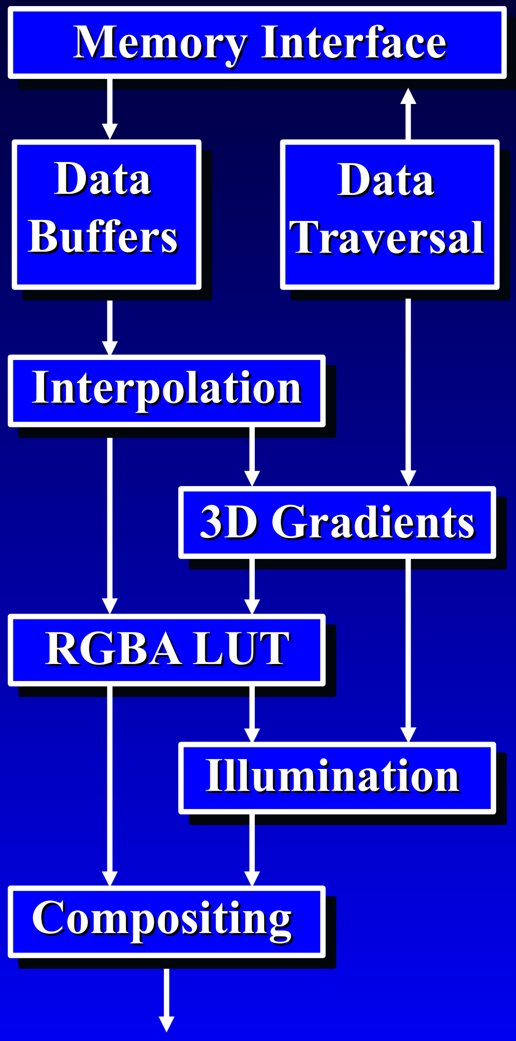
Ray Casting

Planar Texture Resampling

The CUBE Project (Kaufman 1988)

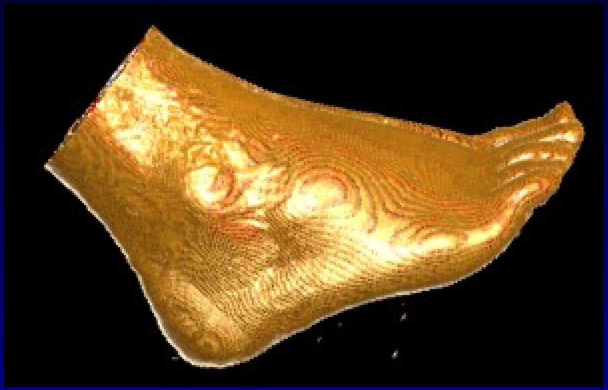
- Based on a *Cubic Frame Buffer* (CFB)
  - linear memory skewing, simultaneous access to a beam of voxels
- Cube 1: orthonormal projections
  - 16^3 data sets, 16 boards
- Cube 2: ditto, VLSI implementation (14000 transistors)
- Resulted in VolumePro (1999)

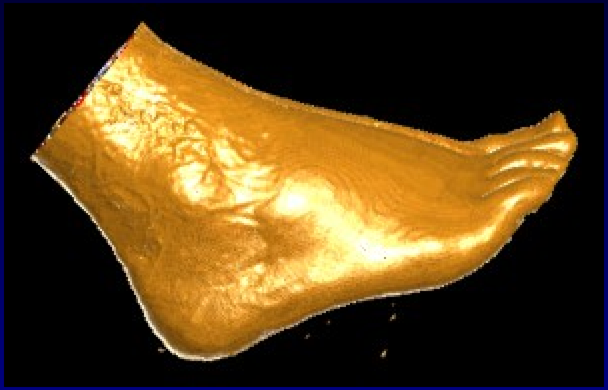
VolumePro: The Ray-Casting Pipeline

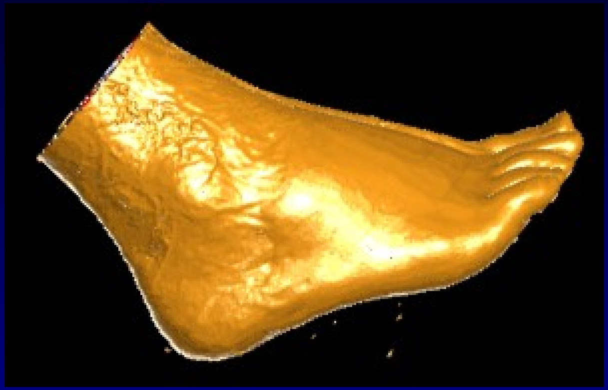


- Data traversal
  - For each pixel, step along a ray
- Resampling
  - Tri-linear interpolation
- Classification
  - Assign RGBA to each sample
- Shading
  - Estimate gradients (normals)
  - Per-sample illumination
- Compositing
  - Blend samples into pixel color

