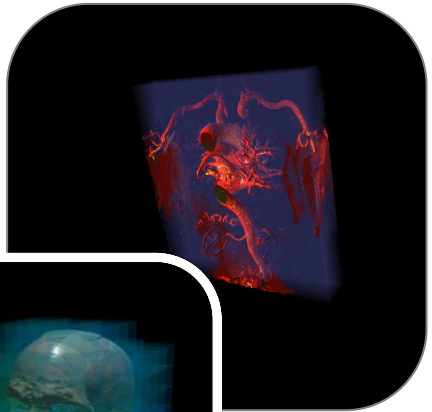
# Interactive visualization of medical datasets

**José A. Iglesias Guitián**
[ jalley@crs4.it ]

**http://3dah.miralab.unige.ch**

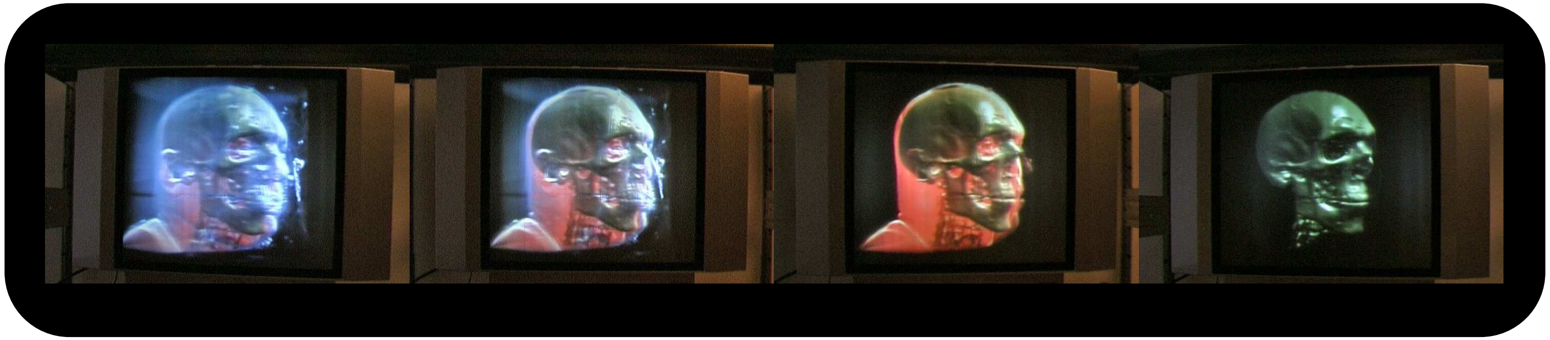
# Outline

## (I) Introduction to medical volume visualization

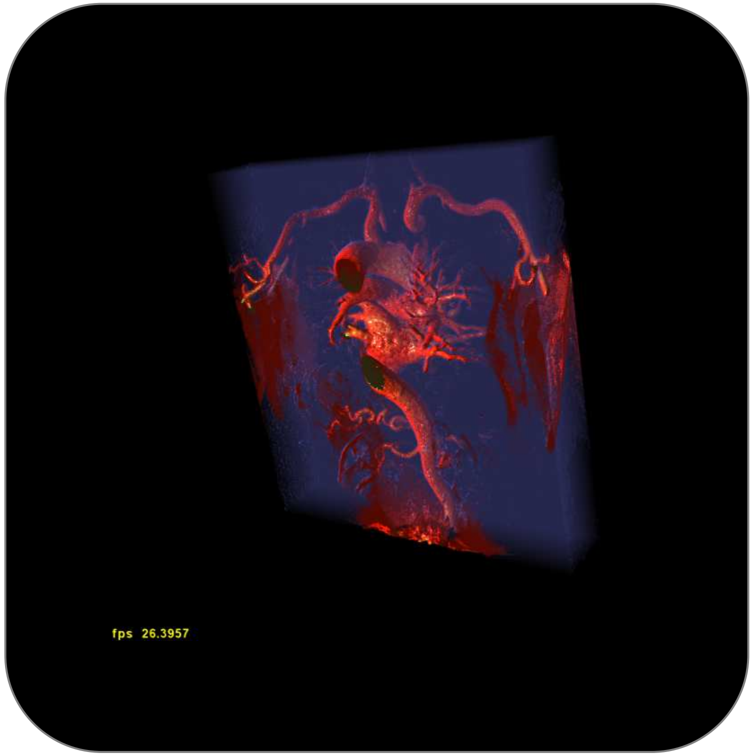


## (I) Rendering of massive volumetric datasets

## (II) Enhanced Direct Volume Rendering using a Light Field Display



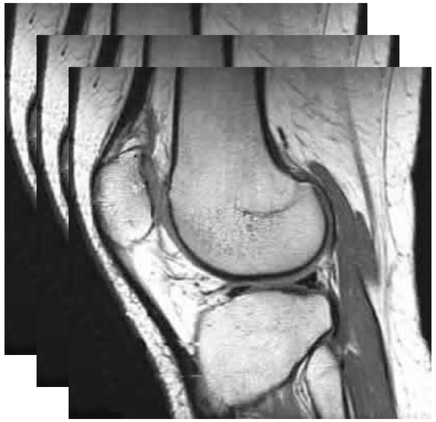
# Introduction to medical volume visualization



# Reconstruction of volumetric datasets

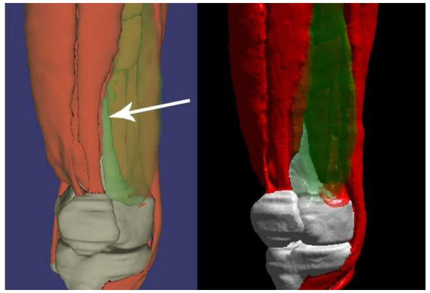
- Application in medicine, geology, archaeology, material science, biology, computational science and engineering, etc.

- Particularly, in medicine, hospitals acquire collections of 2D images



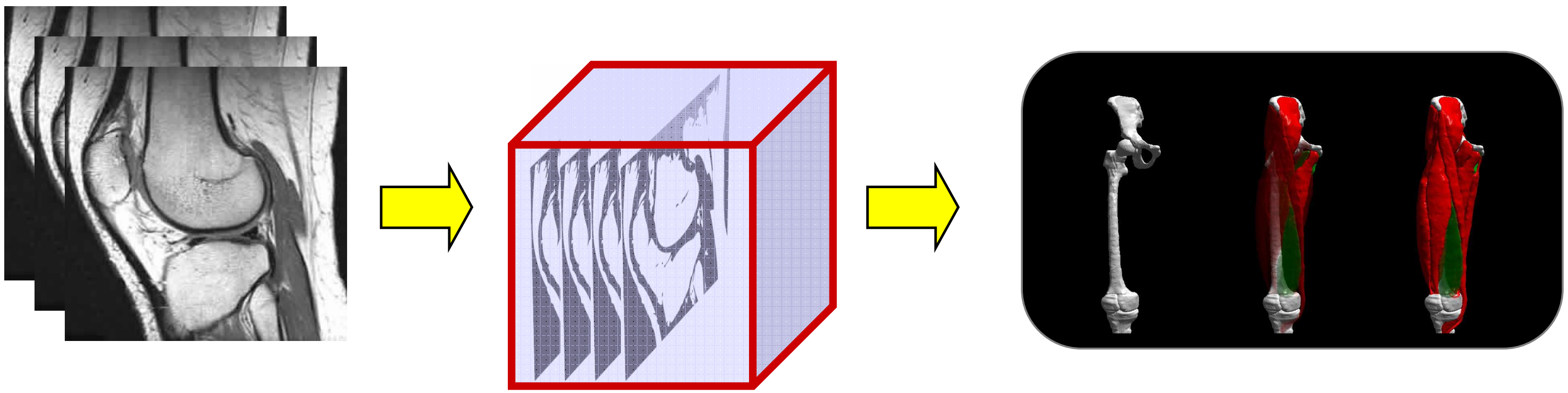
Source: http://www.physics.utoronto.ca

- Volume rendering is the main accepted approach for volume reconstruction



# Volume rendering for 3D reconstruction

Dataset 3D Rendering Interaction+classification



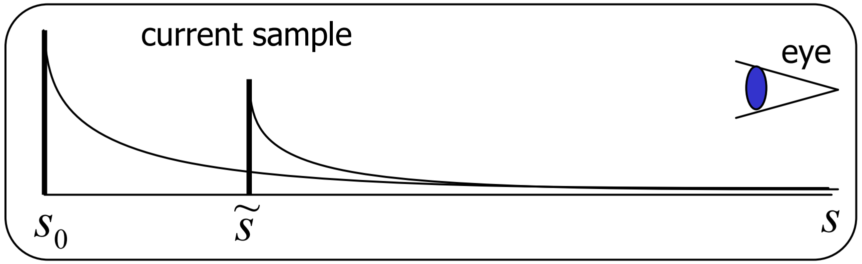
- OBJECTIVE : Real time interaction and rendering on commodity graphics hardware. We would like to support segmented data as input

# Direct Volume Rendering (DVR)

- Map sample values to an opacity and color.
- This mapping is done using a **transfer function**
- The resulting RGBA value is projected onto the correspondent pixel of the frame buffer.
- Projection techniques:
  - Splatting
  - Shear Warp
  - Texture Mapping
  - Ray-casting
  - **GPU Ray-casting**

# Ray-casting integration

- Emission absorption model




$$I(s) = I(s_0)e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(\tilde{s})e^{-\tau(\tilde{s},s)}d\tilde{s}$$

- Numerical solutions

Back-to-front iteration

$$C_i' = C_i + (1 - A_i)C_{i-1}'$$

Front-to-back iteration

$$C_i' = C_{i+1} + (1 - A_{i+1})C_i'$$

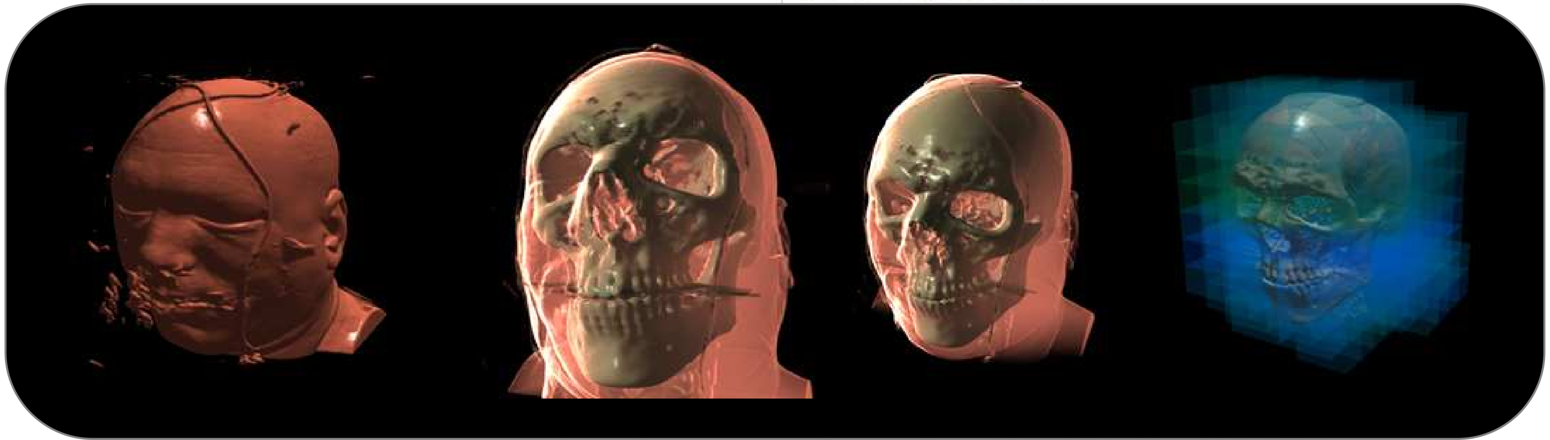
$$A_i' = A_{i+1}' + (1 - A_{i+1}')A_i$$

# Optimization techniques for DVR

- Empty Space Skipping
  - Avoid rendering transparent regions
- Early Ray Termination
  - When the volume is rendered in front to back order, once sufficient dense material has been encountered for a pixel, further samples will make no significant contribution so may be ignored
- Volume Segmentation
- Octree and BSP space subdivision
  - Use of hierarchical structures for both compression and speed-up
- Multiple and Adaptive Resolution Representation
- Pre-integrated volume rendering
  - In order to reduce sampling artifacts by pre-computing much of the required data

