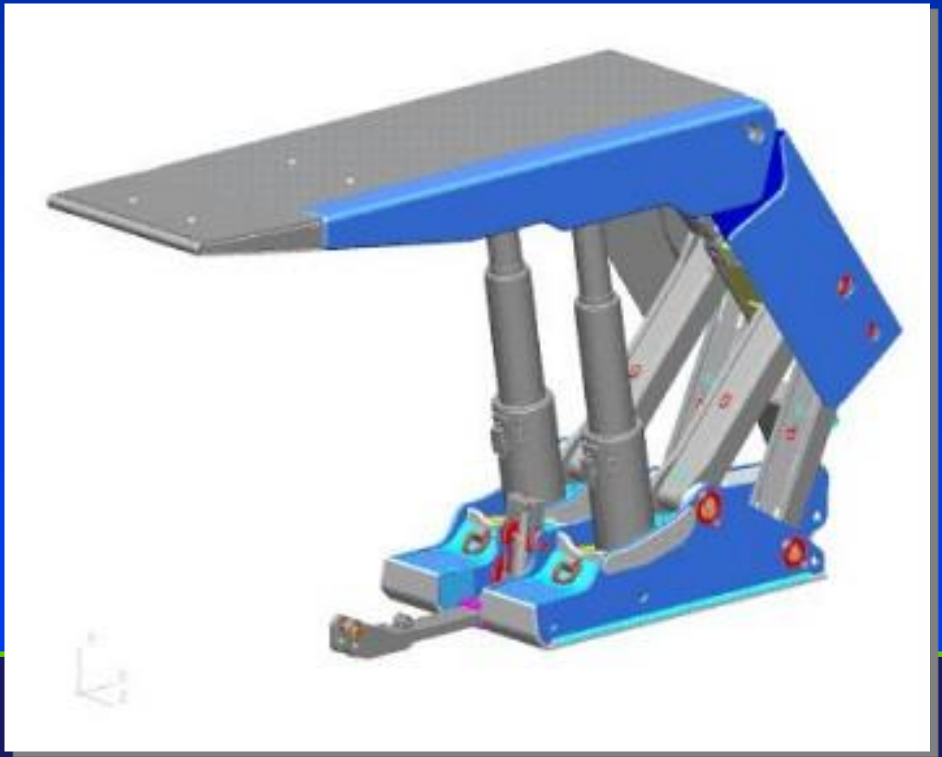
What is Computer Graphics?

- Computational process of generating images from models and/or datasets using computers
- This is called *rendering (computer graphics was traditionally considered as a rendering method)*
- A rendering algorithm converts a geometric model and/or dataset into a picture

What is Computer Graphics?

This process is also called *scan conversion* or *rasterization*

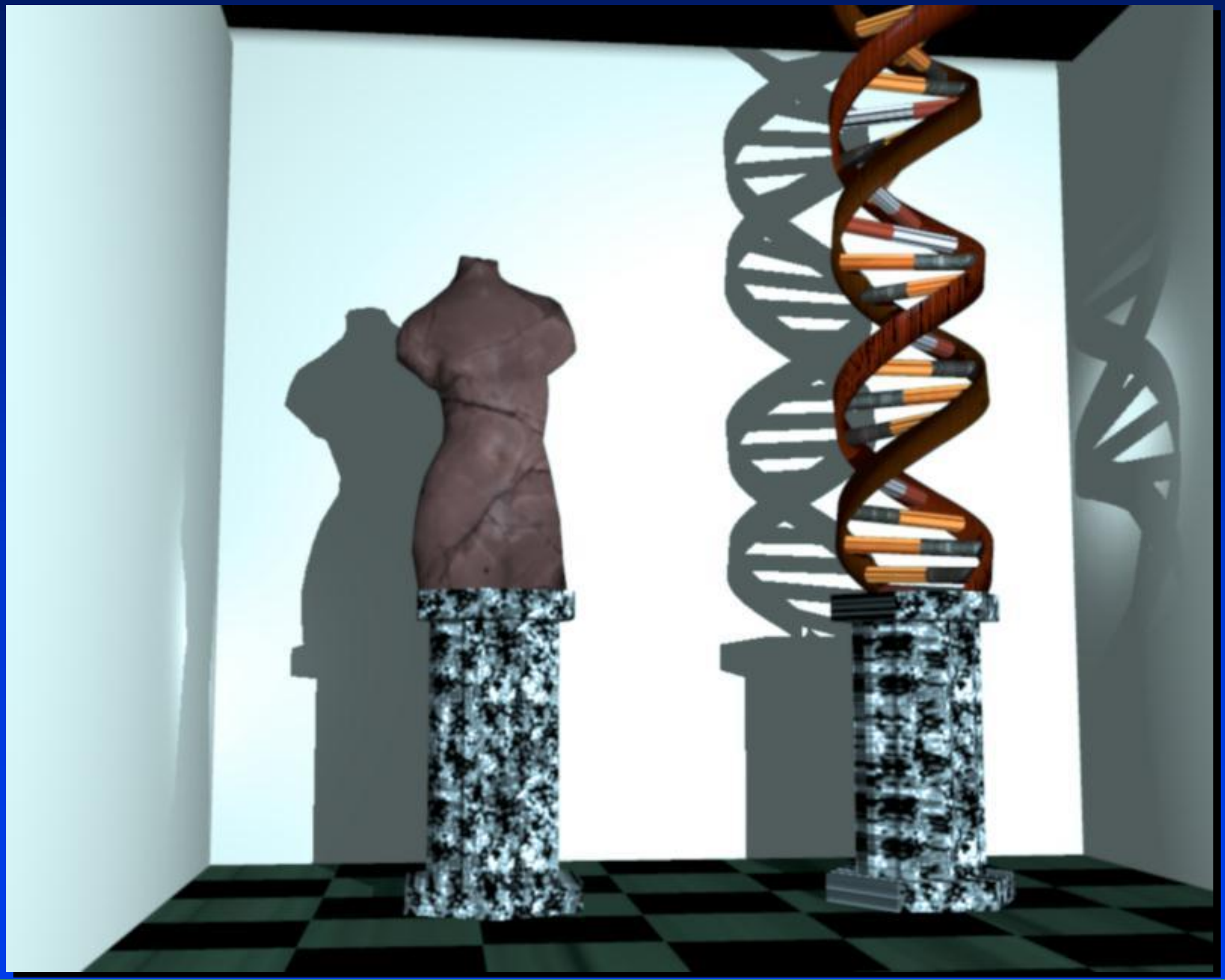
How does Visualization fit in here?



Computer Graphics

- Computer graphics consists of :
  1. Modeling (representations)
  2. Rendering (display)
  3. Interaction (user interfaces)
  4. Animation (combination of 1-3)
- Usually “computer graphics” refers to rendering