Super-Sampling Along Rays



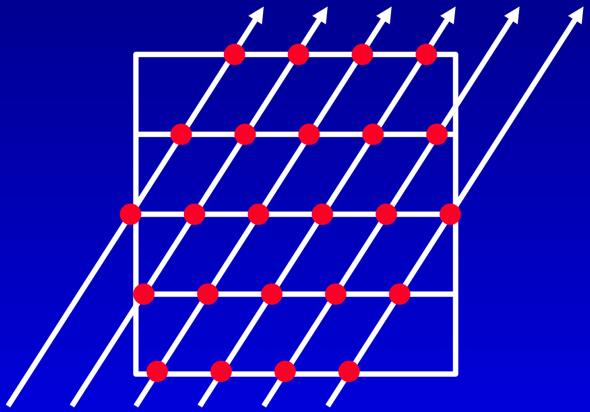


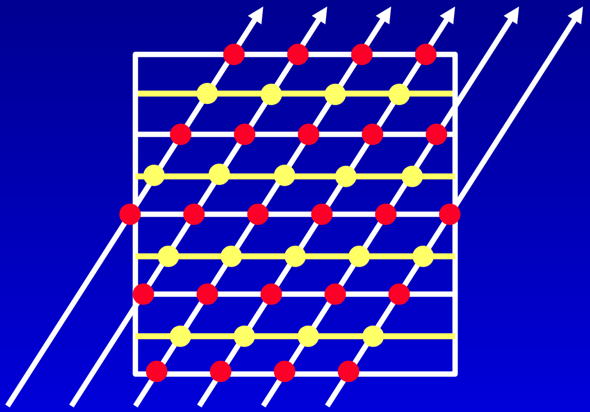


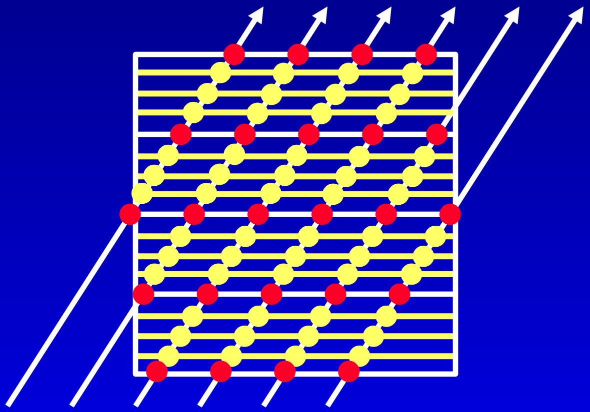
SS = 1

SS = 2

SS = 4

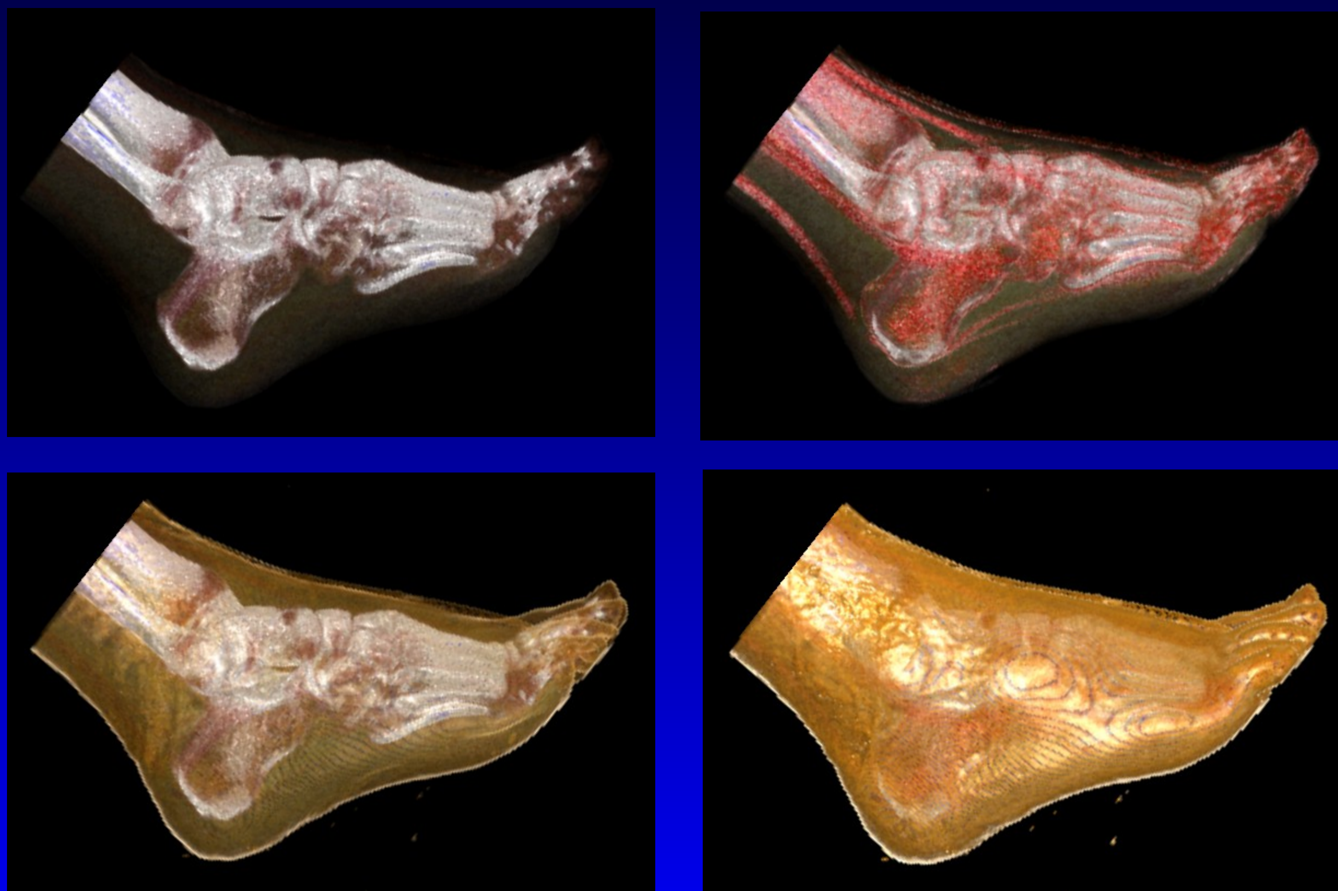






Real-Time Classification

Interactive design of color and opacity transfer functions

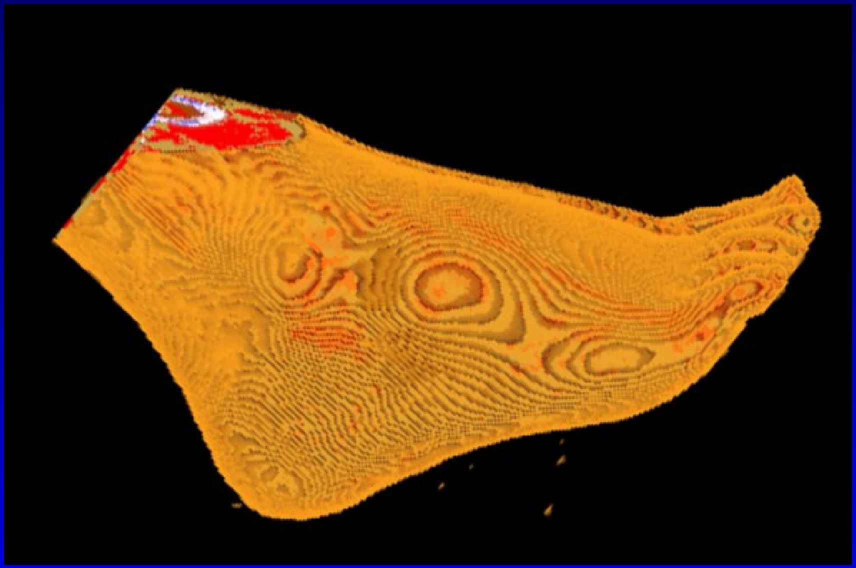


The Phong Illumination Model

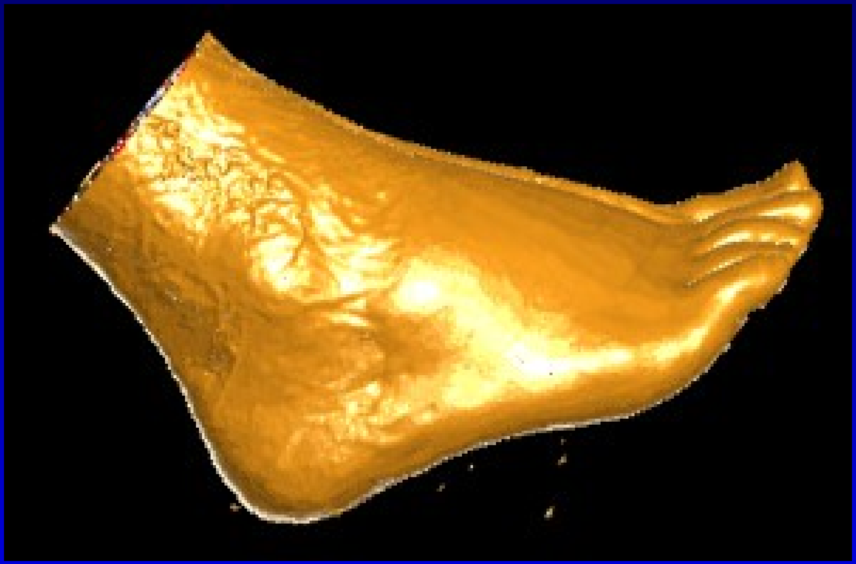
Emissive Diffuse Specular

$$\text{Color} = (k_e + k_d\, I_d)\,\text{SampleColor} + k_s\, I_s\ \text{SpecularColor}$$