# (I) Rendering of massive volumetric datasets

```
fragment.color=float4(0,0,0,0);
fragment.depth=FAR;
```



```
_max, exit_face_idx, exit_dir) =
_clip(ray, t_min, t_max, box_min, box_dim);
non-empty block, access data and accumulate
s_empty(node.w)) {
_ptr=tex3d(spatial_index, node.xyz);
gment.color, fragment.depth) =
cumulate(fragment.color,
         ray, t_min, box_t_max,
         data_ptr, box_min, box_max);

ray exits from current block, move to neighbor
or_offset=float3(1+exit_face_idx,0,0)*texel_sz;
or=tex3d(spatial_index, node.xyz+neighbor_offset);
tr=neighbor.xyz;
_level=neighbor.w;
m=exp2(-octree_level);
n=trunc(box_min/box_dim)*box_dim;
box_t_max;
```

Anatomical Human 3D

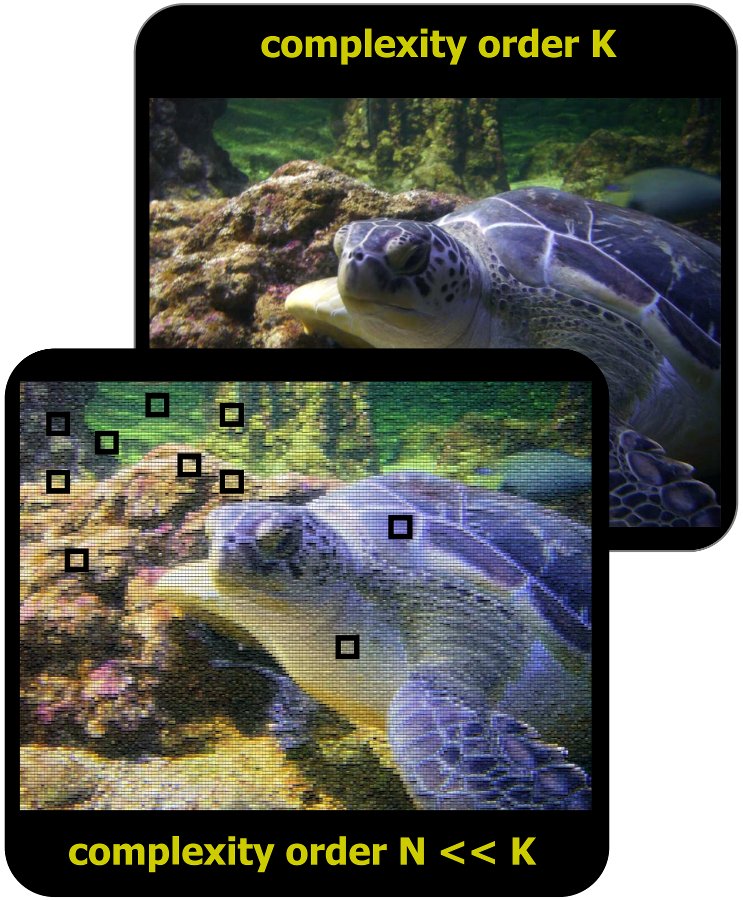
MARIE CURIE ACTIONS

# Goal and Motivation

## Accurate interactive inspection of very large volumes (unlimited size!) on PC platforms.

# Goal : unlimited size!

- Models of unbounded complexity on limited computers
  - We assume less data on screen (N) than in model (K→∞)
  - Need for output-sensitive techniques O(N), not O(K)
- Allow interactive exploration of multigiga-voxel datasets on a desktop PC

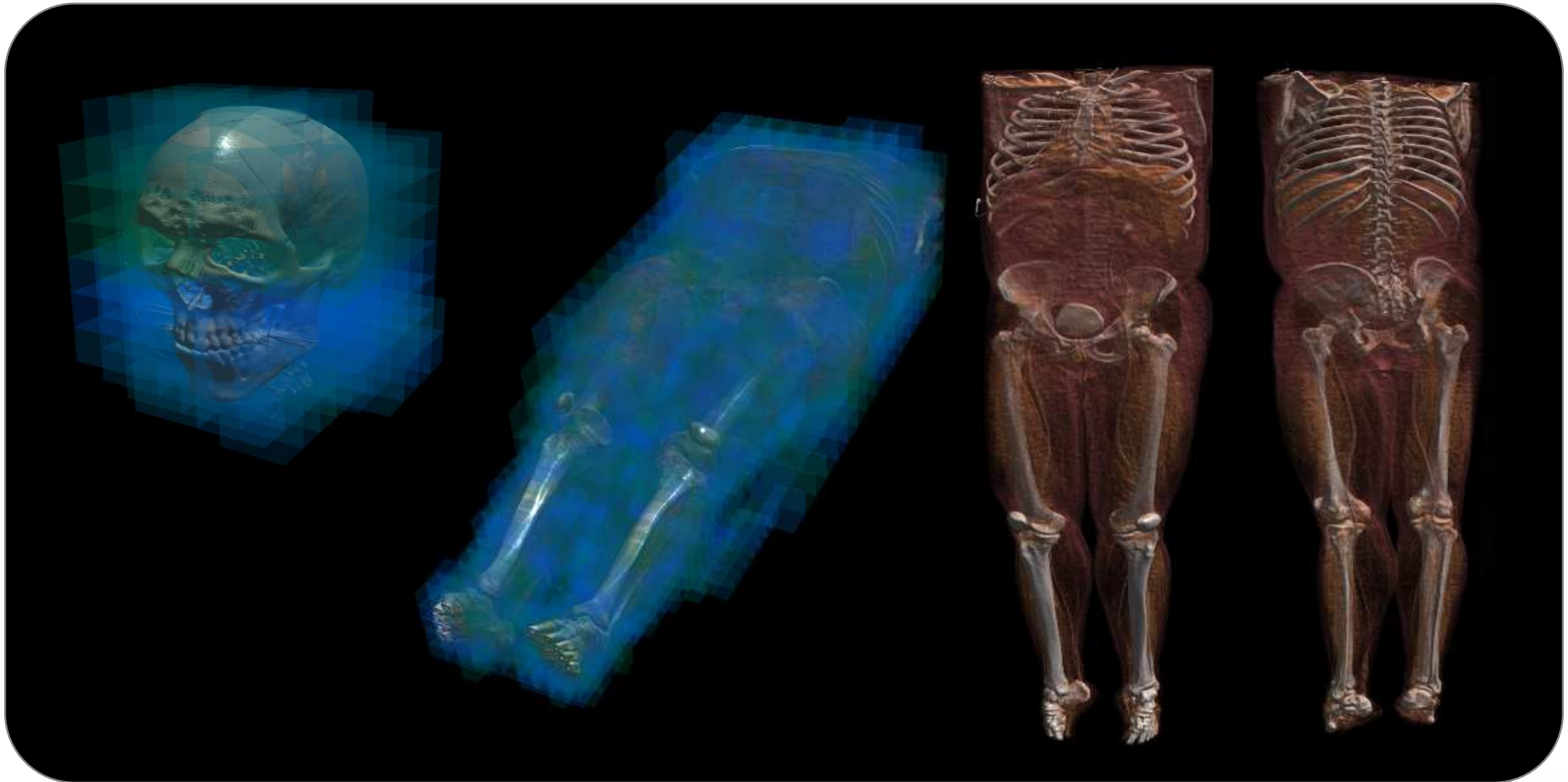


# Motivation

- Nowadays huge digital models are becoming increasingly available for a number of different applications ranging from CAD, industrial design to medicine and natural sciences.
- In the field of medicine, data acquisition devices such as MRI or CT scanners routinely produce huge volumetric datasets.
- Ray-casters fully executed by GPU fragment programs, have demonstrated the ability to deliver real-time frame rates for moderate-size data visualization.

# Our main contribution

- We propose a method based on the decomposition of a volumetric dataset into small cubical bricks, which are then organized into an octree structure maintained out-of-core.



# Related work (1/2) - CPU based methods

- Separate rendering of blocks and frame buffer composition
  - Multiresolution sampling of octree tile blocks according to **view-dependent** criteria

  [LaMar et al. 1999]

  - Coarse octree built upon uniform sub-blocks of the volume, and use **data dependent** measures to select block resolution

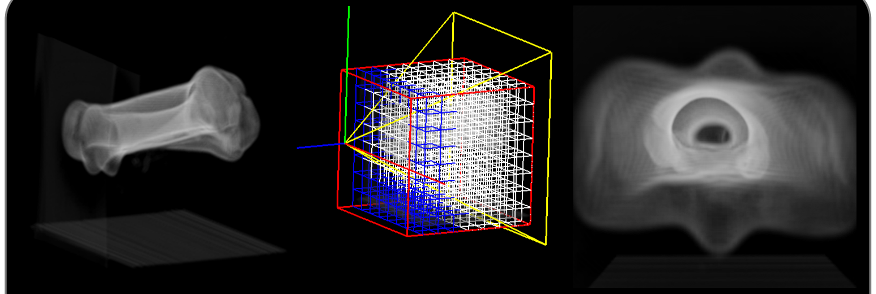
  [Boada et al. 2001]

  - Decomposition into **wavelet compressed blocks**, use block resolution to determine inter-slice distance, introduction of methods for **empty space skipping** and **early ray termination**

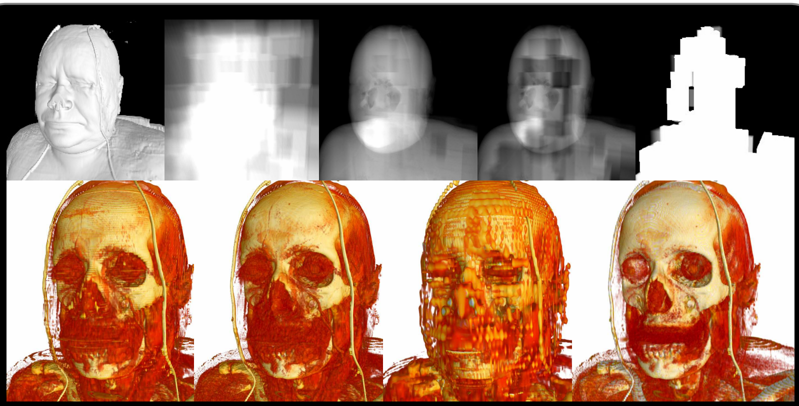
  [Guthe et al. 2004]

- Slice-based volume rendering
  - Accelerated by skipping empty blocks and exploiting an opacity map for **occlusion culling**