Computer Graphics Components



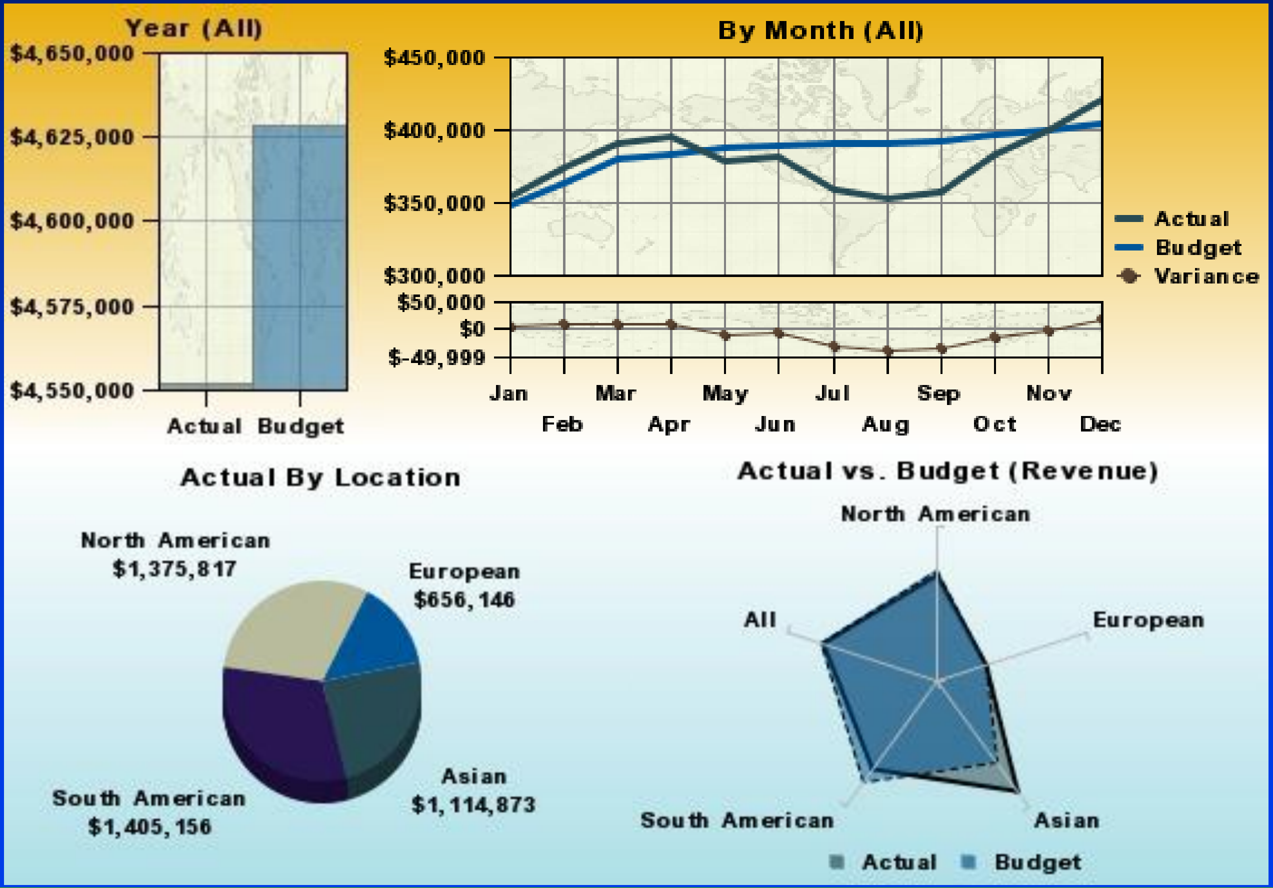
Surface Rendering

- Surface representations are good and sufficient for objects that have *homogeneous* material distributions and/or are not *translucent* or *transparent*
- Such representations are good only when object boundaries are important (in fact, only boundary geometric information is available)
- Examples: furniture, mechanical objects, plant life
- Applications: video games, virtual reality, computer-aided design

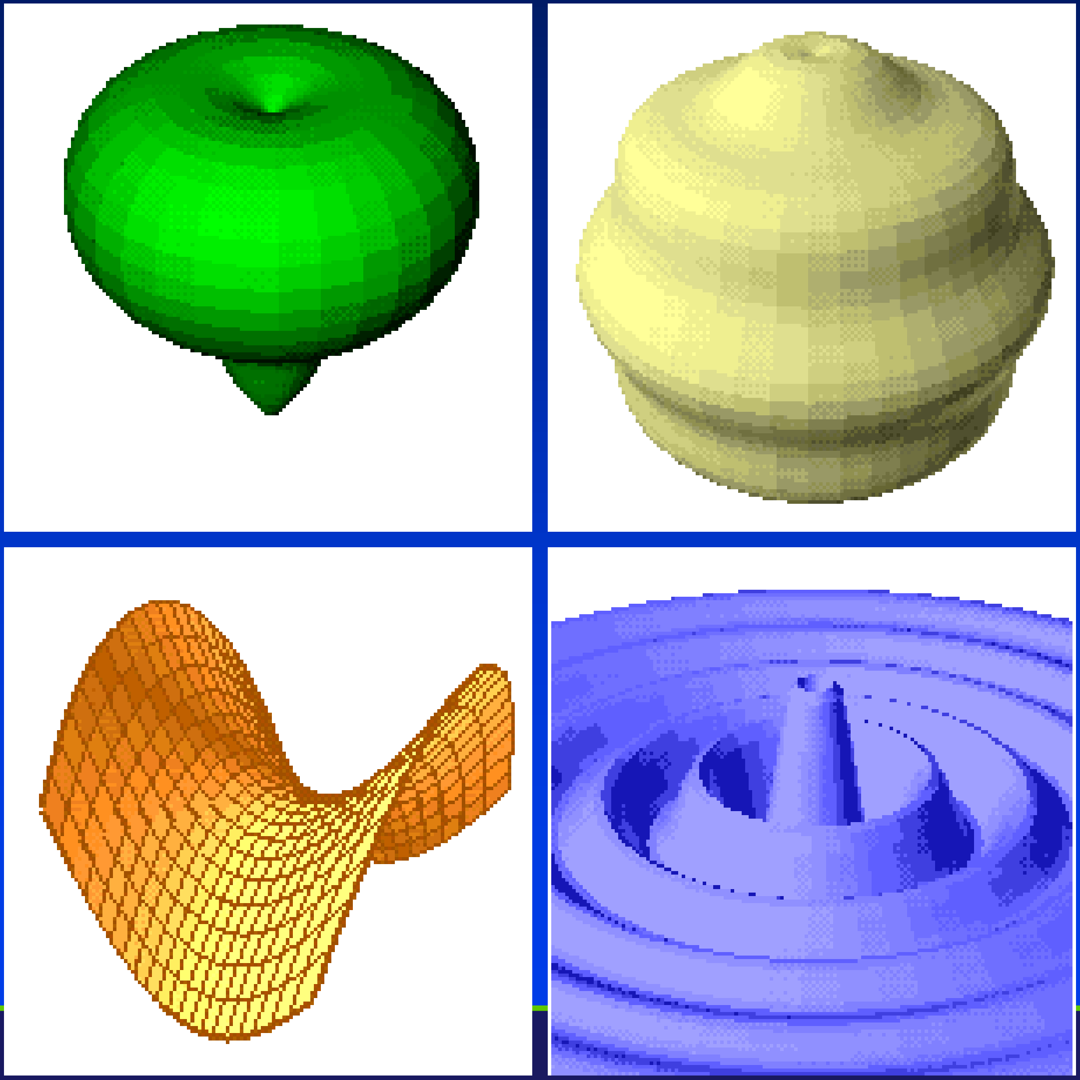
Applications

Earlier Days of Computer Graphics

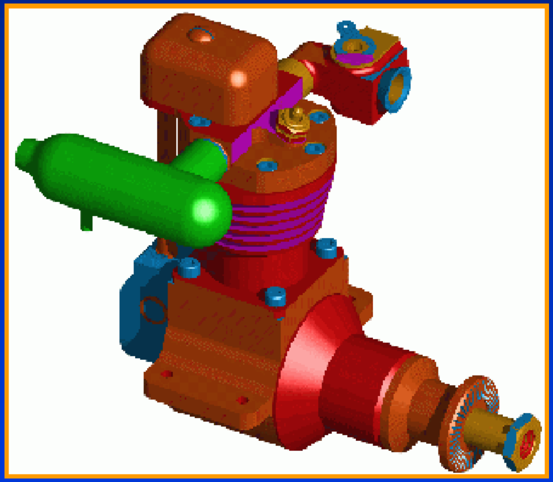
- Visual display of data (graphs and charts)

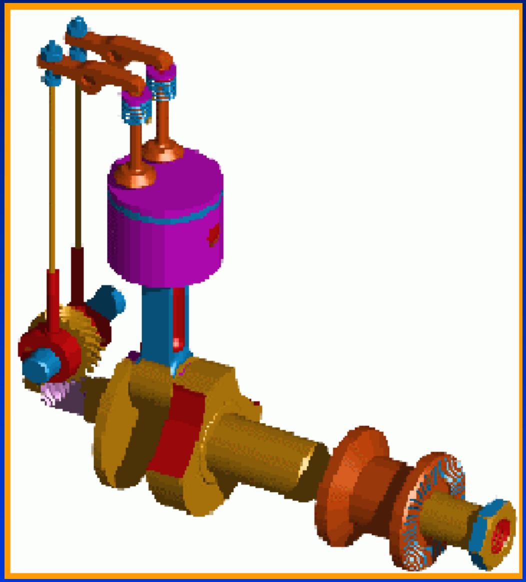


Mathematical Function Plots

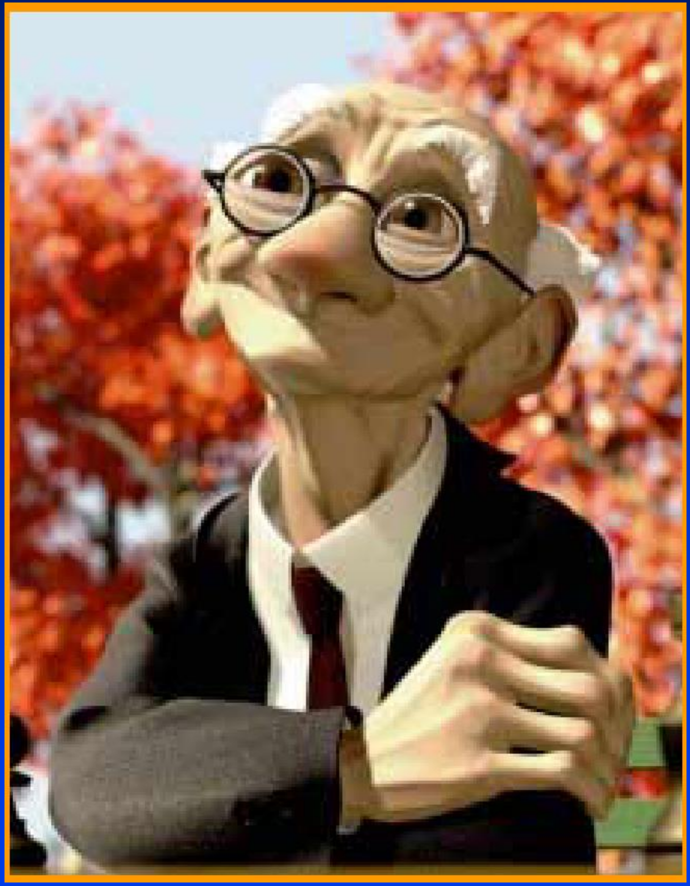


Computer Aided Design (CAD)