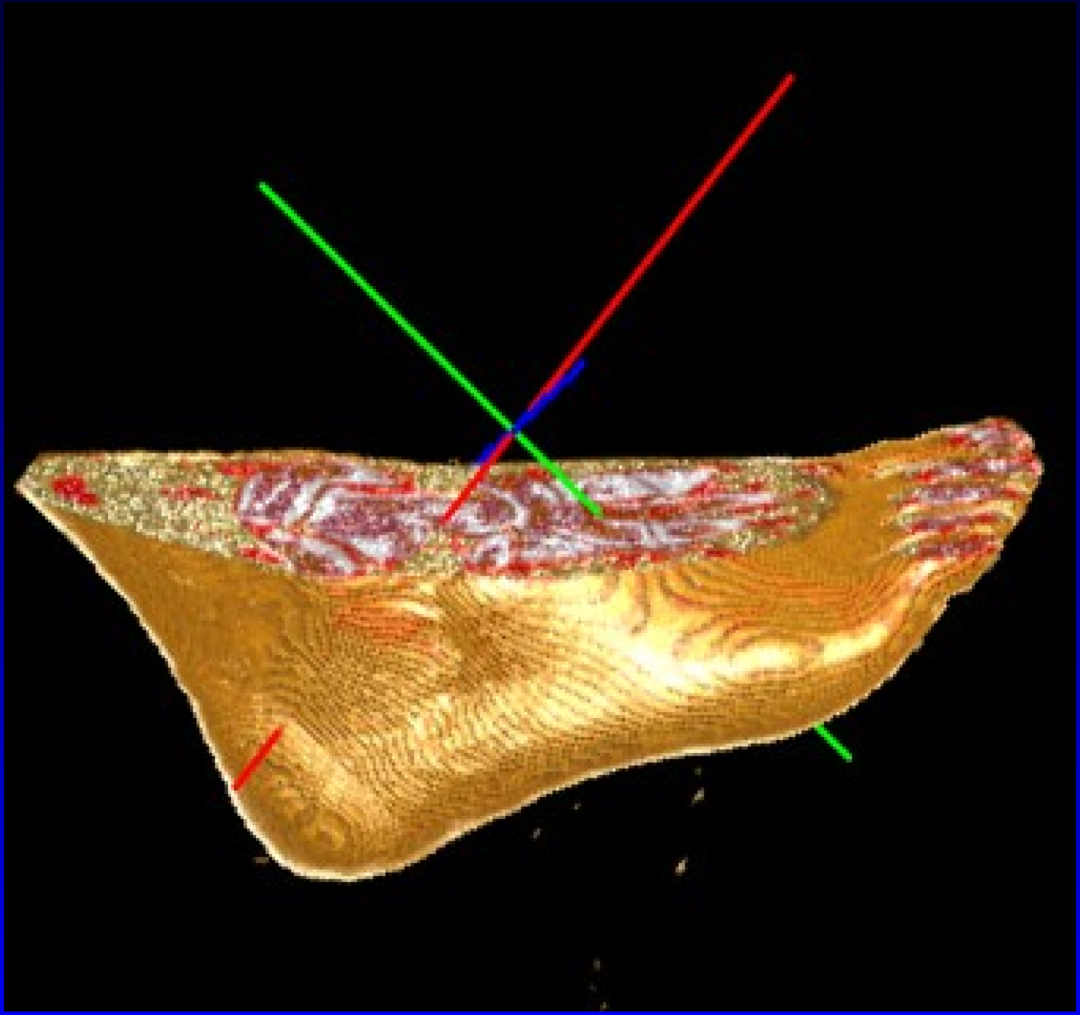
No Illumination



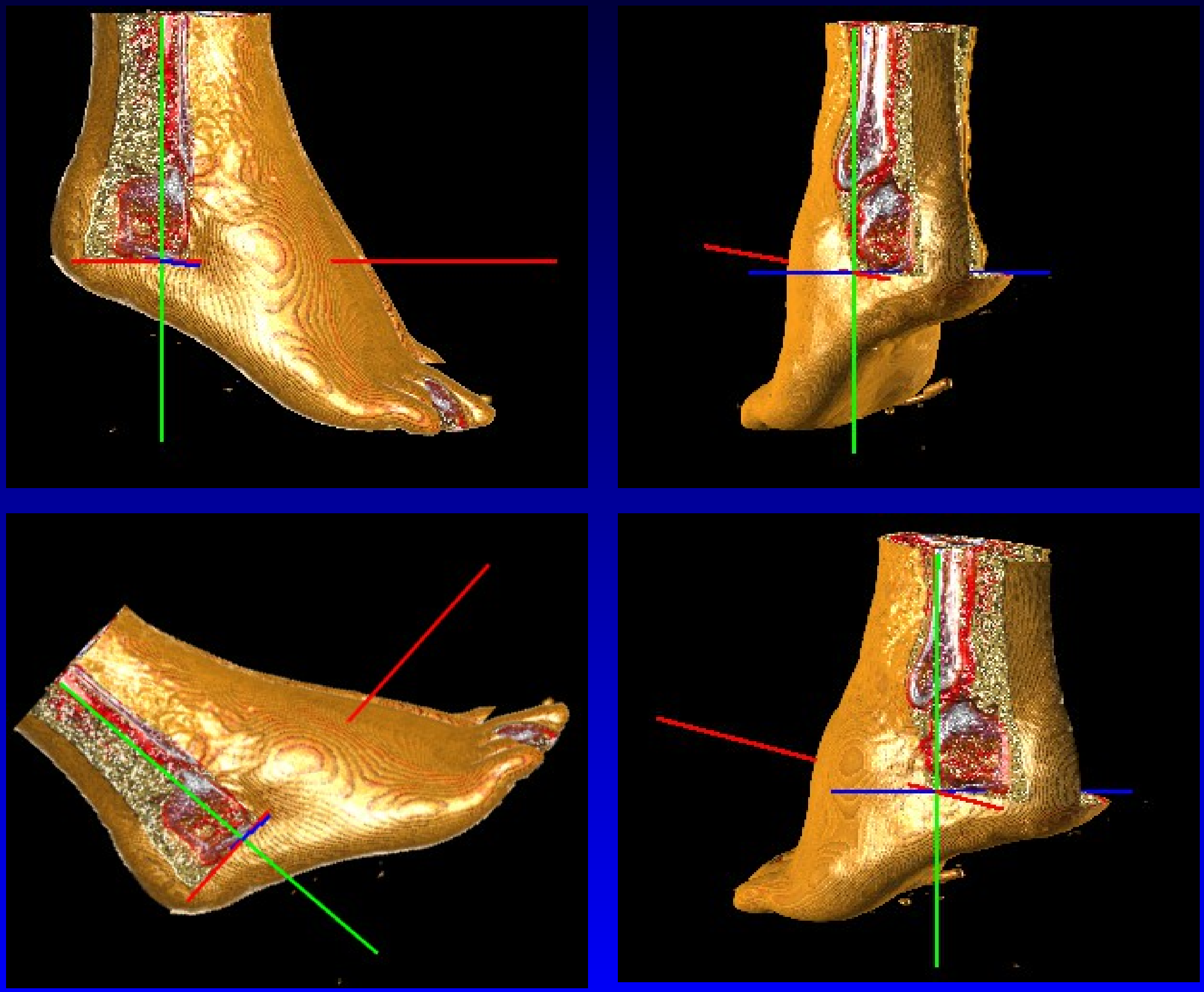
Phong Illumination



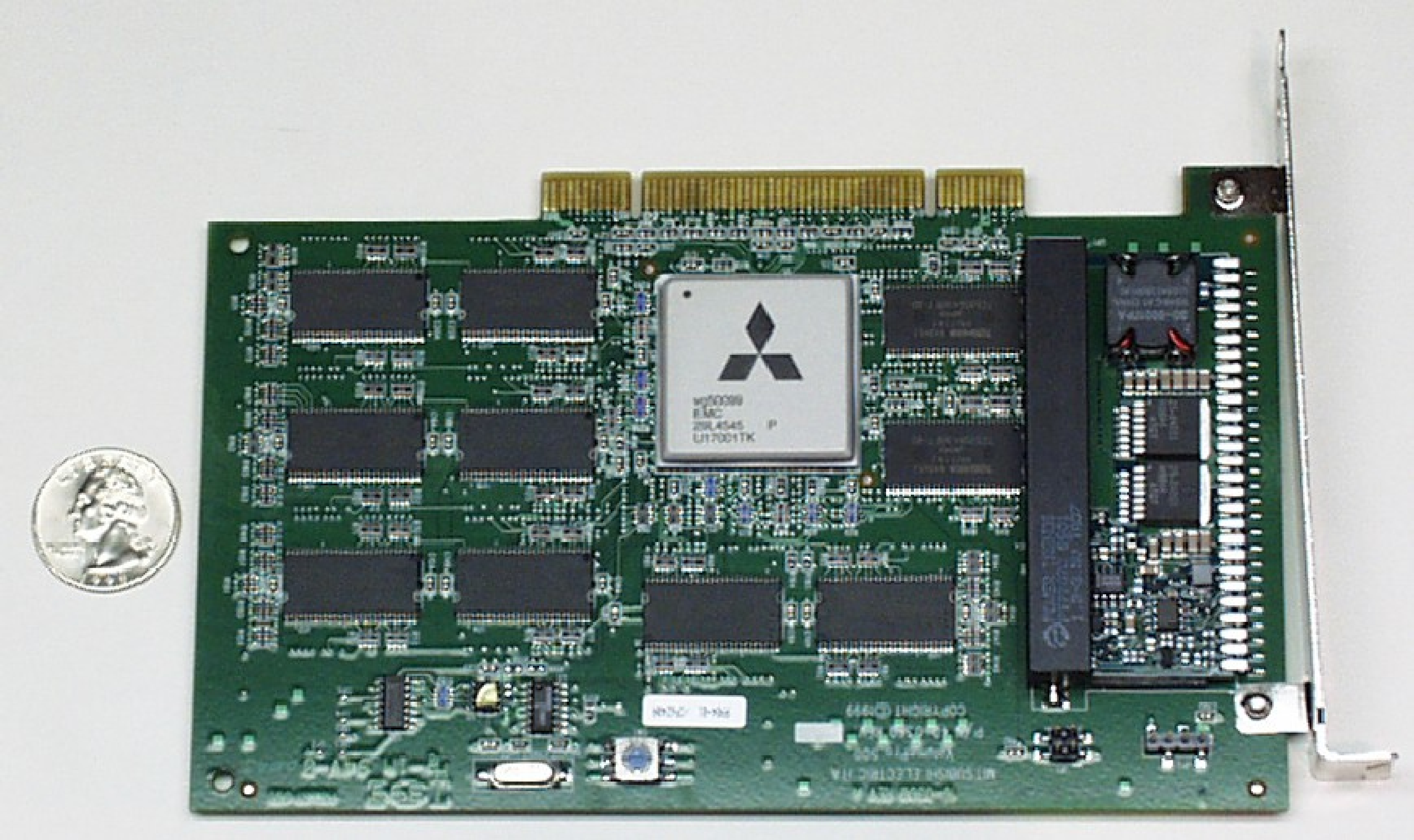
3D Line Cursor and Cut Plane



3D Line Cursor and Cropping



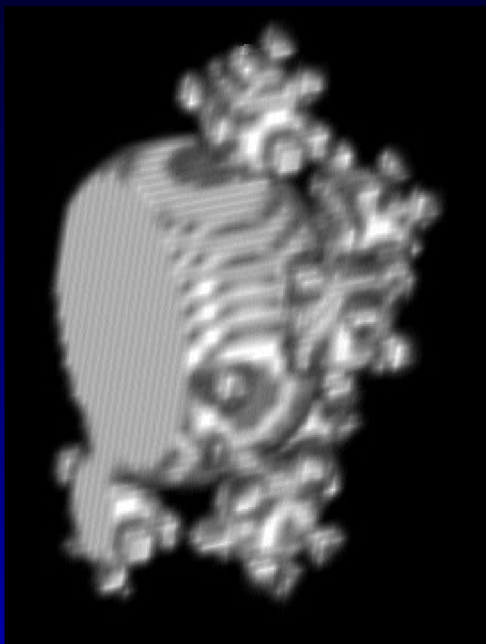
The VG500 Board



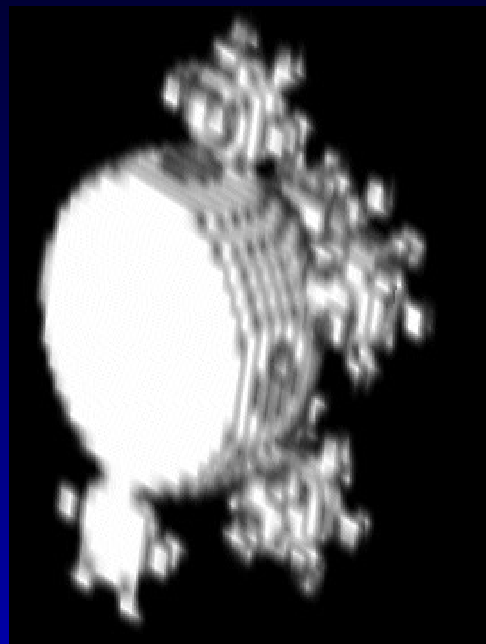
VolumePro 500 Summary

- DVR with trilinear interpolation and Phong sampling
- Future (??)