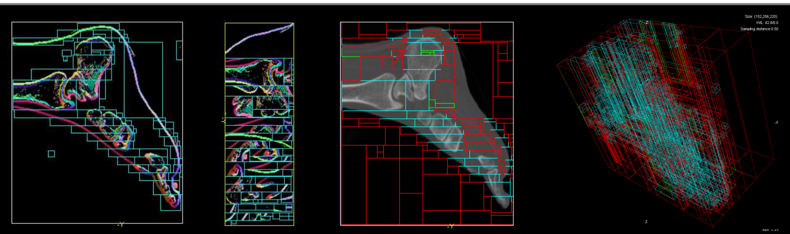
  [Li et al. 2003]



Source: LaMar, IEEE Visualization 1999



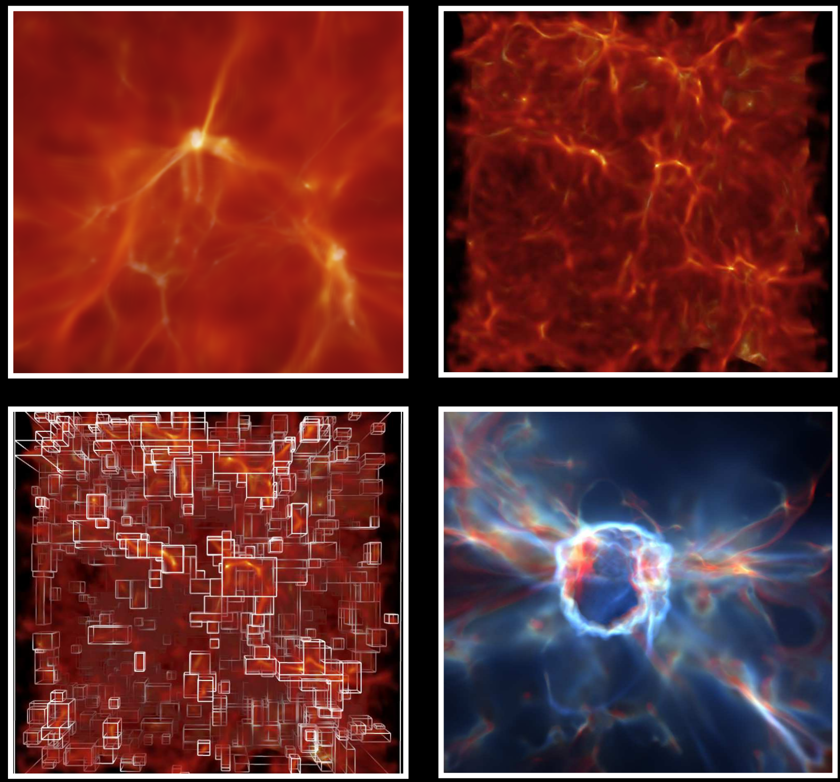
Source: Guthe, Computer & Graphics 2004



Source: Li, IEEE Visualization 2003

# Related work (2/2) - GPU based methods

- [GPU] Separately render blocks using volumetric raycasting on the GPU and sort cells into layers for front-to-back rendering
  - Devise propagation methods to sort cells into layers for front-to-back rendering
  [Hong et al 2005, Kaehler et al 2006]
    - Problems:
      - These methods create artifacts on the boundaries
      - Difficult to implement optical models with rays changing direction (refraction, global illumination, etc.)
- How to fit large volume datasets into GPU memory?
  - Compressing data using:
    - **adaptive texturing schemes** to fit data in a compressed form [Vollrath et al. 2006]
      - Problem: sampling density
    - using **flat multiresolution blocking** methods [Ljung et al. 2006]
      - Problem: number of blocks is constant and the method remains performing only if individual blocks are within a small range of sizes



Source: Kaehler, Eurographics / IEEE VGTC Workshop on Volume Graphics, 2006

# Our contribution

## A GPU-friendly output sensitive technique

- We face a real-time data filtering problem!
- Our proposed solution combine:
  - A multiresolution and spatial subdivision structure
    - Spatial indexing
    - Visual approximation
  - A view-dependent renderer
    - Spatial Index Texture & stackless GPU raycaster
    - Visibility & Occlusion culling
  - An efficient memory management subsystem

# Multiresolution Out-of-core Volume Rendering

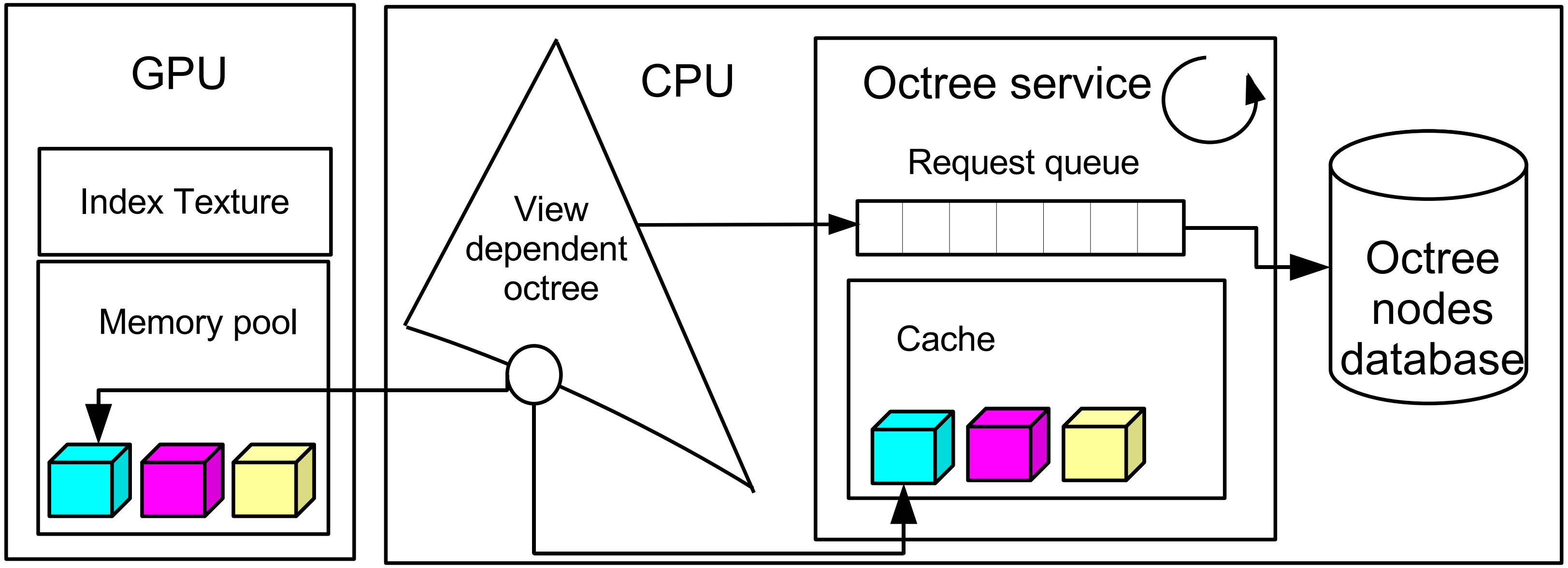
## Multiresolution and spatial subdivison structure

- Preprocessing overview:
  - Use an **octree structure** to save the volumetric model
  - Decompose the original volume into **small cubical bricks**
  - **Empty space skipping** (skipping empty bricks)
  - For each non-empty brick save:
    - Voxel values
    - Range of values (min-max)
    - Optional precomputed gradients
  - **Visual approximation** : reconstruct inner nodes by bottom-up recombination using:
    - Median filtering for values
    - Sobel 5x5x5 3d filtering for gradients

# Multiresolution Out-of-core Volume Rendering

## View dependent renderer

- Real-time rendering overview:



# Multiresolution Out-of-core Volume Rendering

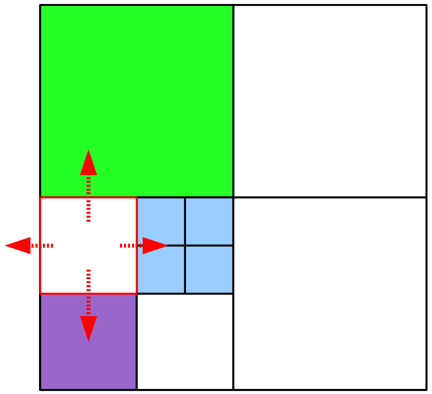
## View dependent renderer

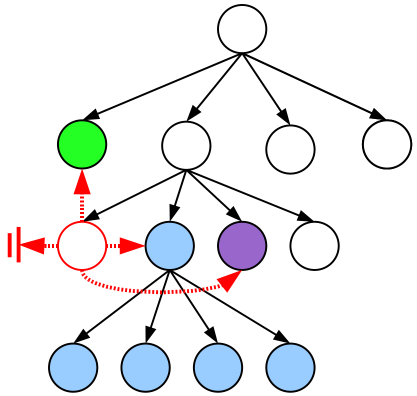
- Real-time rendering overview:
  - Use a CPU runtime loader that updates a view and transfer function – dependent **working set of bricks**
  - **Asynchronously** mantain bricks on both CPU and GPU memory fetching data from the out-of-core octree
  - **Adaptive refinement** method guided by priority:
    - Sorted by decreasing projected screen-space size of voxels
    - Sorted by the decreasing number of pixels visible resulting from the feedback of the occlusion queries
  - **Spatial Index Texture**
  - **Stackless GPU raycaster**
  - **Visibility & Occlusion culling**

# Multiresolution Out-of-core Volume Rendering

## View dependent renderer - Spatial Index Texture

- Spatial Index Texture:
  - Structure created **on-the-fly** at each frame which encode the minimum amount of data required for octree traversal.
  - Use an 8 bit RGBA texture encoding a tag in the alpha value:
    - Set A = 0.0 if RGB is a pointer to an **empty node**
    - Set A = 0.5 if RGB is a pointer to an **inner node**
    - Set A = 1.0 if RGB is a pointer to **data**
  - **Octree ropes** structure for stackless traversal [Havran et al, 1998]



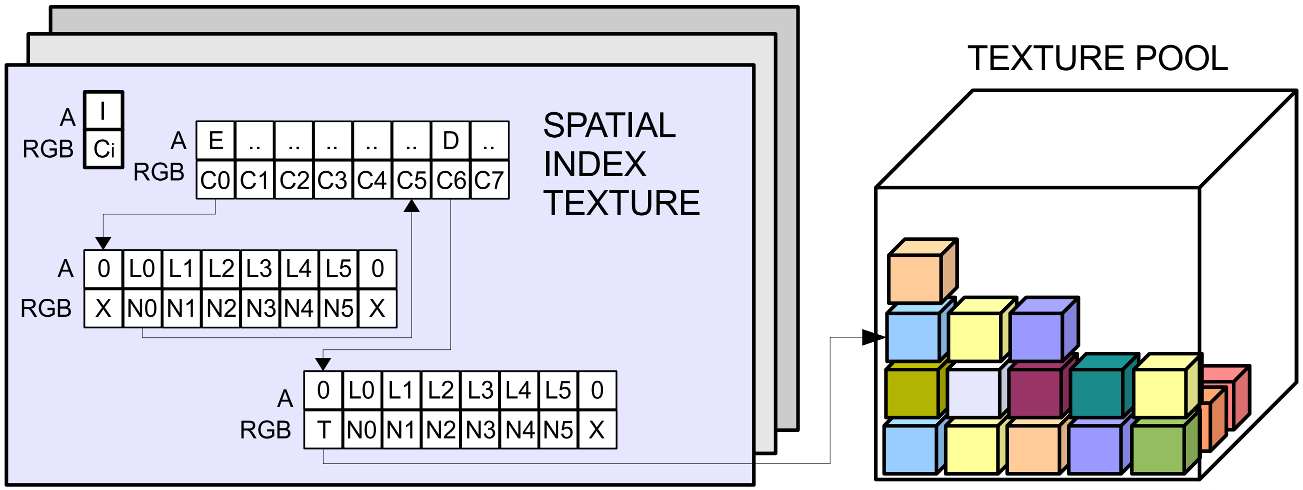


# Multiresolution Out-of-core Volume Rendering

## View dependent renderer – Stackless GPU raycaster

- Stackless GPU raycaster:
  - Streamlined octree extension of an efficient stackless ray traversal method for kd-trees [Popov et al, 2007]
  - Computes the volume rendering integral using non-empty bricks in **front-to-back** order and **early ray termination**.
  - **Adapt sampling density** to brick resolution.