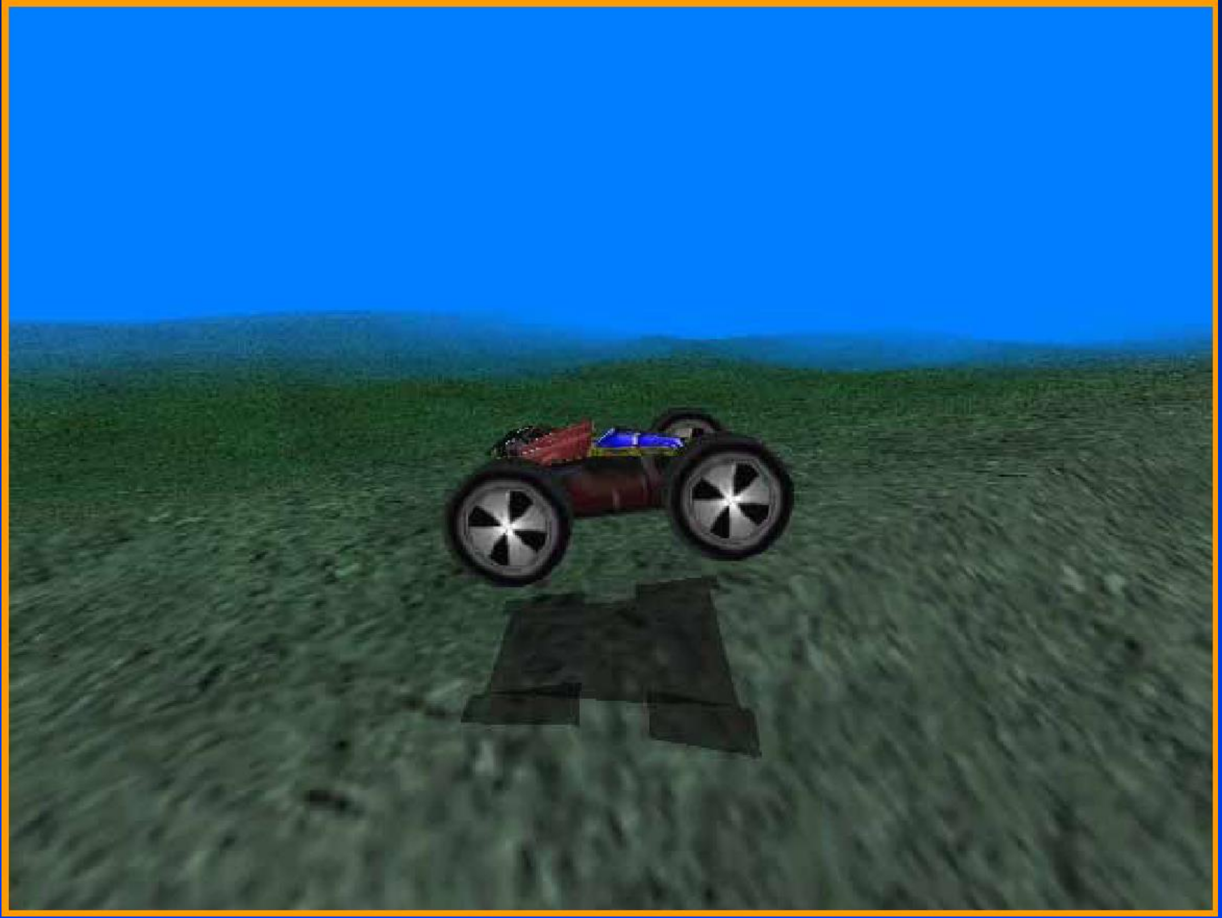




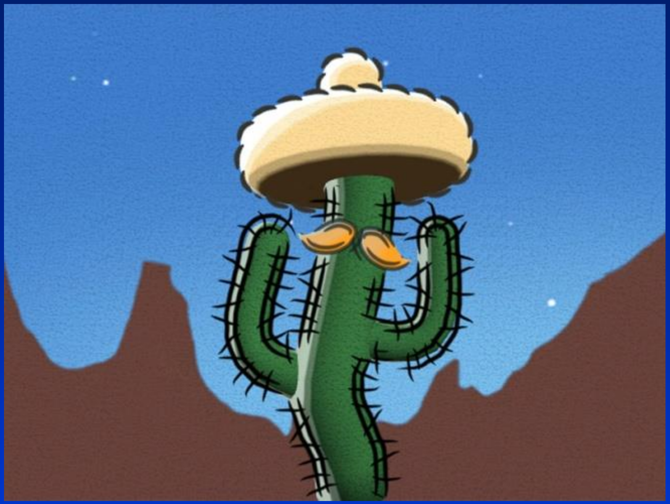
Computer Animation

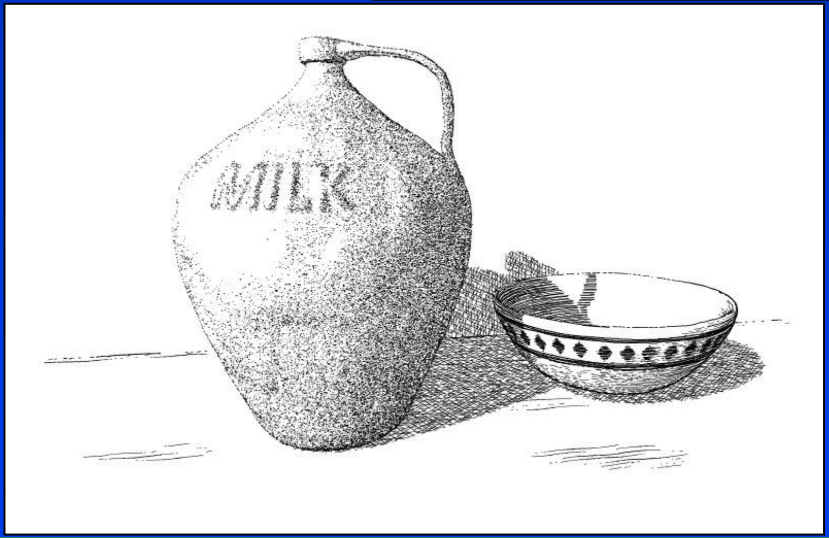
 

Video Games



Digital Art

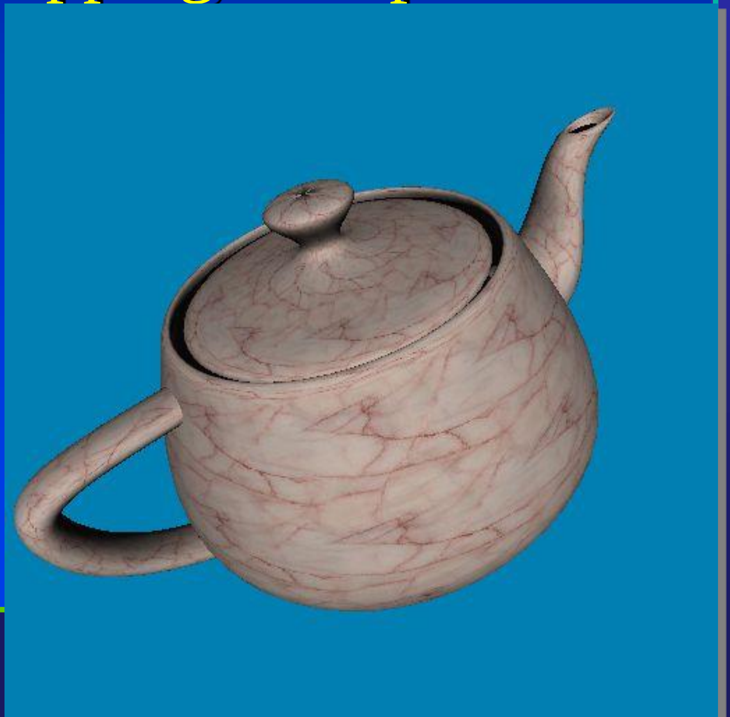




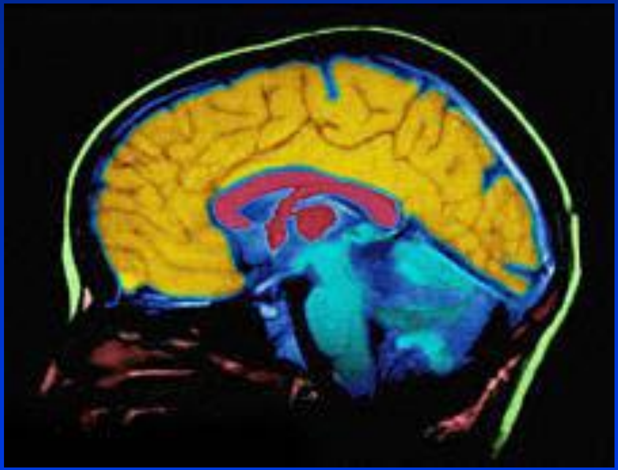
Architecture

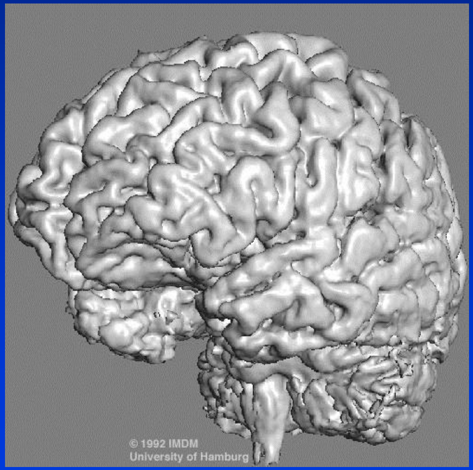
Surface Rendering – Pros and Cons

- Good: explicit distinction between inside and outside makes rendering calculations easy and efficient