  - Perspective projection
  - Objects (masking)
  - Overlapping volumes
  - Intermixing volumes and geometry

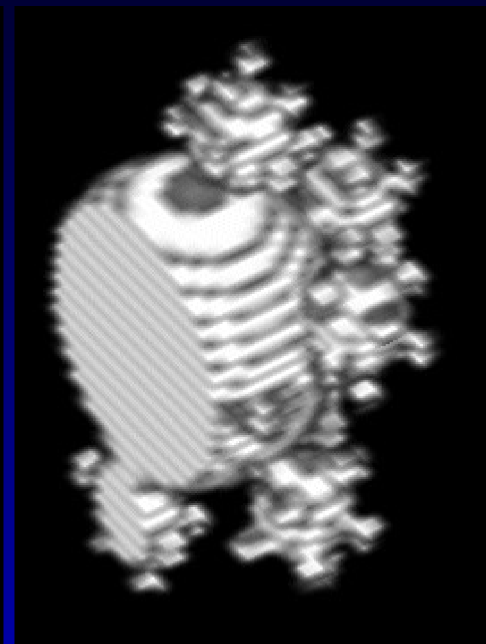
2D Texture Mapping



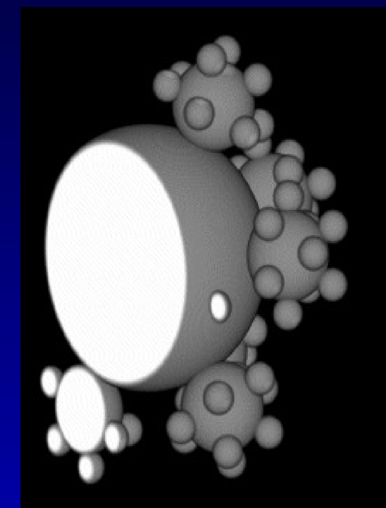
X textures



Y textures



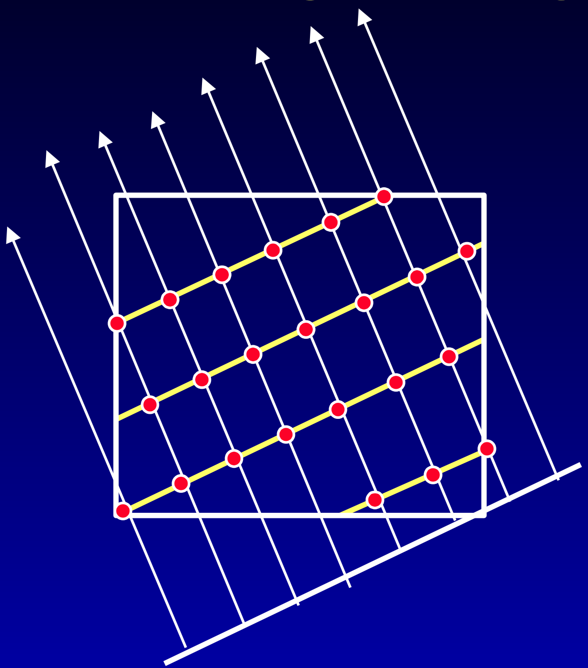
Z textures

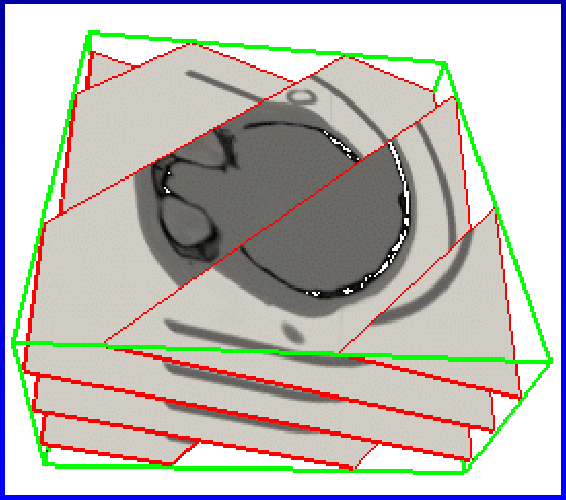


Ray traced

Rendered by VolView on standard 8MB graphic board (1998)