I : inner node tag
E : empty leaf tag
D : non-empty leaf tag
C0..C7: children pointers
N0..N5: neighbor pointers
L0..L5 : neighbor levels
T : data pointer
X : NULL pointer



# Multiresolution Out-of-core Volume Rendering

## View dependent renderer – Fragment shader (octree traversal)

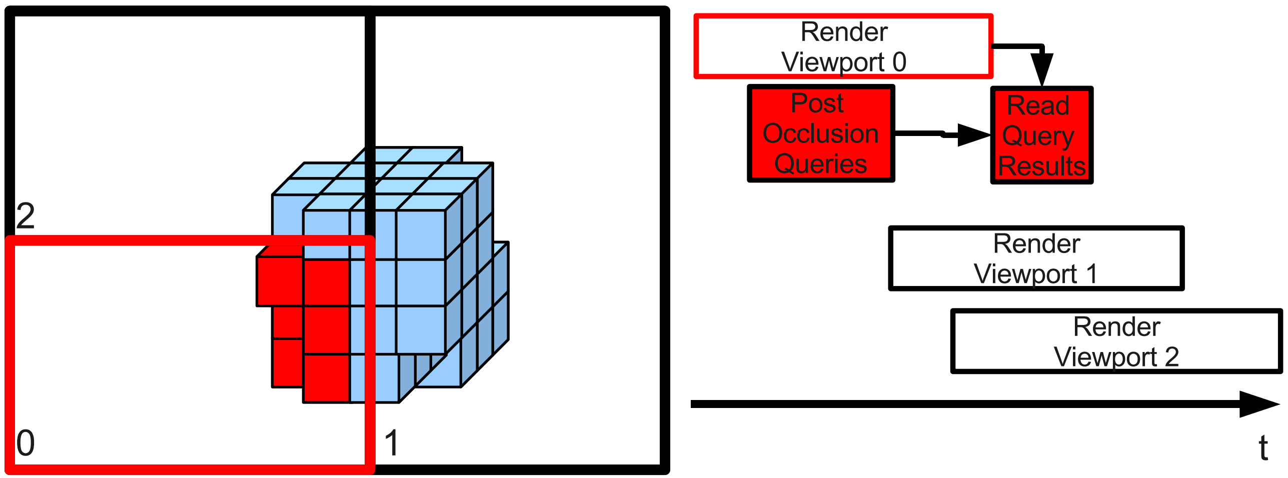
- Stackless algorithm
  - Compute neighbour information and bounding boxes on the fly
- Simple state for a ray
  - current node + entry point into the brick
- Reduce texture memory accesses
  - exploiting the regular structure of an octree
- Front to back rendering
- Adaptive sampling

```
fragment.color=float4(0,0,0,0);
fragment.depth=FAR;
// Start at octree root
node_ptr = float3(0,0,0); octree_level=0;
box_min=float3(0,0,0); box_dim=float3(1,1,1);
while (!is_null(node_ptr) and color.a<1) {
  // Find leaf containing current sampling point
  P = ray.start+ray.dir*t_min;
  node = tex3d(spatial_index, node_ptr);
  while (is_inner(node.w)) {
    box_dim/=2; box_mid=box_min+box_dim;
    s=step(P,box_mid); box_min+=s*box_dim;
    child_offset=dot3(s,float3(1,2,4))*texel_sz;
    node_ptr=node.xyz+float4(child_offset,0,0,0);
    node=tex3d(spatial_index, node_ptr);
    ++octree_level;
  }
  // Clip ray to box and find exit face
  (box_t_max, exit_face_idx, exit_dir) =
    box_clip(ray, t_min, t_max, box_min, box_dim);
  // If non-empty block, access data and accumulate
  if (!is_empty(node.w)) {
    data_ptr=tex3d(spatial_index, node.xyz);
    (fragment.color, fragment.depth) =
      accumulate(fragment.color,
                 ray, t_min, box_t_max,
                 data_ptr, box_min, box_max);
  }
  // If ray exits from current block, move to neighbor
  neighbor_offset=float3(1+exit_face_idx,0,0)*texel_sz;
  neighbor=tex3d(spatial_index, node.xyz+neighbor_offset);
  node_ptr=neighbor.xyz;
  octree_level=neighbor.w;
  box_dim=exp2(-octree_level);
  box_min=trunc(box_min/box_dim)*box_dim;
  t_min=box_t_max;
}
```

# Multiresolution Out-of-core Volume Rendering

## View dependent renderer – Visibility & Occlusion culling

- Visibility & Occlusion culling:
  - Occlusion culling only with early ray termination is not optimal
  - We propose a **feedback mechanism** by marking visible bricks in the previous frame and using occlusion queries
  - Use **screen space subdivision** in order to avoid wasting time waiting for the occlusion queries response



# Multiresolution Out-of-core Volume Rendering

## Overview

- Independent brick processing:
  - For each brick:
    - Filtering
    - Compressing (LZO)
- Out-of-core + Parallelizable

- Construction:
  - Decompose the original volumetric model into **small cubical slightly overlapped bricks**
  - **Skip empty bricks** and for those not empty save the range of values and optional precomputed gradients
  - Reconstruct inner nodes by bottom-up recombination using:
    - Median filtering for values
    - Sobel 5x5x5 3d filtering for gradients

- Out-of-core + GPU octree traversal / GPU optimized cache.
- View + Occlusion culling
- NPR + Isosurface rendering

- Rendering:
  - GPU octree traversal and view dependent octree reconstruction
  - GPU-friendly cache refilling to exploit GPU bandwidth
  - Occlusion culling using Z-buffer + OpenGL occlusion queries

# Multiresolution Out-of-core Volume Rendering

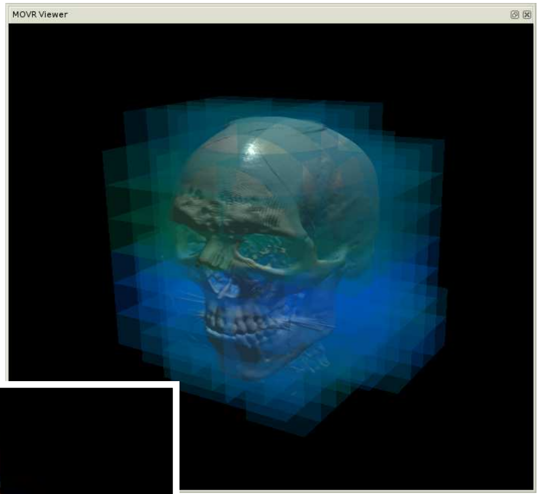
## Results

**Visible human head data set**

**Source**: The National Library of Medicine, USA

**Resolution**: 256x256x128

**Platform**: Linux PC
AMD Opteron QuadCore
4GB RAM memory
SATA2 disks
GeForce 8800 Ultra






MARIE CURIE ACTIONS

**Alias Name**: OBELIX
**Modality**: CT 16
**File Size: 636 MB**
**Description**: Whole body contrast CTA acquired on a 16 detector CT scanner. Normal study.

**Source**: http://pubimage.hcuge.ch

Scientific Name: **Zaedyus pichiy**,
Common Name: Pichi Armadillo

Specimen Description: Upper Body

Specimen Source: uncatalogued

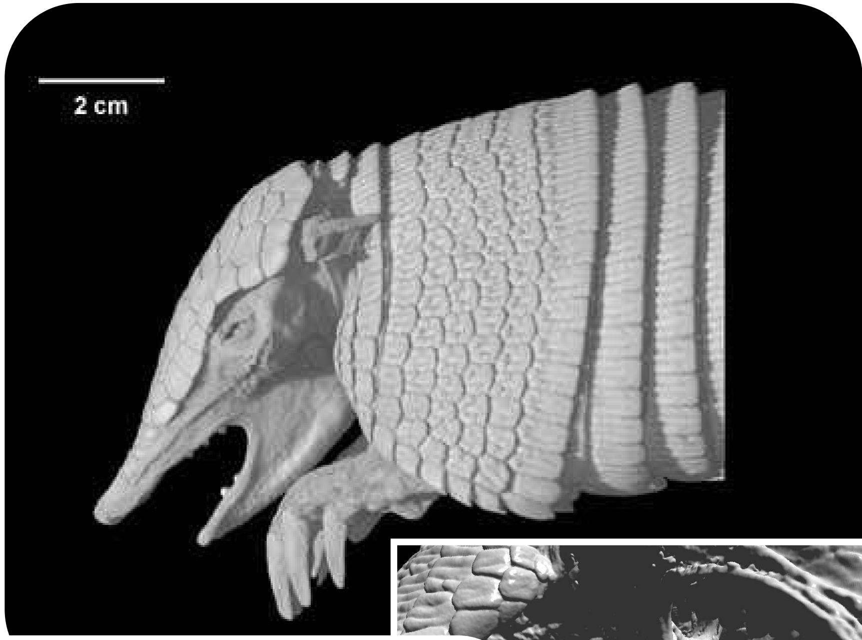
Scan Resolution: 1024x1024

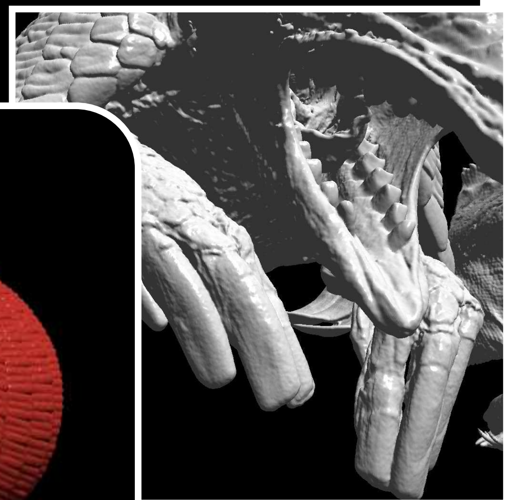
Number of Slices: 999

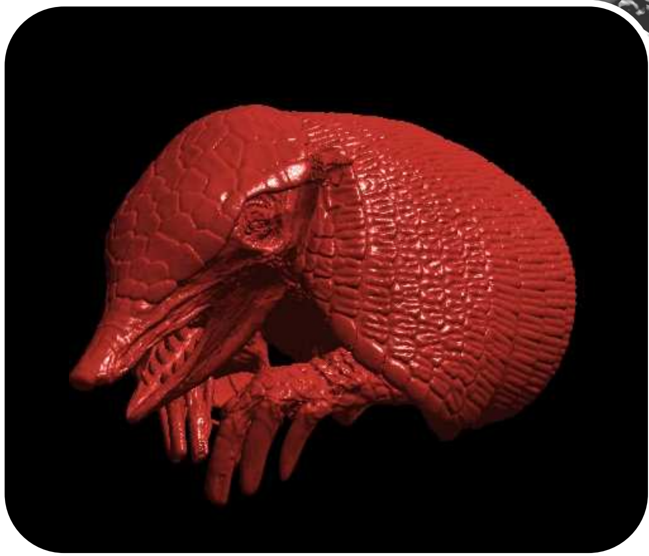
Slice Thickness: 0.1 mm

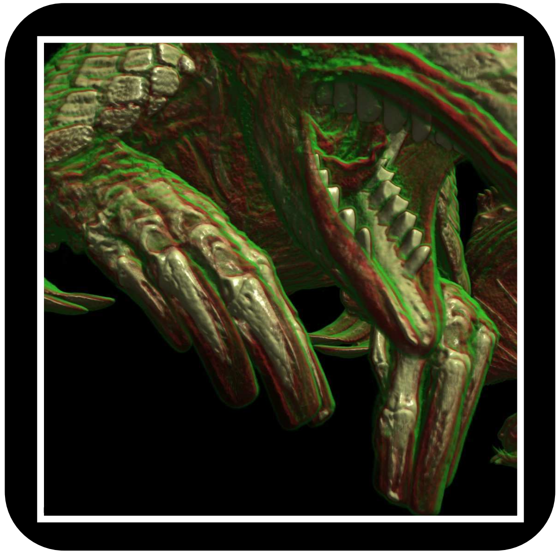
XY resolution: 0.0859375 mm (at full resolution)