- Good: hardware implementations are inexpensive
- Good: can use tricks like texture mapping to improve realism
- Bad: an approximation of reality
- Bad: does not let users peer into and thrsee ough objects

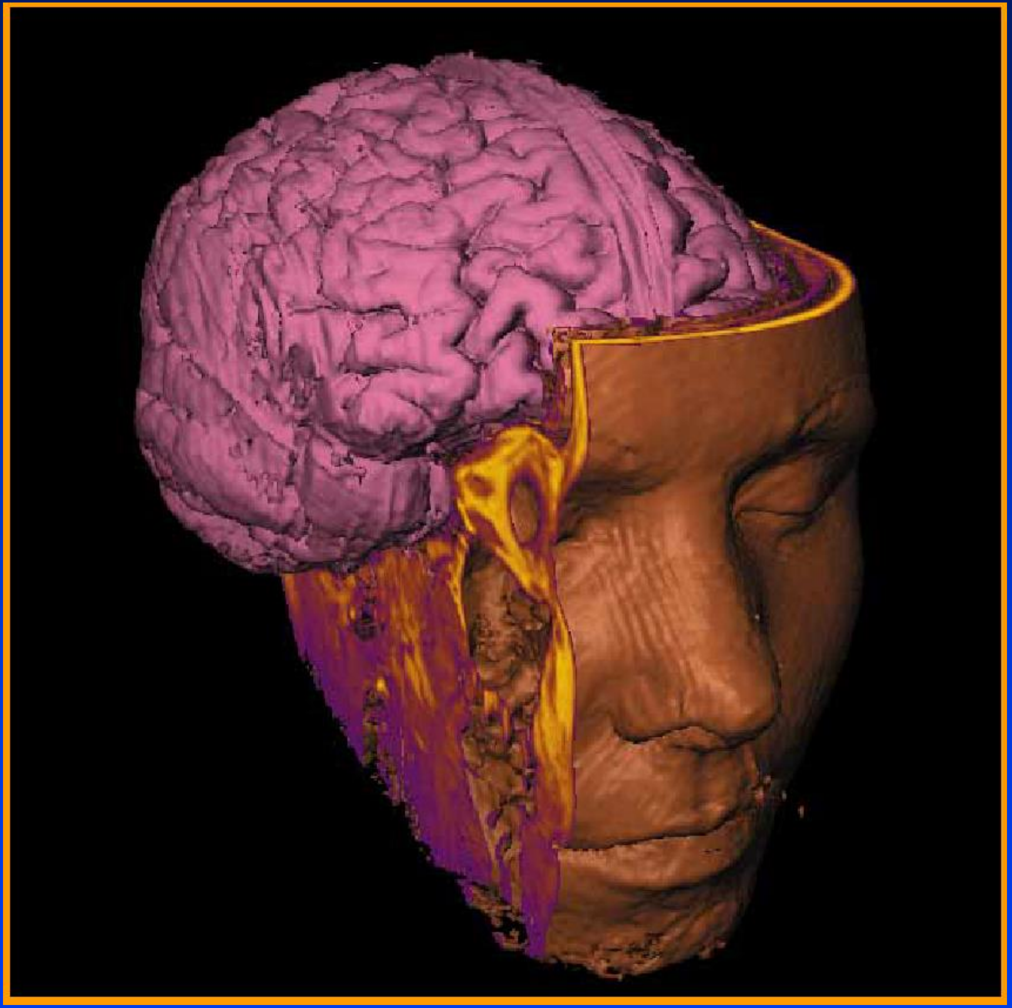


Scientific Visualization

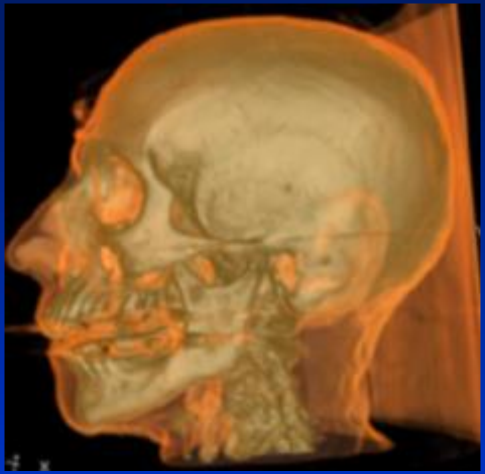


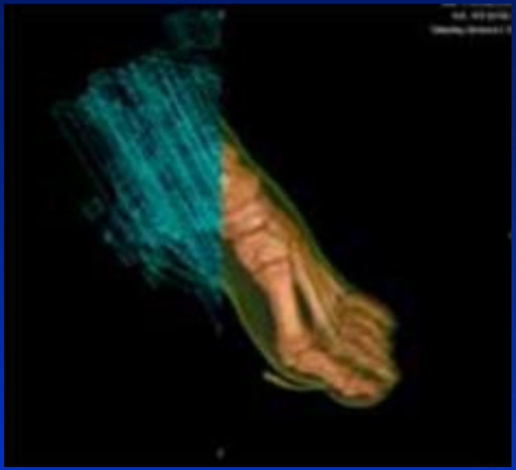


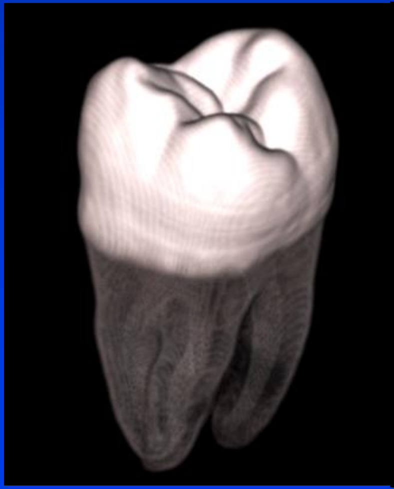
Visualization (Isosurfaces)



Visualization (Volume Rendering)