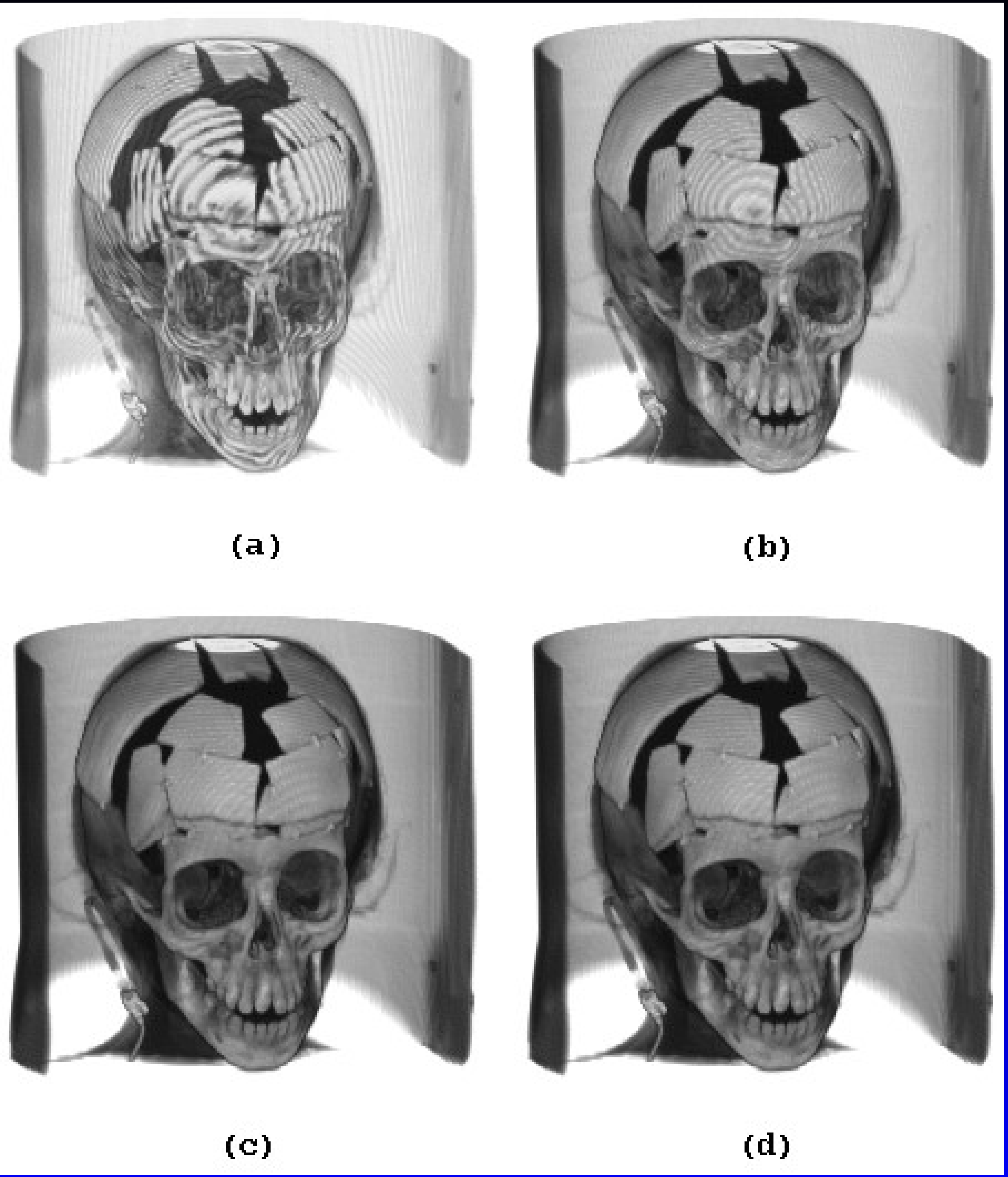
3D Texture-Mapping HW





- Volume is a 3D texture
- Proxy geometry:
  - polygons perpendicular to viewing direction
  - Clipping against volume bounding box
  - Assign 3D texture coordinates to each vertex of the clipped polygons
  - Project back-to-front using OpenGL blending operations
- Originally, no shading!!