Scan Date: 01-21-2004







Scientific Name: **Chamaeleo calyptratus**, Common Name: Veiled Chameleon

Specimen Description: Upper Body

Specimen Source: Texas Memorial Museum (TNHC 62768)

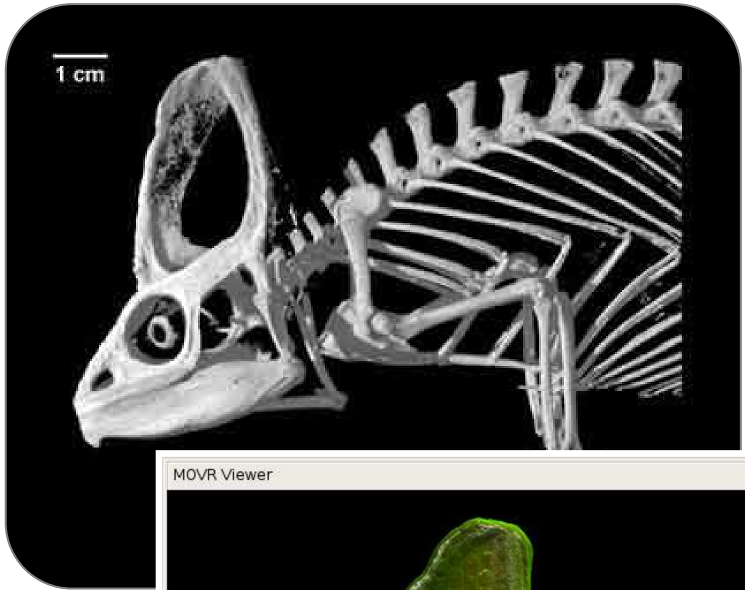
Scan Resolution: 1024x1024

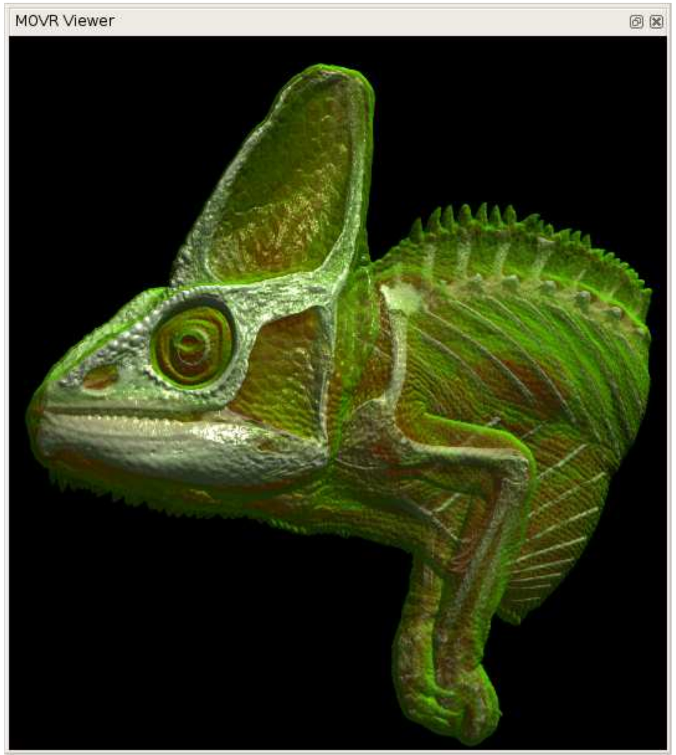
Number of Slices: 1080

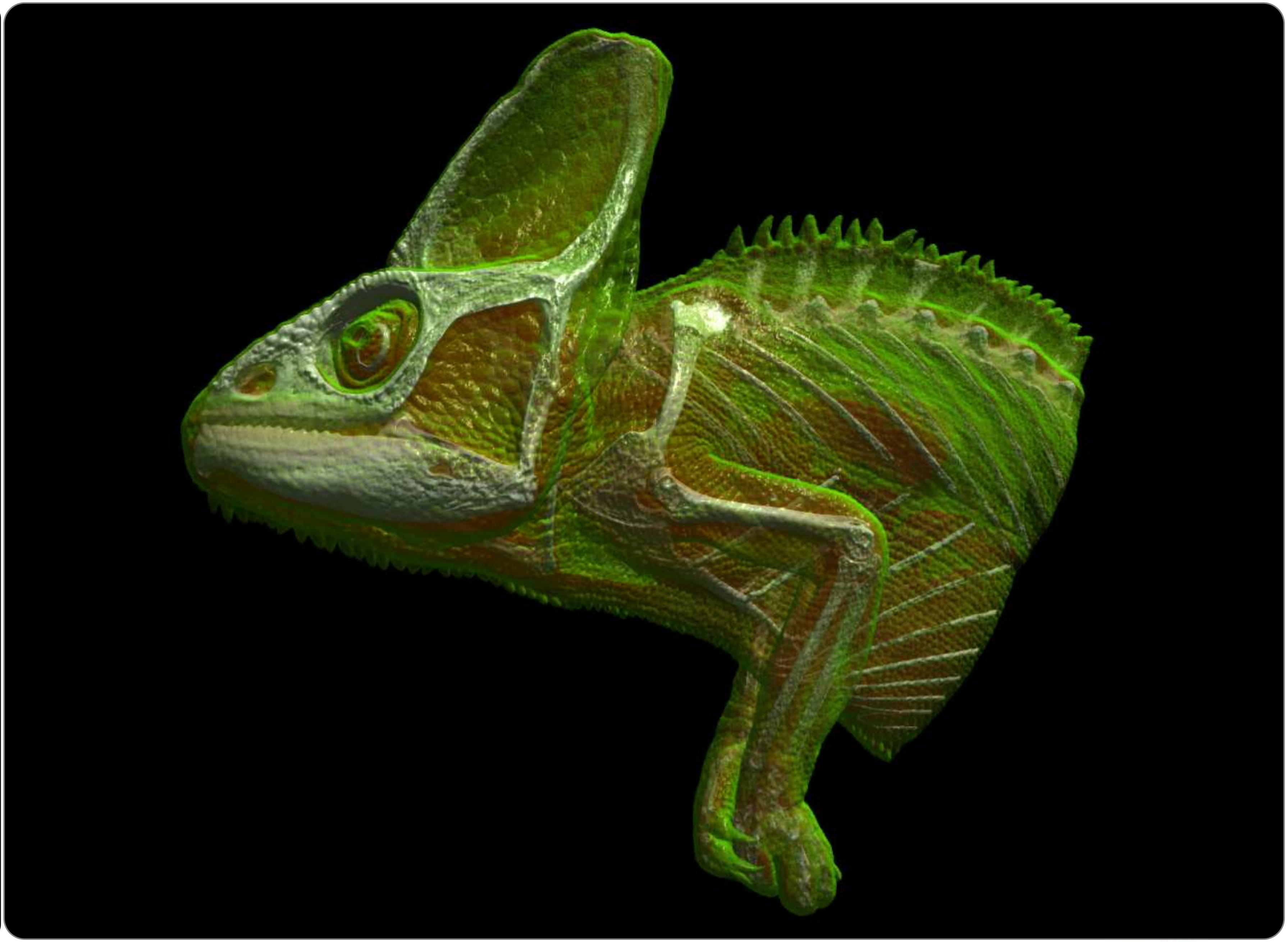
Slice Thickness: 0.105 mm

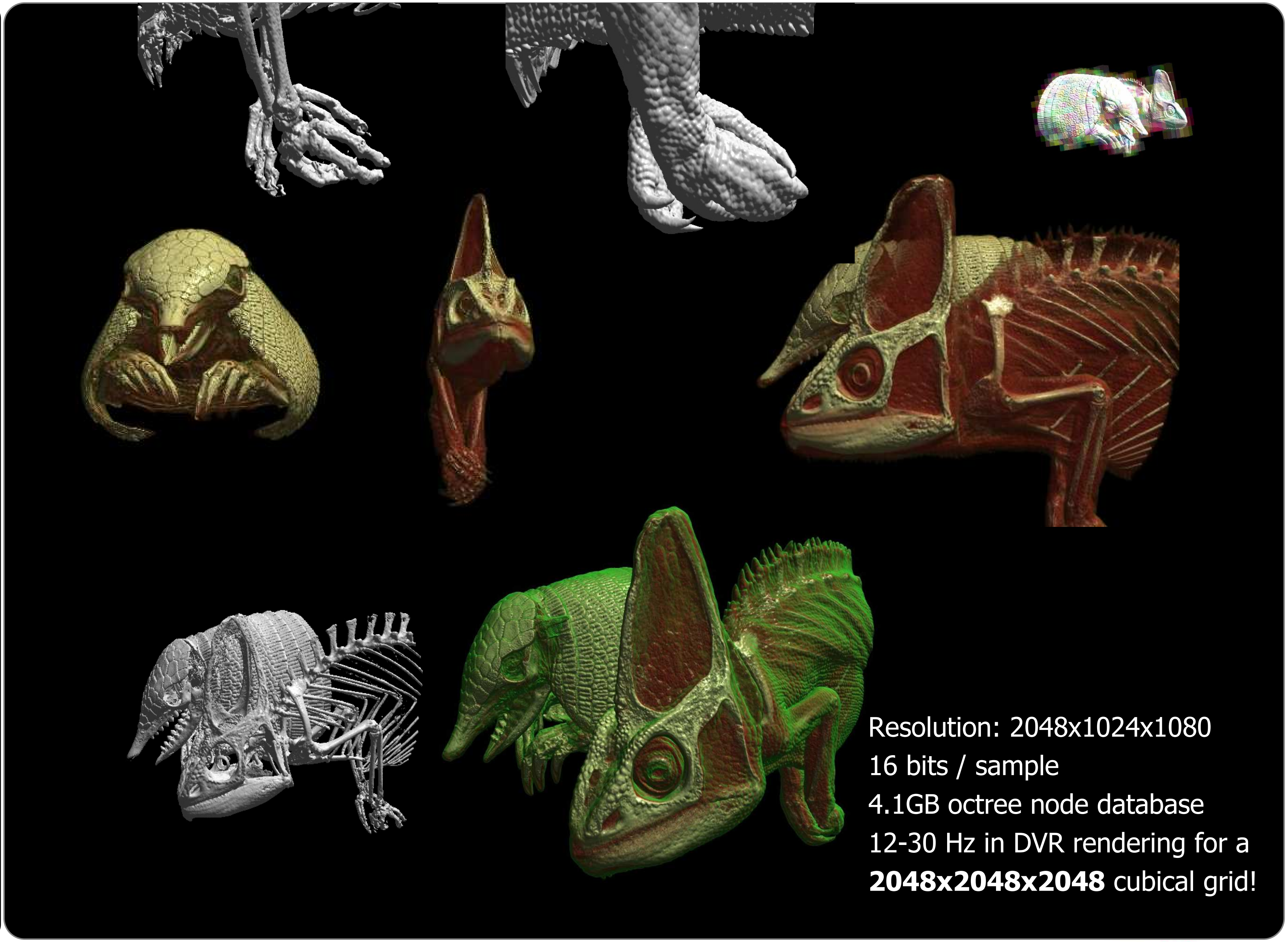
XY resolution: 0.09228515625 mm (at full resolution)

Scan Date: 07-11-2003






Resolution: 2048x1024x1080
16 bits / sample
4.1GB octree node database
12-30 Hz in DVR rendering for a
**2048x2048x2048** cubical grid!