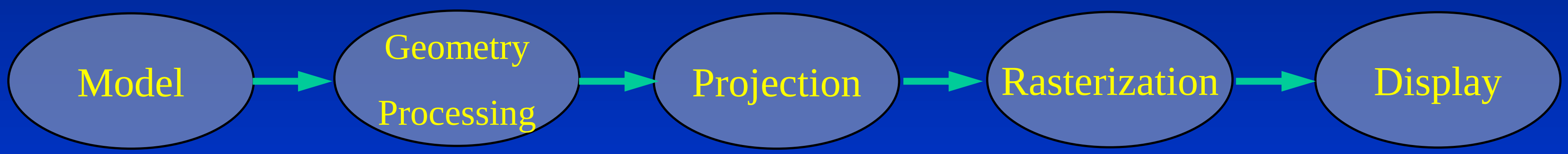
Computer Simulation

Volume Visualization

- Surface rendering doesn't work so well for clouds, fog, gas, water, smoke and other *amorphous phenomena*
- "amorphous" = "without shape"
- Surface rendering won't help users if we want to explore objects with very complex internal structures
- Volume graphics provides a good technical solution to these shortcomings of surface graphics
- Volume graphics includes *volume modeling (representations)* and *volume rendering* algorithms to display such representations

Computer Display Pipeline

Computer Display Pipeline



Graphics Pipeline

Modeling → Rendering

→

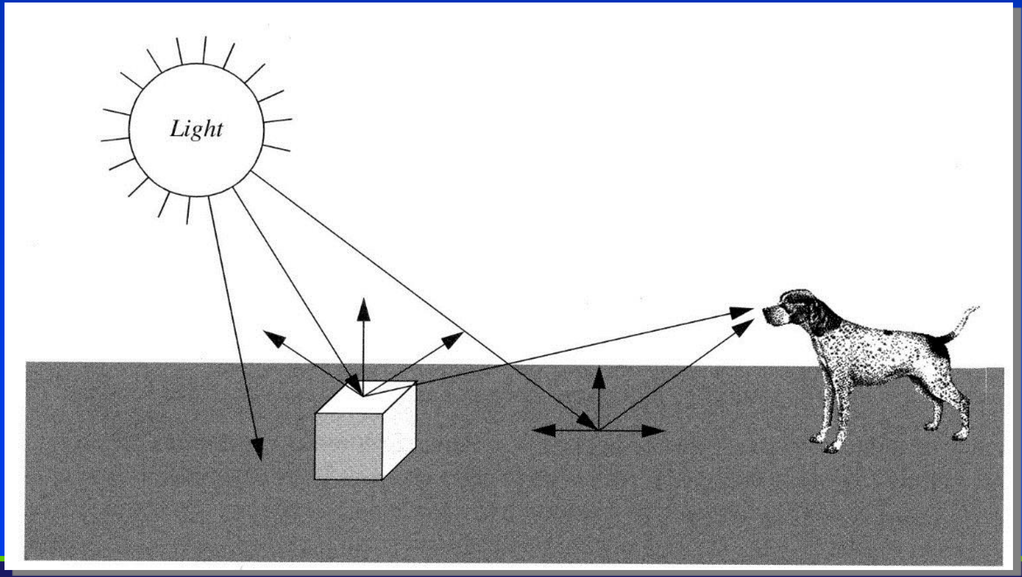
Cameras, Lights, and Objects (Datasets)

- In reality, how are we able to see things in the real world?
- What's the computational process that occurs?
- Let us start this process:
  1. Open eyes
  2. Photons from light source strike object
  3. Bounce off object and enter eye
  4. Brain interprets image you see

We will have to simulate this process computationally!

Cameras, Lights, and Objects (Datasets)

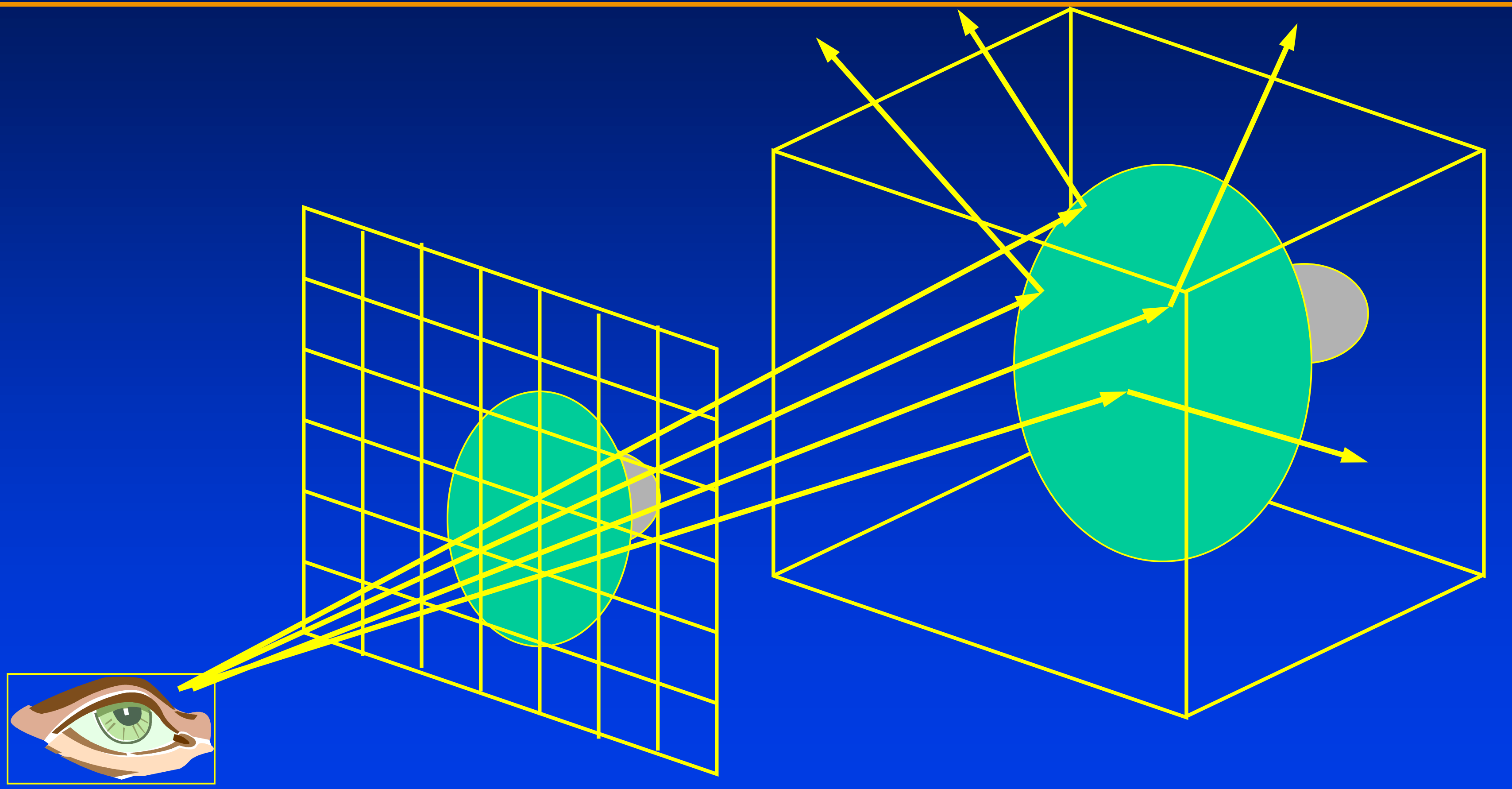
- Rays of light emitted by light source
- Some light strikes object we are viewing
  - Some light absorbed
  - Rest is reflected
  - Some reflected light enters into our eyes



Cameras, Lights, and Objects (Datasets)

- How do we simulate light transport in a computer?
- Several ways
- *Ray-tracing* is one
- Start at eye and trace rays the scene
- If ray strikes object, bounces, hits light source → we see something at that pixel
- Most computer applications don't use it. Why?
- With many objects very computationally expensive

Surface Ray-Tracing



Rendering Processes: Image-Order and Object-Order

- Ray-tracing is an *image-order* process: operates on *per-pixel basis*
- Determine for each ray which objects and light sources ray intersects
- Stop when all pixels processed
- Once all rays are processed, final image is complete
- *Object-order* rendering algorithm determines for each object in scene how that object affects final image
- Stop when all objects processed

Rendering Processes: Image-Order and Object-Order

- Image-order approach: start at upper left corner of picture and draw a dot of appropriate color
- Repeat for all *pixels* in a left-to-right, top-to-bottom manner
- Object-order approach: paint the sky, ground, trees, barn, etc. back-to-front order, or front-to-back
- Image-order: very strict order in which we place pigment
- Object-order: we jump around from one part of the regions to another