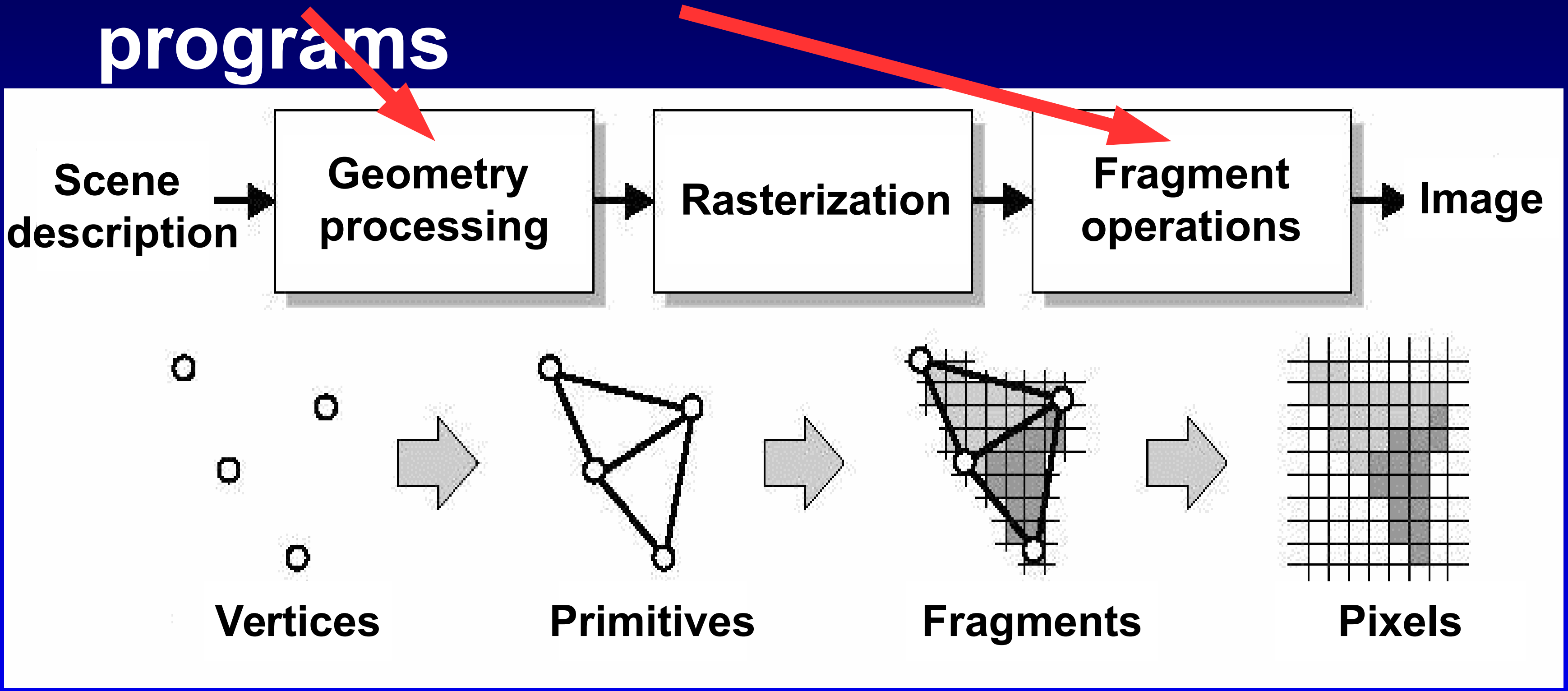
3D Texture Mapping



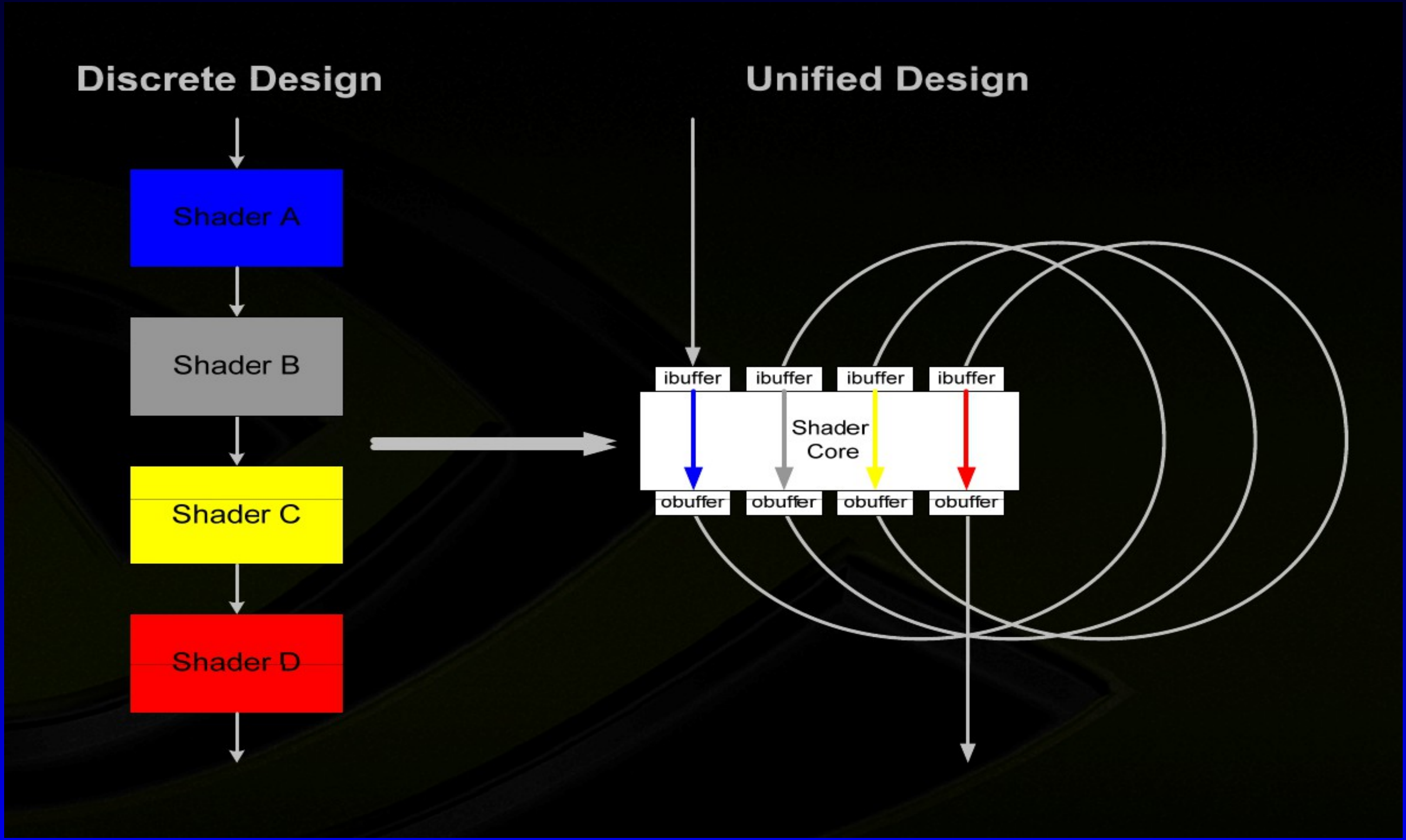
Increasing sampling rate

Programmable Graphics HW

- Nvidia, ATI
- Vertex & Fragment Shaders run programs

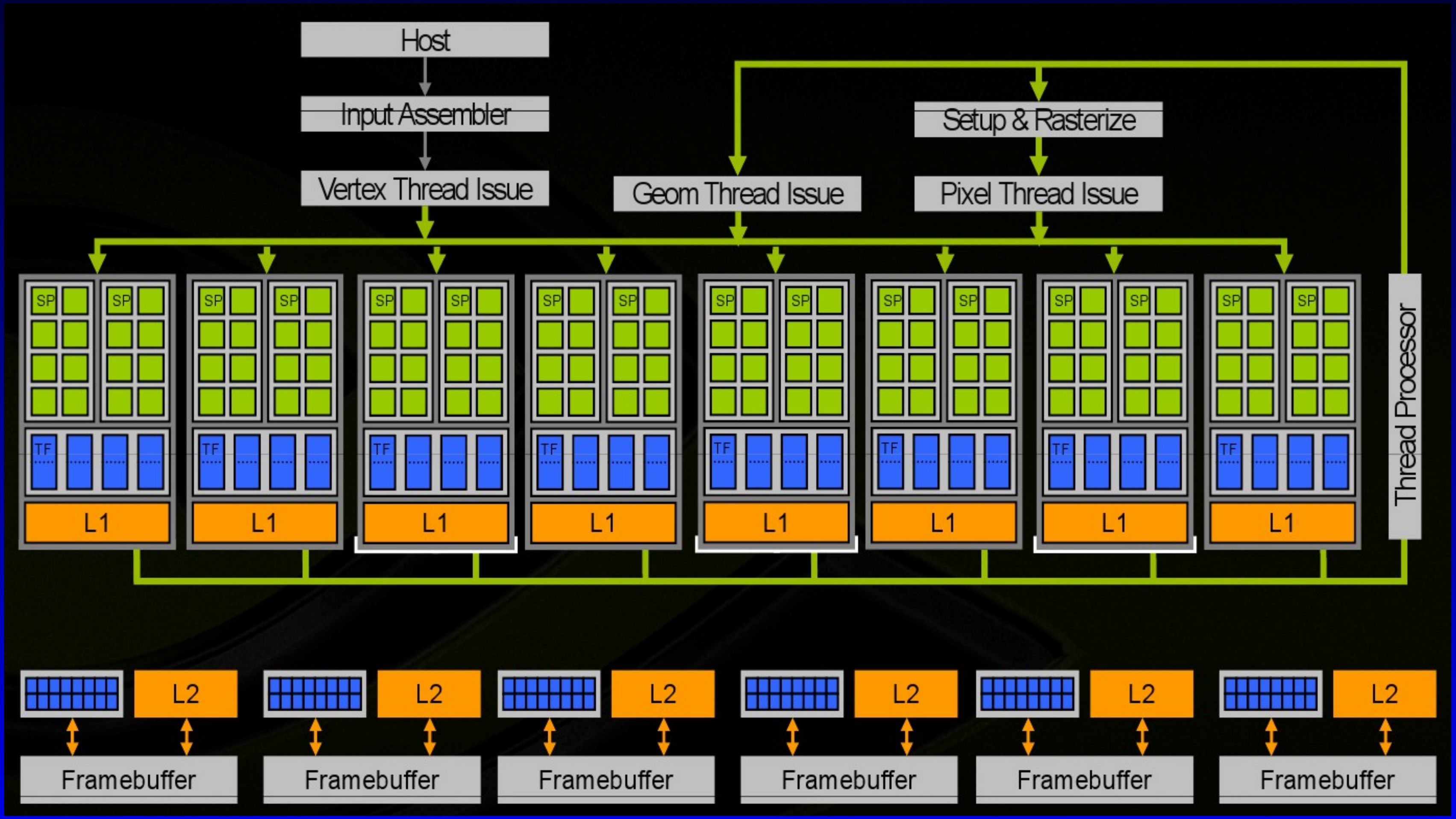


Modern GPUs: Unified Design



Vertex shaders, pixel shaders, etc. become threads running different programs on flexible cores

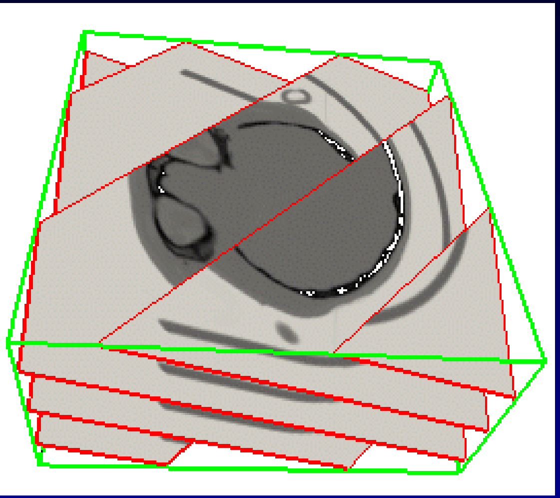
A Modern GPU Architecture



C for Graphics (Cg)

- A C-style language for writing vertex and fragment programs
- On-demand system-dependent compilation
- Significant simplification o HW programing

DVR using HW acceleration

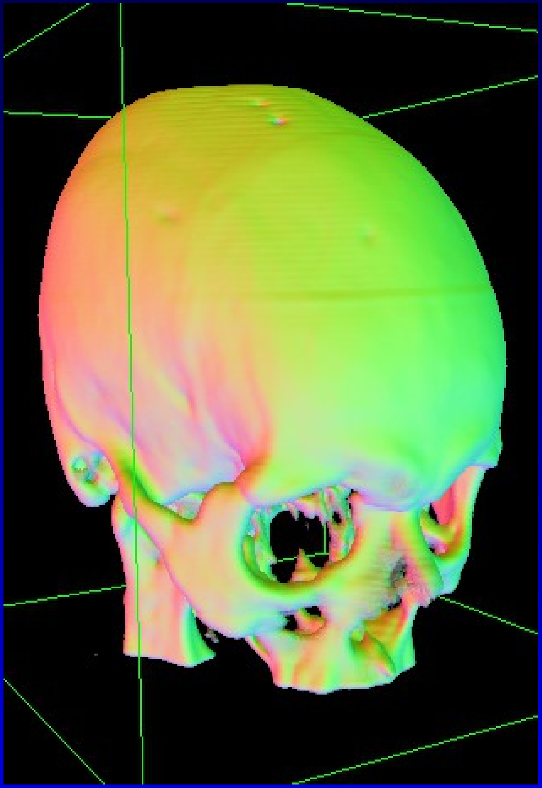


- Proxy geometry based
  - A set o polygons
  - Defines relation between volume (3D texture) and viewing parameters
- Polygons textured by the shading program
- The role of HW
  - Texture interpolation
  - Evaluation of the program
  - Blending of textured polygons