# Multiresolution Out-of-core Volume Rendering

## Conclusions

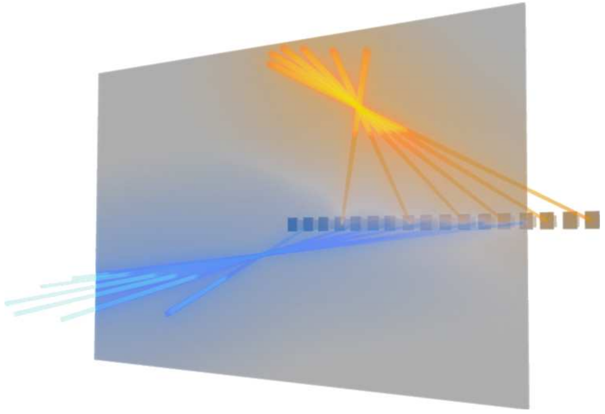
- We have proposed an adaptive out-of-core technique for rendering massive scalar datasets within a **single-pass GPU raycasting framework**
- We separate the working set mantainance on the CPU, from rendering, which is performed fully on GPU by a stackless raycaster
- Results demonstrate that the resulting method is able to interactive explore of multigiga-voxel datasets on a desktop PC

# Multiresolution Out-of-core Volume Rendering

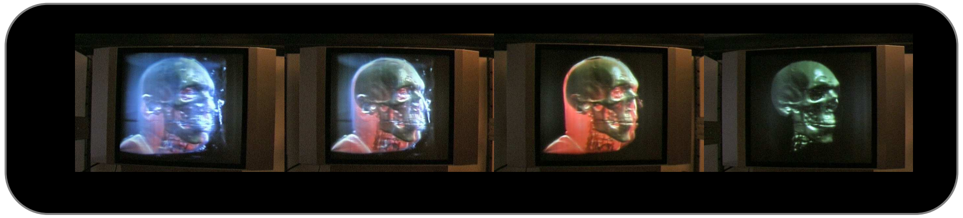
## Present and Future work

- Support new light-field displays prototypes
- Support RGB-based datasets and/or multidimensional transfer functions
- Parallelization on graphics-clusters: improve load balancing of the occlusion and visibility culling tasks

# (II) GPU Accelerated Direct Volume Rendering on an Interactive Light Field Display



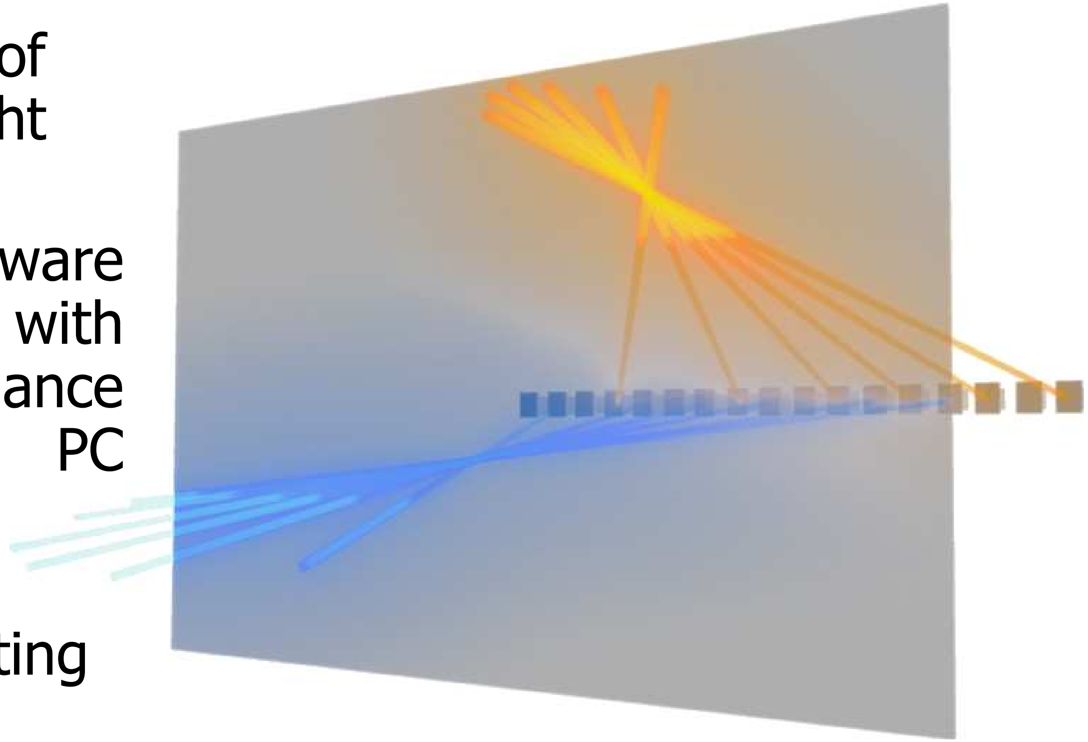
# Motivation



- Resolving the spatial arrangement of **complex 3D structures** in images produced by DVR techniques is a difficult task
- In particular, in medical data CT's and MRI's often contains **overlapping structures**, leading to cluttered images difficult to understand
- Two othogonal **research directions**:
  - Improving rendering quality with **advanced photo-realistic** and **non-photorealistic techniques**
  - Improving **volumetric understanding** by employing displays able to elicit more depth cues than the conventional 2D monitor or providing improved **color reproduction**

# Our main contributions

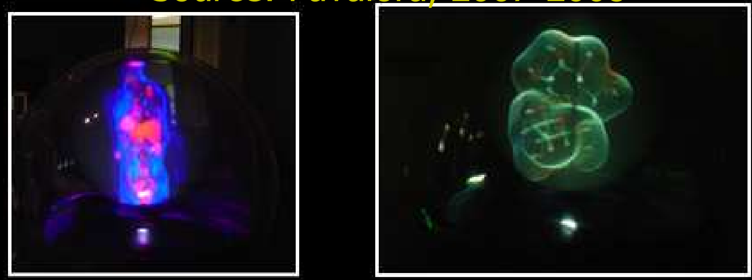
- A general MCOP technique for a class of horizontal parallax light field display
- A hardware and software prototype system with interactive performance on a single PC configuration
- GPU accelerated framework implementing volume ray-casting



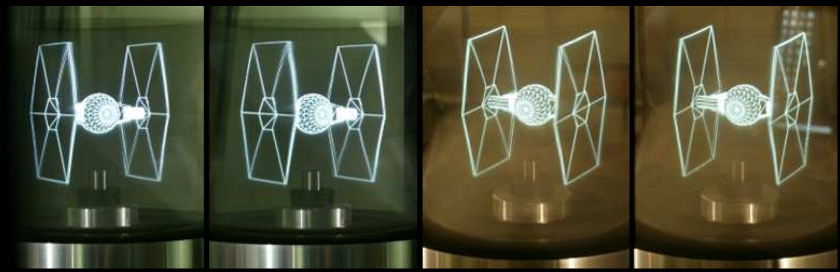
# Related work (1/3)

- **Interactive 3D displays**. The key feature characterizing 3D displays is direction-selective light emission
- Volumetric approaches
  - light beams projected on refractive/reflective media positioned or moved in space [McKay00, Favalora01, Jones07,Cossairt07]
- Pure holographic approaches
  - holographic patterns reconstructing the light wavefront originating from the displayed object, e.g., using optically addressed spatial light modulators [Stanley00], or digital micro-mirror devices [Huebschman03]
- Multi-view approaches
  - based on an optical mask or a lenticular lens array [Matusik04]
- Our display prototype employs multi-view technology combined with light shaping capabilities of a holographically recorded screen

Source: Favalora, 2007-2008



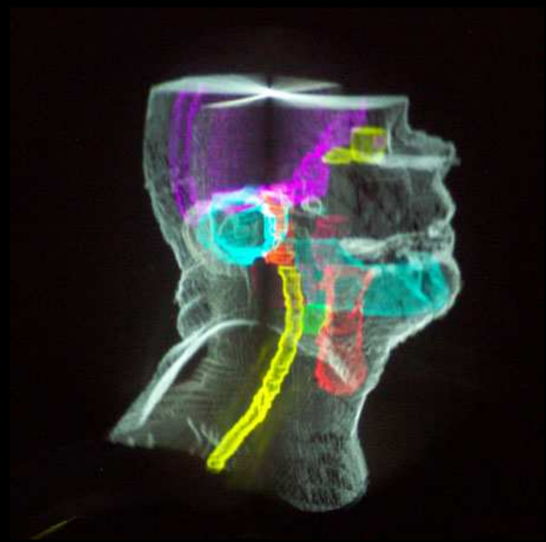
Source: Jones, Siggraph 2007



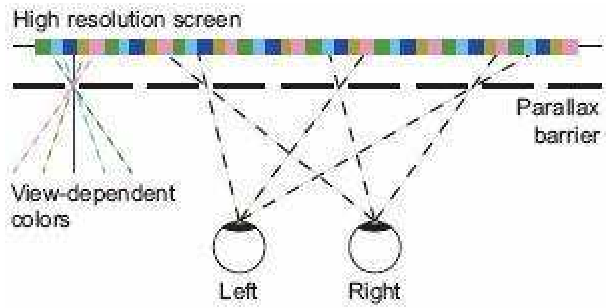
Source: Matusik, Siggraph 2004

# Related work (2/3)

- Projecting graphics to the 3D display
  - Multiple-center-of-projection techniques to produce images exhibiting correct stereo and motion parallax cues [Jones07,Halle98]
  - Standard orthographic or perspective projections simplify rendering but produce perspective distortions [Raskar98,Cossairt07]
  - Framework for studying sampling and aliasing for 3D displays [Zwicker06]