Rendering Processes: Image-Order and Object-Order

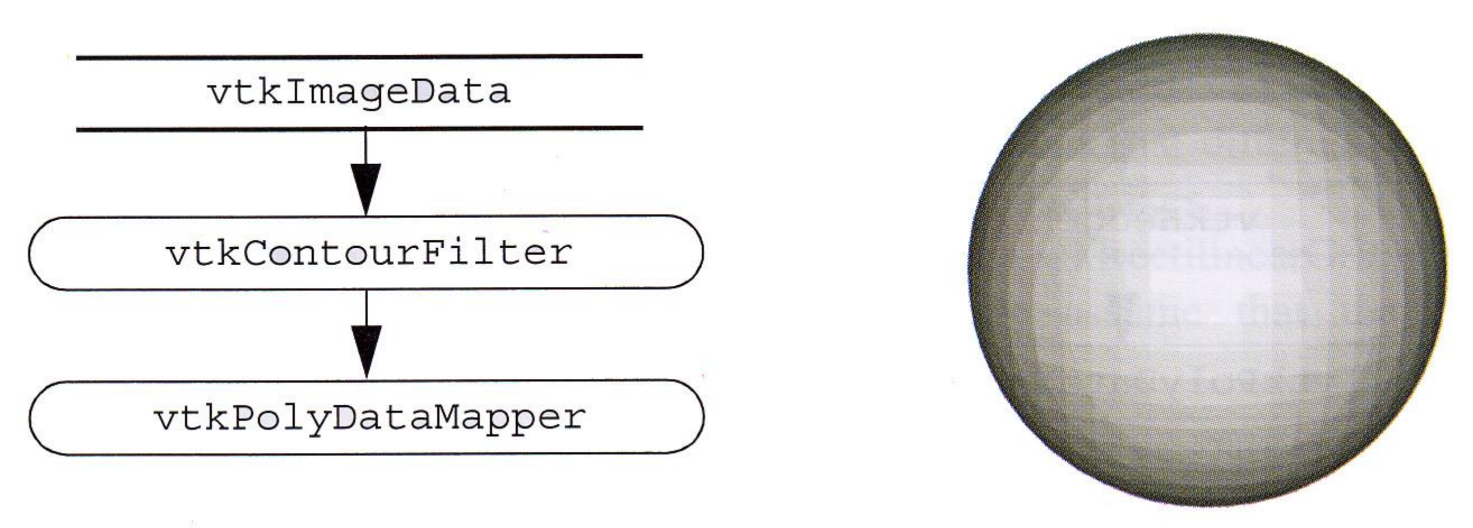
- Advantages and disadvantages of each
- Ray-tracing can produce very realistic looking images, but is very computationally expensive
- Object-order algorithms more popular because hardware implementations of them exist
- Not as realistic as ray-tracing

Surface Rendering

- We have considered interaction between light rays and object boundaries
- This is called *surface rendering* and is part of *surface graphics*
- Computations take place on boundaries of objects
- Surface graphics employs *surface rendering* to generate images of *surface' mathematical and geometric representations*

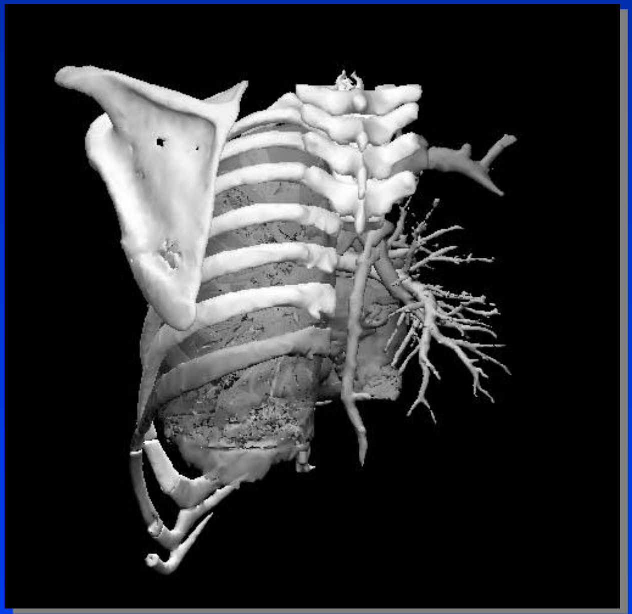
Mathematical Surfaces (Sphere)

- Equation of a sphere: $x^2 + y^2 + z^2 = r^2$
- How thick is the surface?
- Are there objects in real world thickness zero?



Surface Graphics

- Can you think of objects or phenomena for which this approach to rendering will fail?
- When is a surface representation not good enough?
- Would a surface representation suffice to represent the internal structure of the human body?



From Surface Graphics to Volume Visualization

- Visualization: transformation of data into graphical form
- Object-oriented-based approach: data are the objects, transformations are the methods

Data Visualization Example

- Usually we **evaluate** the equation of a sphere for a particular radius, r: $$x^2 + y^2 + z^2 = r^2$$
- Suppose we evaluate it for different values of r?
- We get a solid sphere
- Now imagine we evaluate it for any value of *x*, *y*, *z* and *r* $$F(x, y, z) = x^2 + y^2 + z^2 - r^2$$
- We get what's called a **field function**
- You plug in some values for *x*, *y*, *z*, *r* and get some number. That number is "located" at position (*x*, *y*, *z*)

Data/Model Visualization Example

- A **quadric** is a special function with maximum

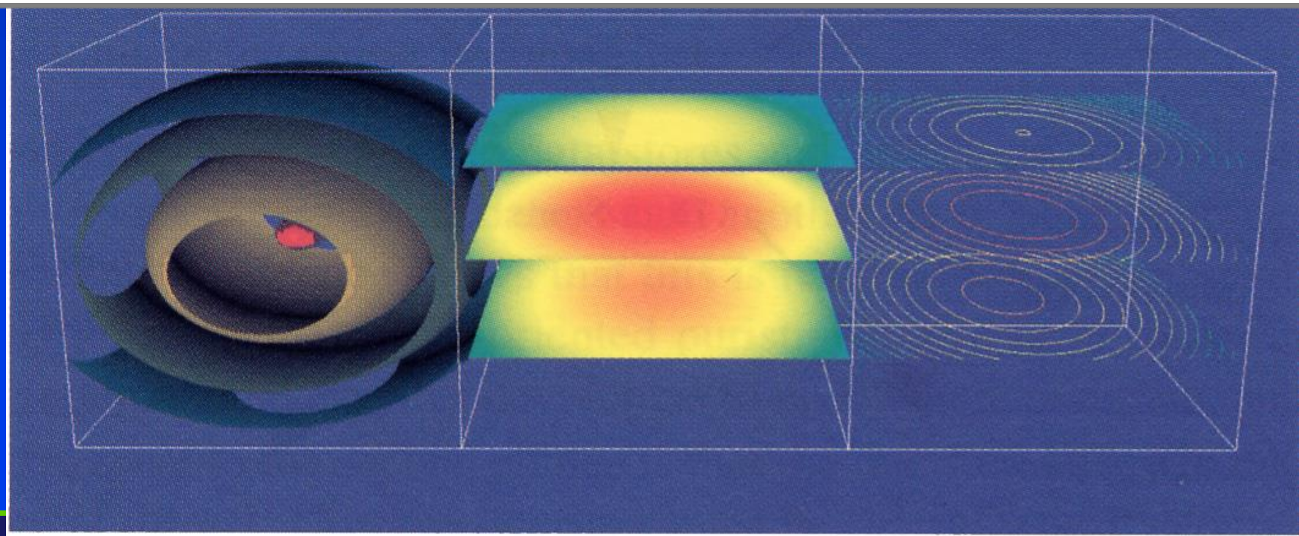
$$F(x, y, z) = a_0x^2 + a_1y^2 + a_2z^2 + a_3xy + a_4yz + a_5xz + a_6x + a_7y + a_8z + a_9$$

- A **solid** sphere is an example of a quadric with a_3, a_4, a_5, a_6, a_7, and a_8 all equal to zero
- If those values aren't zero, we get some pretty strange shapes
- Imagine squishing a **solid** rubber ball (i.e., not a hollow ball, like a tennis ball)

Data Visualization Example

- If we plug in x, y, z, r for any quadric, we can get some very strange-looking **field functions**. Here's an example:

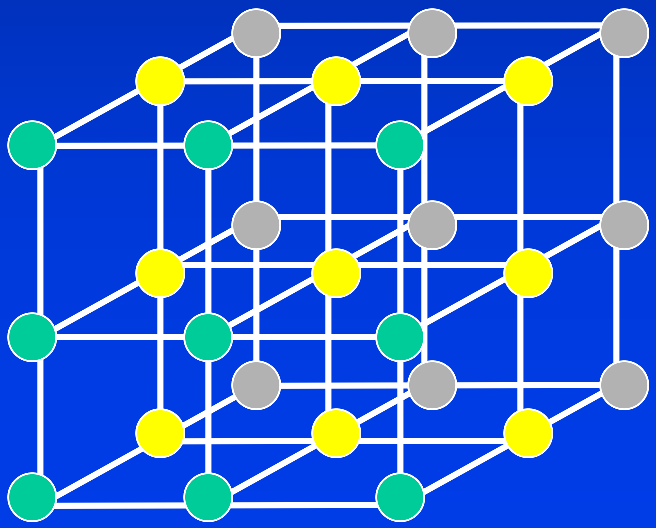
$$F(x, y, z) = a_0x^2 + a_1y^2 + a_2z^2 + a_3xy + a_4yz + a_5xz + a_6x + a_7y + a_8z + a_9$$



(a) Quadric visualization (Sample.cxx)

Volumetric Representations

- A volumetric data-set is a 3D regular grid, or *3D raster*, of numbers that we map to a gray scale or gray level
- Where else have you heard the term *raster*?
- An 8-bit volume could represent 256 values [0, 255]
- Typically volumes are at least 200^3 in size, usually larger
- How much storage is needed for an 8-bit, 256^3 volume?