A Cg example (1)

Shading by setting RGB colors to data gradient

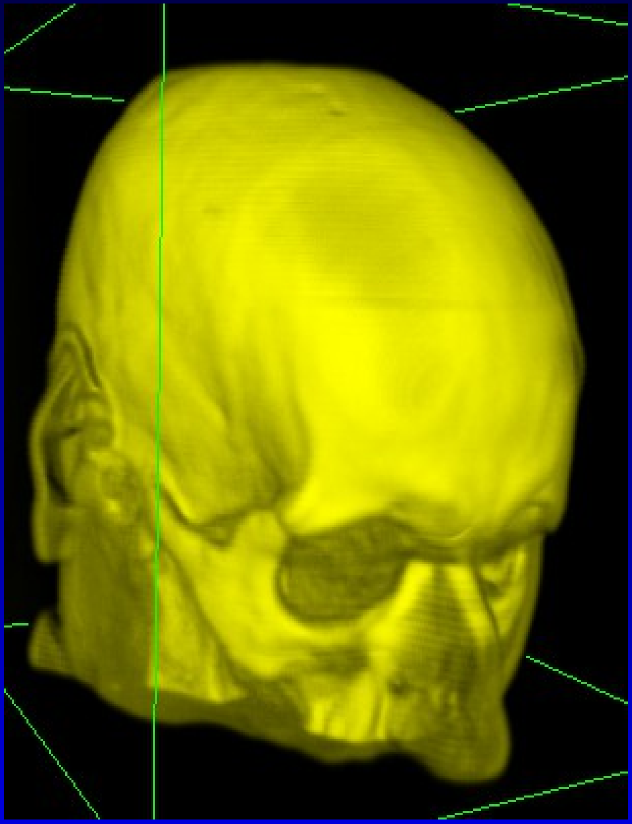
```
output_data main(
            input_data IN,
            uniform sampler3D volume)
{
output_data OUT;
float4 color1 = tex3D(volume, IN.texcoord1);
float3 normal;
float3 t0 = IN.texcoord1; float3 t1 = IN.texcoord1;
t0.x-=K; t1.x+=K;
normal.x = tex3D(volume,t1).r-tex3D(volume,t0).r;
......
normal = normalize(normal);
OUT.color.rgb = normal*0.5+0.5;
OUT.color.a = color1.a;
return OUT;
}
```



A Cg example (2)

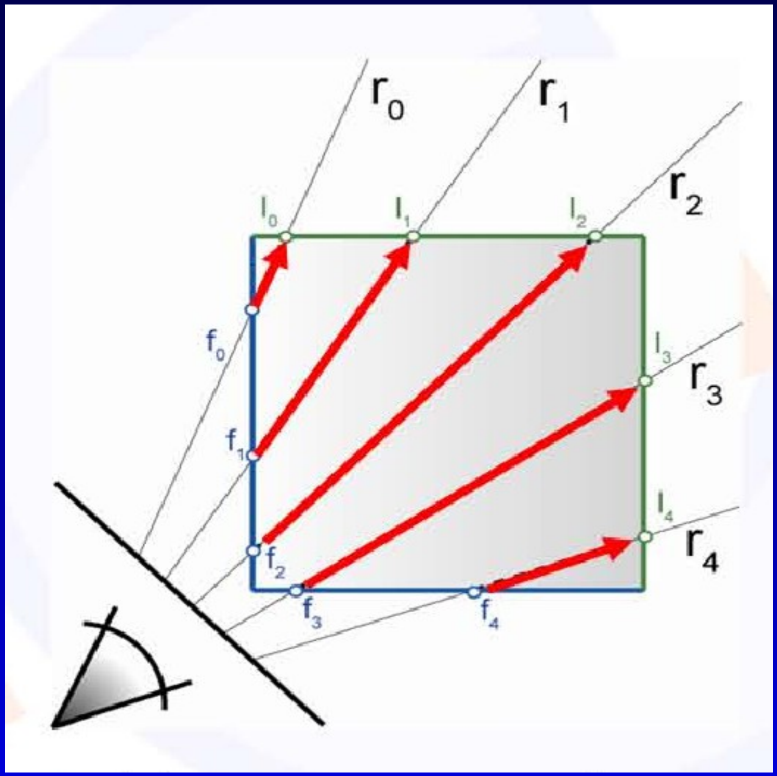
Surface enhancement and shading

```
output_data main(input_data IN,
            uniform float3 light,
            uniform sampler3D volume,
            uniform sampler3D gradient
{
output_data OUT;

float4 color1 = tex3D(volume,IN.texcoord1);
float3 normal = tex3D(gradient,IN.texcoord1);

float3 vecToLight = normalize(light - IN.position);
float difuseLight = max(dot(
            normalize(normal),vecToLight),0);

OUT.color.xyz =  float3(1.0,1.0,0.0)*difuseLight;
OUT.color.a = 10*length(normal)*color1.a;

return OUT;
}
```



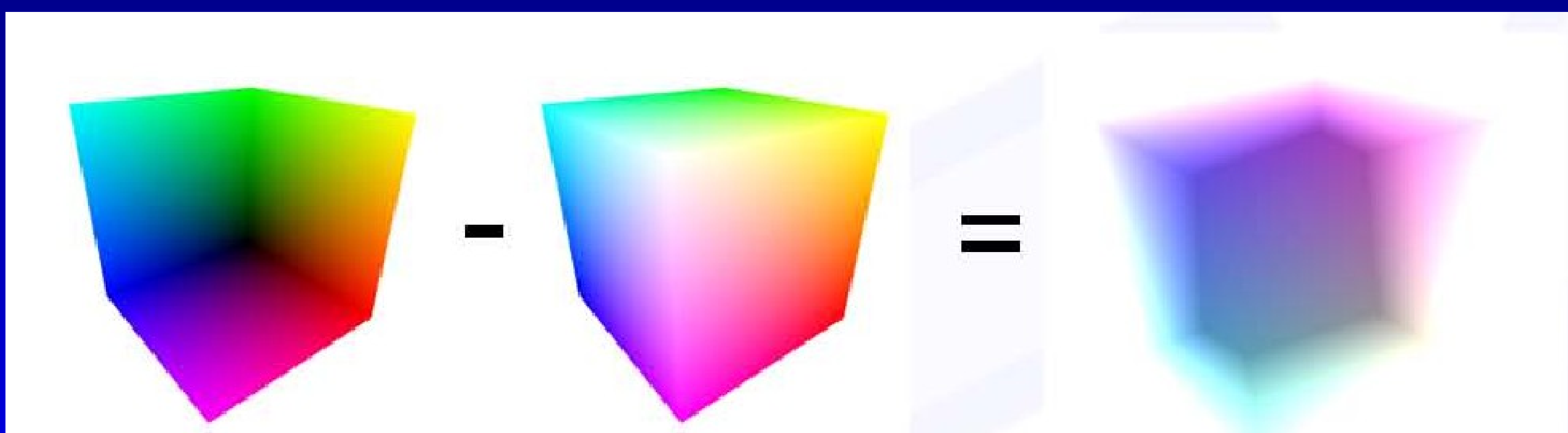
GPU-based Volume Ray Casting

- On principle the same as CPU ray casting
- Special conditions of the environment
  - Store volume as 3D-texture, cast rays in fragment program, ...