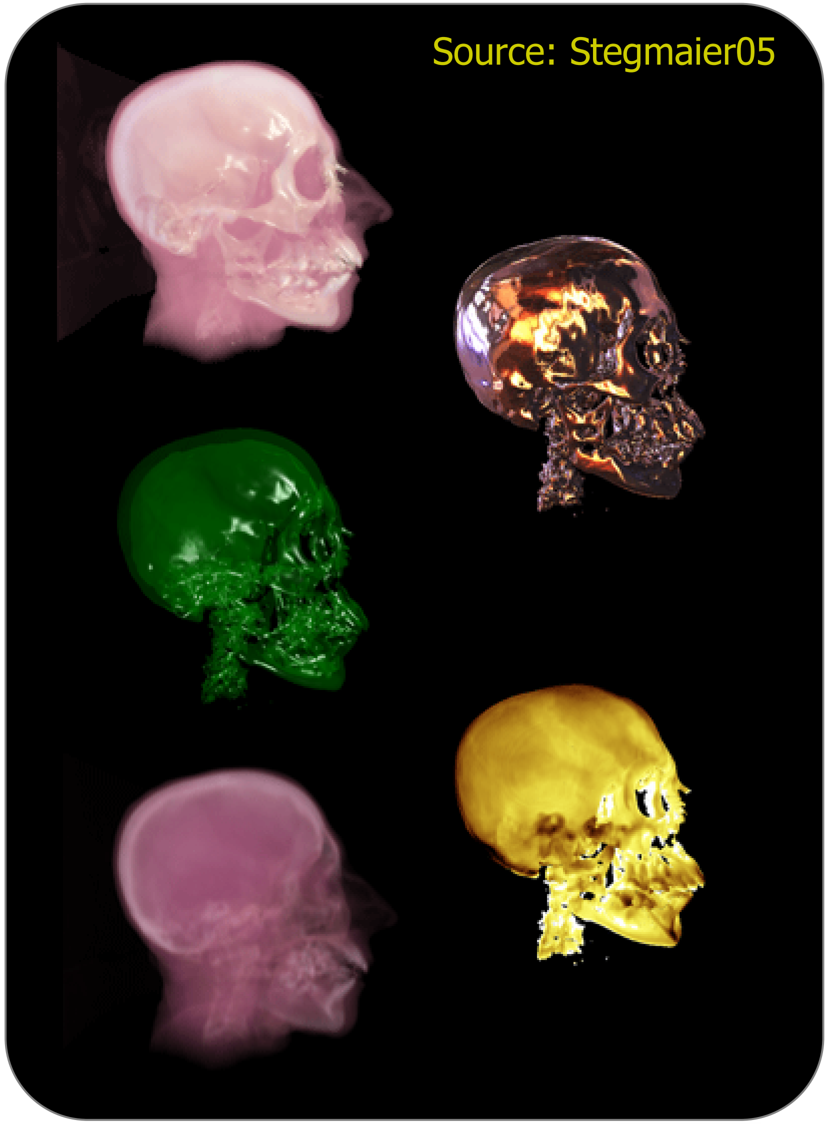
Source: Cossairt, 2007




Source: Zwicker06

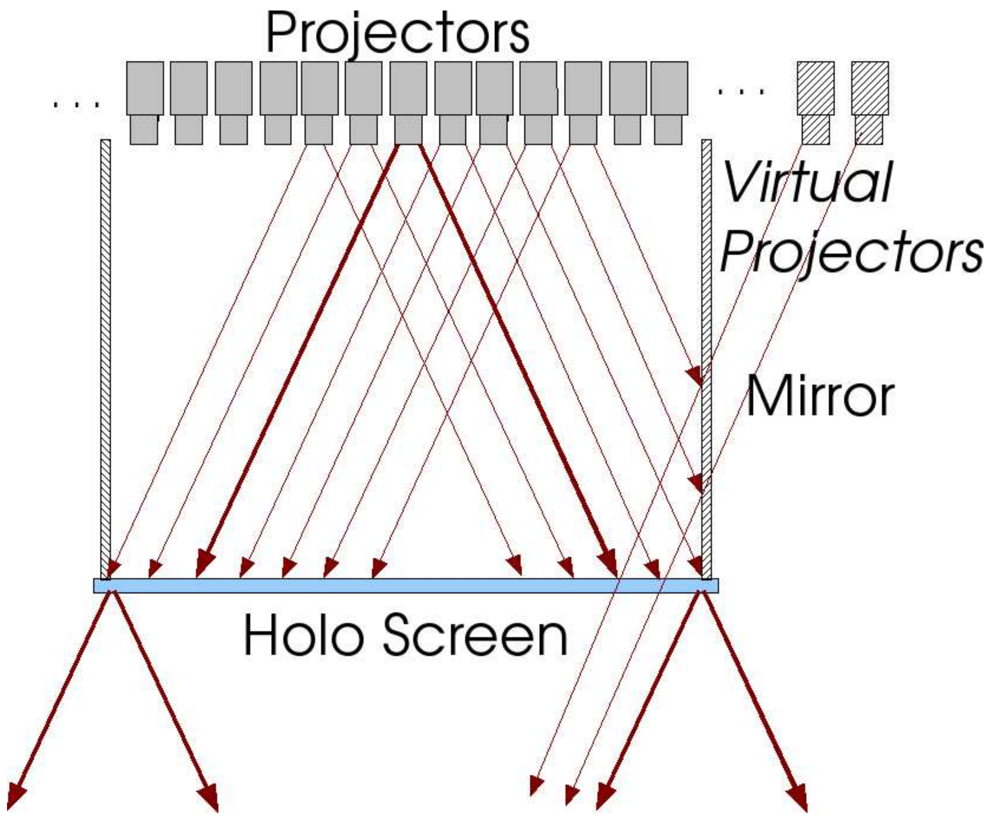
# Related work (3/3)

- GPU accelerated volume visualization on multi-view displays
  - survey of GPU accelerated volume rendering methods [Engel06]
  - single-pass GPU ray-casting [Stegmaier05]
  - acceleration methods for stereo volume rendering [Wan04]
- We exploit GPU vertex shaders to render proxy geometry that activates a fragment shader performing the actual ray-casting



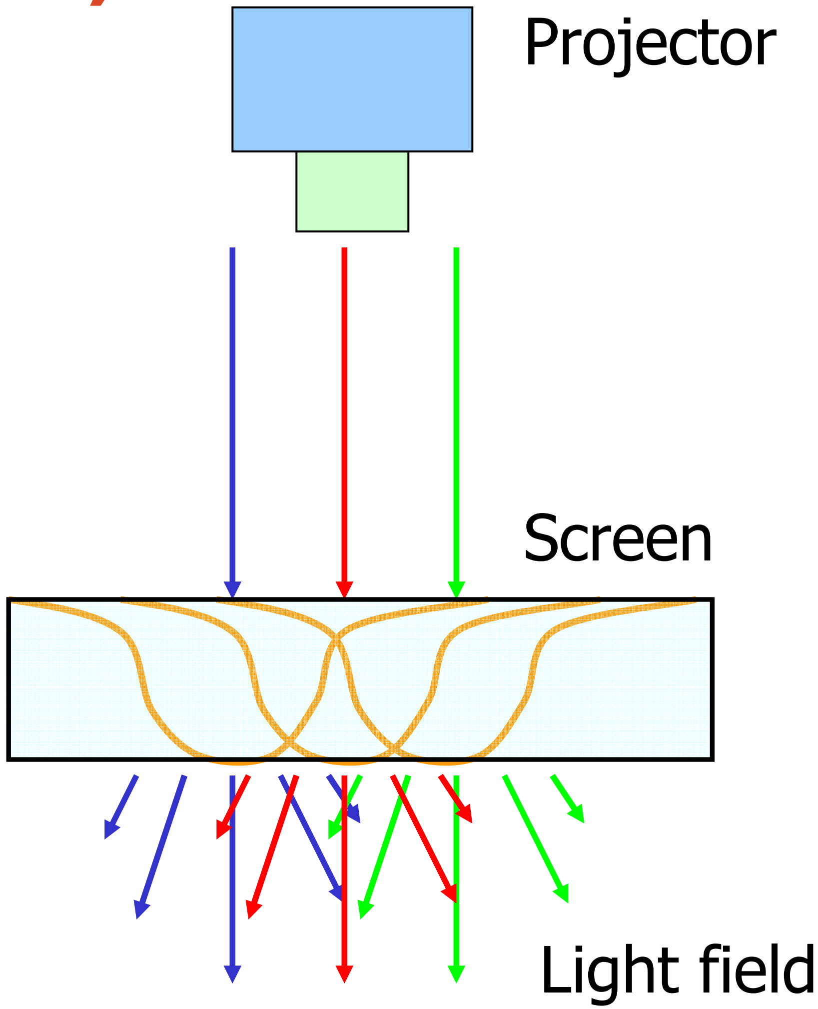
# Display concept (1/2)

- specially arranged projector array and a holographic screen
- each projector emits light beams toward a subset of the points of the holographic screen
- side mirrors increase the available light beams count



# Display concept (2/2)

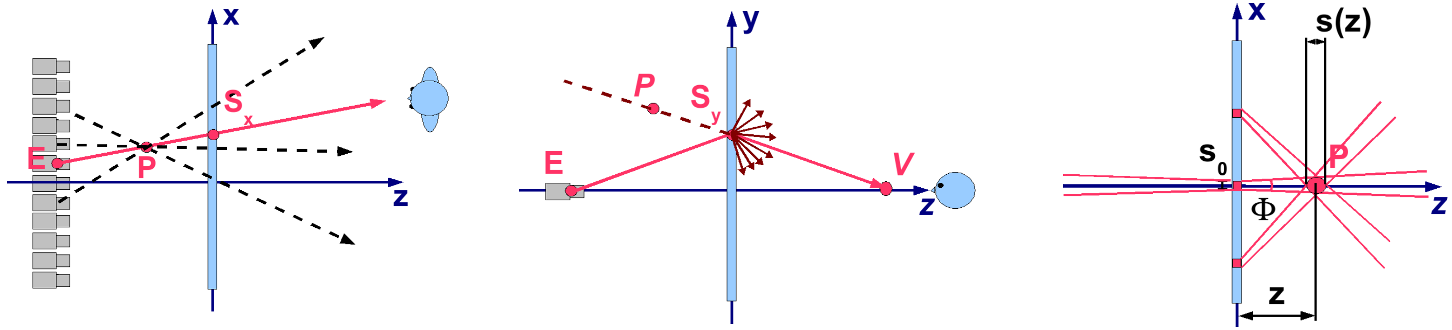
- The holographic screen enables selective directional transmission of light beams
  - Horizontally, sharply transmissive
  - Vertically, the screen scatters widely
- Angular light distribution characterized by a wide plateau and steep Gaussian slopes
  - homogeneous light distribution and continuous 3D view with no visible crosstalk



# Light field geometry

- Control light beams as if emitted from physical objects
- Rendered scene reconstruction
  - Precompute projection parameters
  - Generate multiple views for the same image
- Geometric calibration as a two-step approach
  - Projectors position and frustum found through parametric optimization
  - Error correction with post-rendering 2D image warp