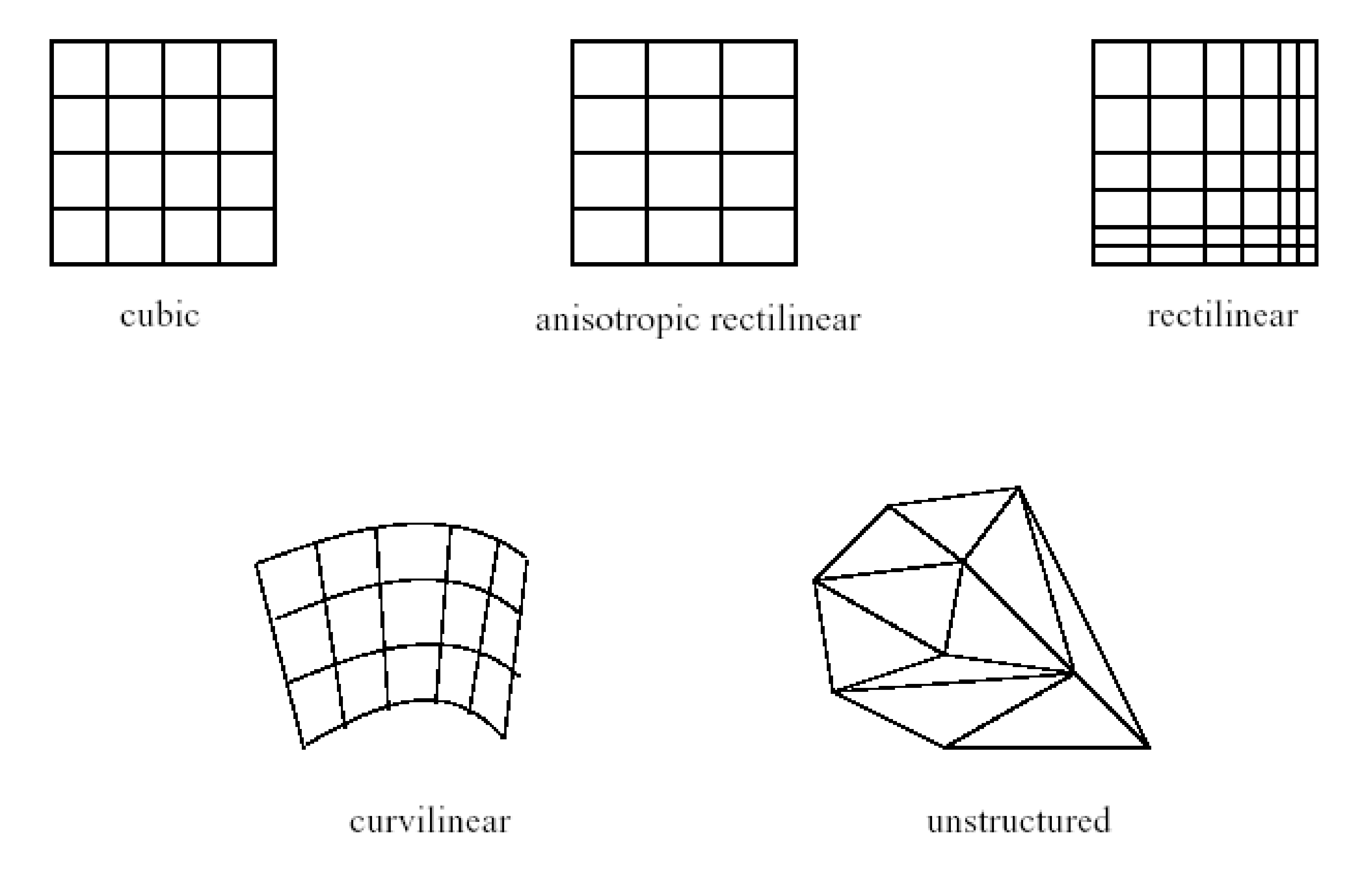


Volume Visualization

- Volumetric objects have interiors that are important to the rendering process (what does that mean?)
- Interior affects final image
- Imagine that our rays now don't merely bounce off objects, but now can penetrate and pass through
- This is known as *volumetric ray-casting* and works in a similar manner to surface ray-tracing

Volume Visualization (Rendering)

Volume Grid Types



Volumetric Data Format

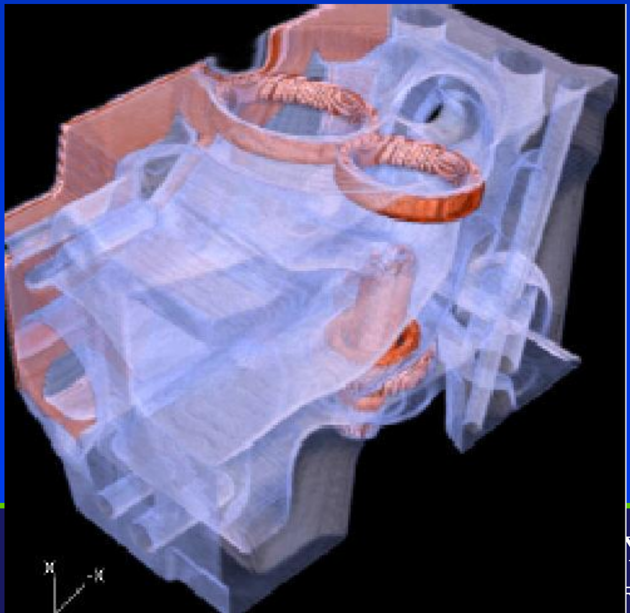
- As one might expect, the format of the input data generally depends on the source used to generate the data
- Some general classes exist:
  – rectilinear
  – curvilinear
  – unstructured (or irregular)
- We will focus most of our attention on rectilinear data