Basic Ray Setup

- Start & end point and direction rquired
- Evaluated in shader by rasterization of the volume bounding box



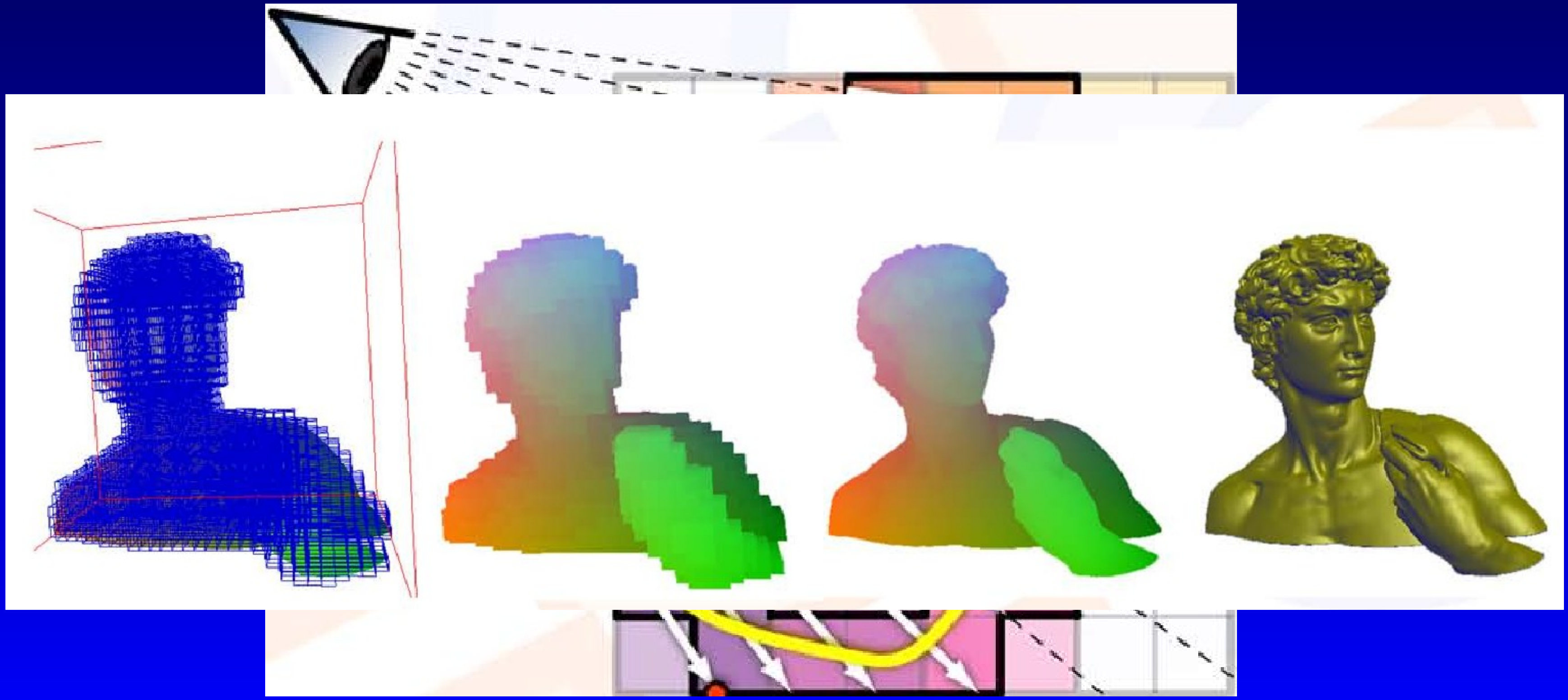
Back face   Front face   Ray directions

Standard Optimizations Possible

- Early ray termination:
  - Isosurface: stop on a surface
  - DVR: stop when accumulated opacity > threshold
- Empty space skipping:
  - skip transparent samples
  - Traverse hierarchy (e.g.: octree)

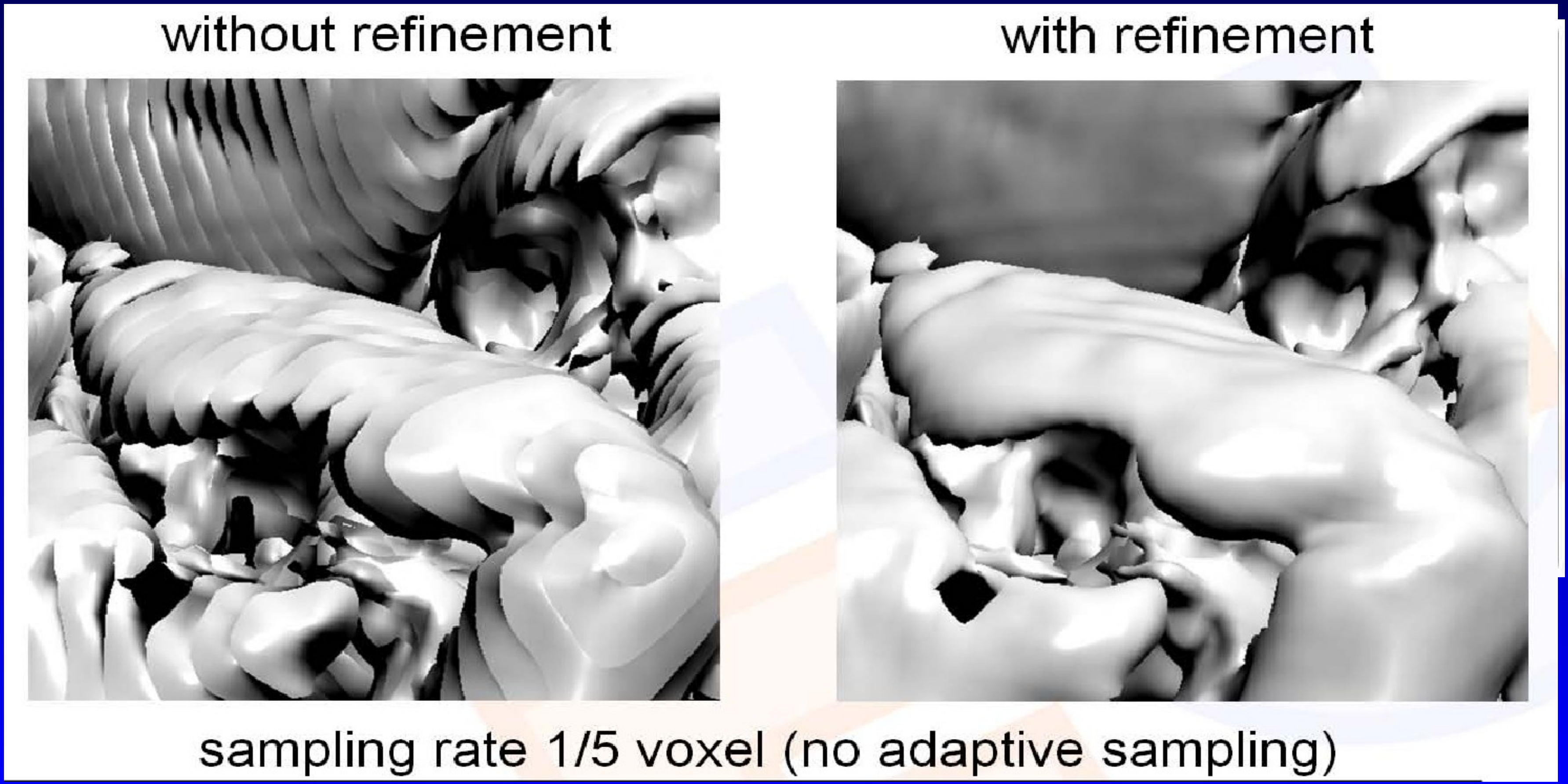
Empty Space Skipping

- **Bricking: use approximation instead of the bounding volume**



Intersection Refinement

- Bisection: fixed step number or



Advanced Techniques

- Light interaction
- Illumination models
  - Reflection
  - Shadows
  - Semi-transparent shadows
- Ambient occlusion (local, dynamic)
- Scattering (single and multiple, Monte-Carlo,...)

c) Close-up of vessels in (a)

d) Close-up of vessels in (b)

GPU for General Computations (gpgpu)

- Modern GPUs: Single and double precision computational units available
- Accessible through special API
  - CUDA (NVIDIA)
  - Brooks (ATI)
  - OpenCL (HW independent, support multiple CPUs)
- Often used in supercomputers (see top500.org)

Gpgpu example: Gaussian filtering

- Filtering of m^3 volume by n^3 filter
- Theoretical complexity: $3nm^3$
- GPU requires enough data to process