# Projecting graphics




- The renderer assumes a virtual viewer $V(v_y, v_z)$ in order to fix the vertical viewing angle
- Screen position *S* for a virtual point *P* as projected by emitter *E*
- Normalized projected coordinates with respect to image rectangle *R*
- Depth dependent spatial resolution

$$S_x = E_x - E_z \cdot \frac{E_x - P_x}{E_z - P_z}$$

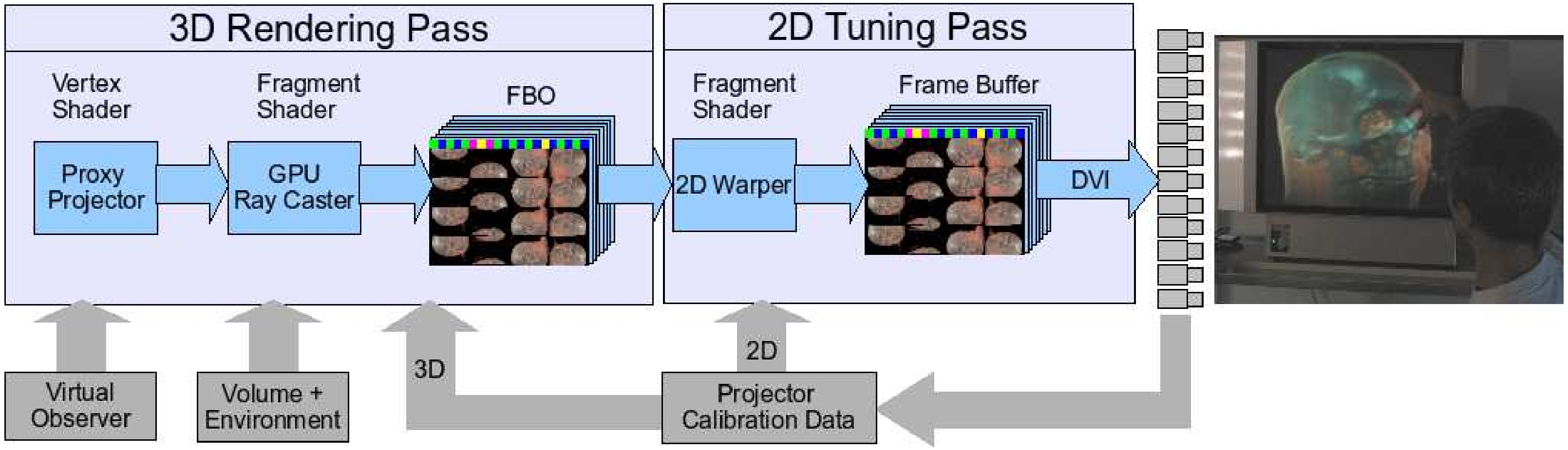
$$S_y = V_y - V_z \cdot \frac{V_y - P_y}{V_z - P_z}$$

$$H_x = \frac{2(S_x - E_x) - (R_x^+ + R_x^-)}{R_x^+ - R_x^-}$$

$$H_y = \frac{2(S_y - E_y) - (R_y^+ + R_y^-)}{R_y^+ - R_y^-}$$

$$H_z = -\frac{P_z}{V_z}$$

# GPU-based volume ray casting (1/2)



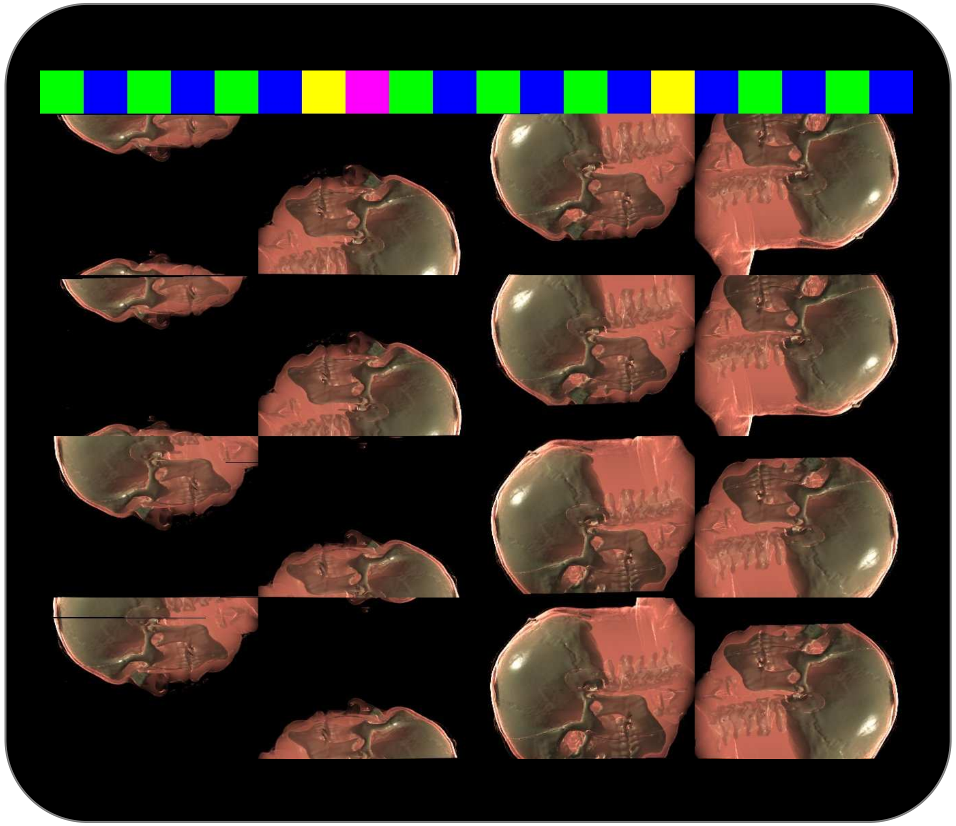

- Two-pass approach typical of multi-projectors display
  - Off-screen rendering to a frame-buffer-object
  - Geometry and color correction through 2D warping
- Modified GPU ray-casting
  - MCOP cannot be recast into the traditional homogeneous matrix
  - Proxy is a coarsely tessellated version (8x8 quads) of a slightly enlarged bounding volume

# GPU-based volume ray casting (2/2)

- For each fragment
  - Screen pixel position ***S*** and ray direction ***d*** are computed using the MCOP projection and transformed in local texture coordinates
  - ray entry point and integration lengths are computed by clipping the line ***S,d*** against the unit box
  - Fragments with null length are discarded, otherwise renderer performs classic volume sampling and composition
- Mip-mapping takes into account depth dependent spatial resolution

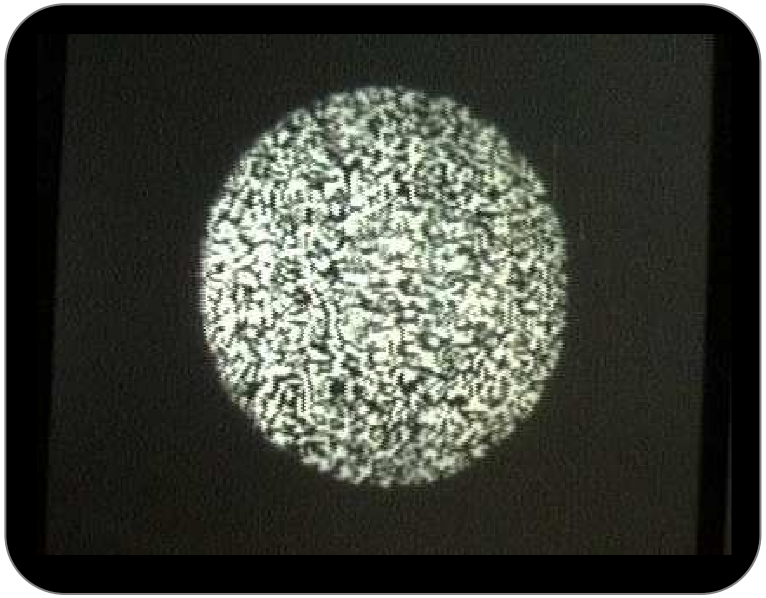
# Prototype system setup

- Display system built by Holografika
  - 7.4M beams/frame
  - 96 fast 320x240 LCD displays
  - FPGA input processing units decoding DVI stream
  - 2D pixel size 1.25 mm, angular accuracy 0.8°
- Athlon64 3300+ PC with a NVIDIA 8800GTX graphics board
- C++, OpenGL, Cg shaders implementing volume ray casting with different composition techniques

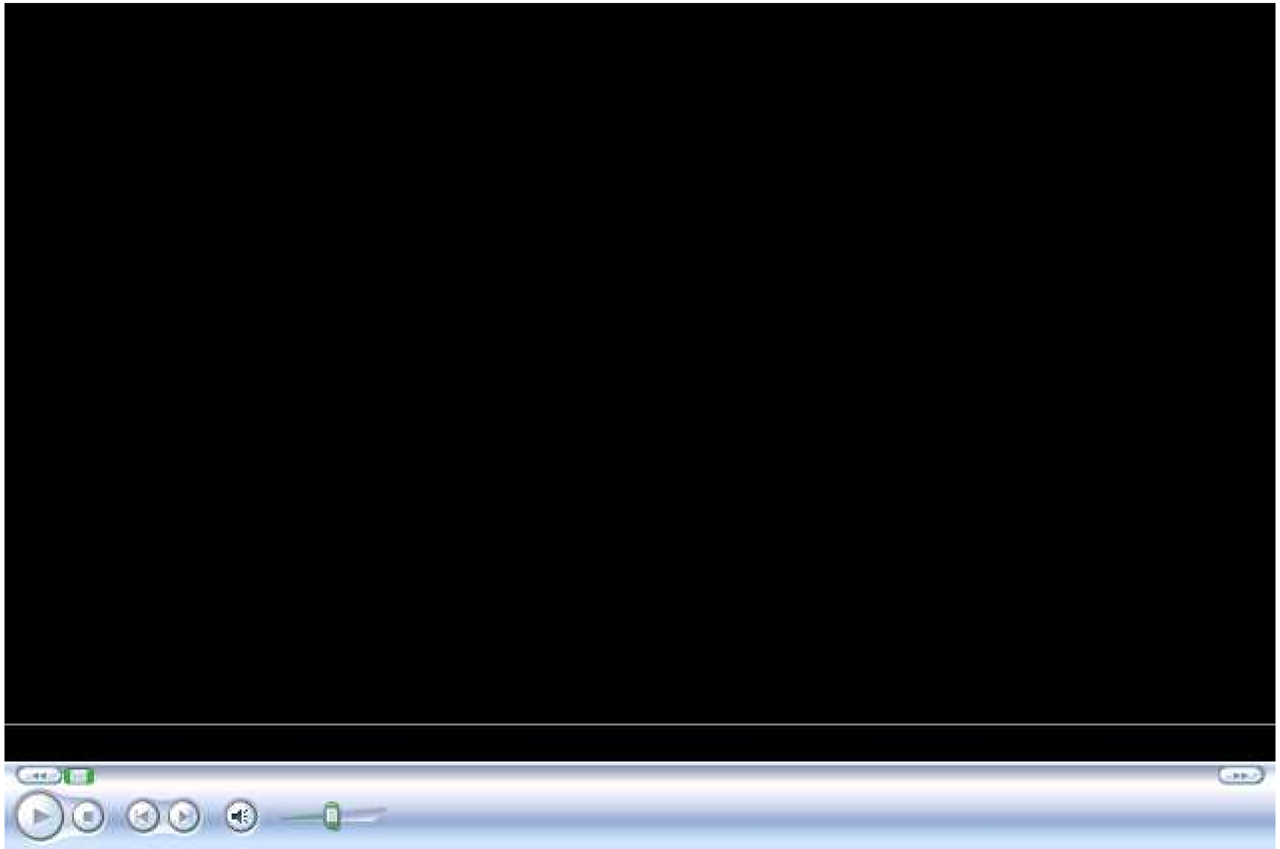


# Evaluation

- Enhanced 3D understanding
  - stereopsis and parallax effects through ego-motion
  - 2IFC perceptual experiment enforced this hypothesis
- Users rapidly recover all depth cues to instantaneously recognize complex structures
  - Very useful for analysis of angiography datasets



# Interactive sequences



# Limitations

- Rendering performance during interaction
  - frame rate improved by reducing the pixel count and doubling the integration step size
  - misalignment between tiles visible when objects are moved with a too slow refresh rate
- Distortion artifacts
  - occur when users move away from the expected optimal viewing position

# Conclusions

- Today we have introduced volume rendering techniques, possible optimizations and acceleration using GPU ray-casting.
- We have reviewed one state-of-the art approach of how we can visualize massive volume datasets on a commodity PC platform
- Furthermore, we have seen how we can enhance 3D visual understanding and interaction using a new generation of light field displays

# Interactive visualization of medical datasets

**José A. Iglesias Guitián**
[ jalley@crs4.it ]

Thanks : )

CRS4 Visual Computing
http://www.crs4.it/vic/

**http://3dah.miralab.unige.ch**