Volumetric Data

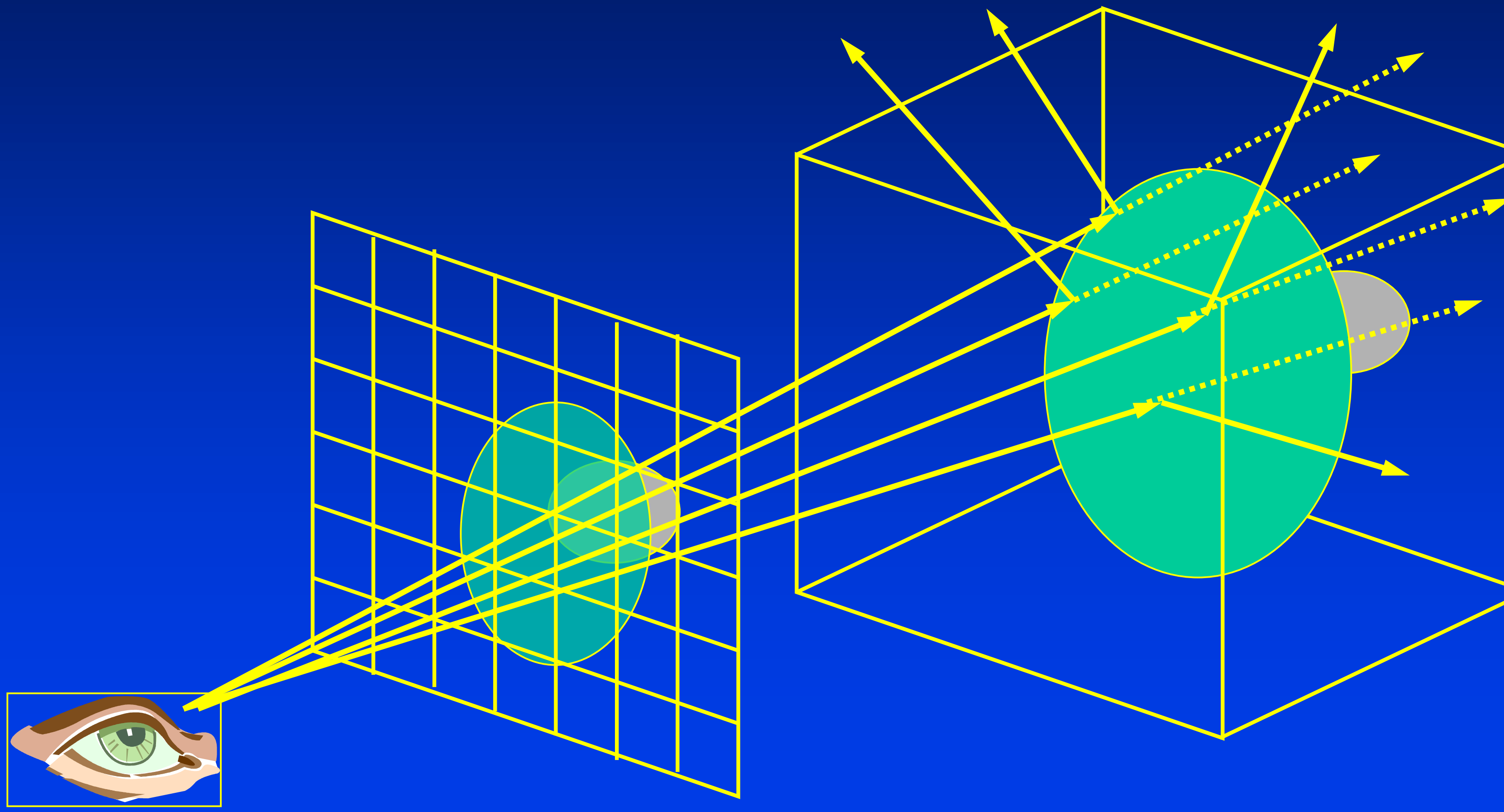
- A volumetric data set is typically a set *V* of samples (*x*,*y*,*z*,*v*) called *voxels*, short for "volume elements" (3D analog of pixels)
- If *v* can be only either 0 or 1, we call this a binary volume
- Usually, we have 8 or more bits per voxel
- Voxel may be a scalar, or could be vector-valued
- Data could even be time-varying: (*x*,*y*,*z*,t,*v*)
- Can you think of some examples of time-varying, volumetric data or objects?

Volume Visualization

- Volume visualization is a method of extracting meaningful information from volumetric data using interactive graphics and imaging
- Volume data representation, modeling, manipulation, and rendering
- Volume data: 3D entities that have information inside them, but may not consist of tangible edges or surfaces
- Obtained via sampling, simulation, or modeling techniques
- Used in many areas, such as medical imaging, CFD, study of mechanical parts

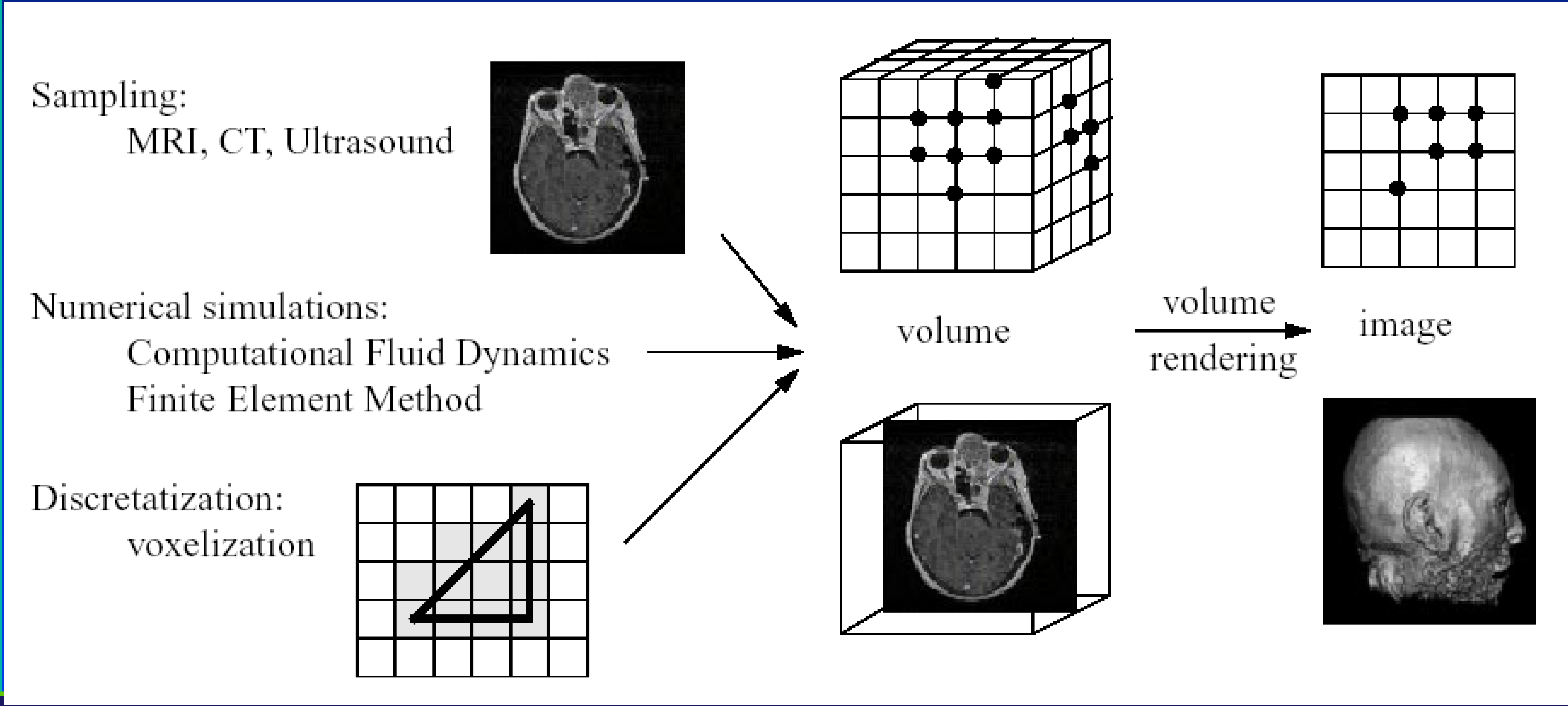


Volumetric Ray-Tracing



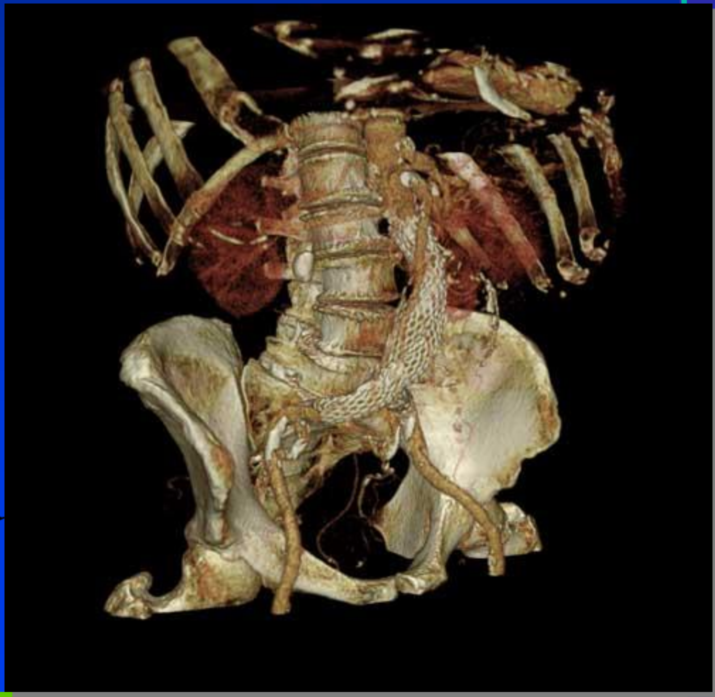
Volume Rendering

•The process of generating a 2D image from the 3D volume is called *volume rendering*



Volume Rendering

- In volume rendering, imaginary rays are passed through a 3D object that has been *discretized* (e.g., via CT or MRI)
- As these *viewing rays* travel through the data, they take into account of the *intensity* or *density* of each datum, and each ray keeps an accumulated value

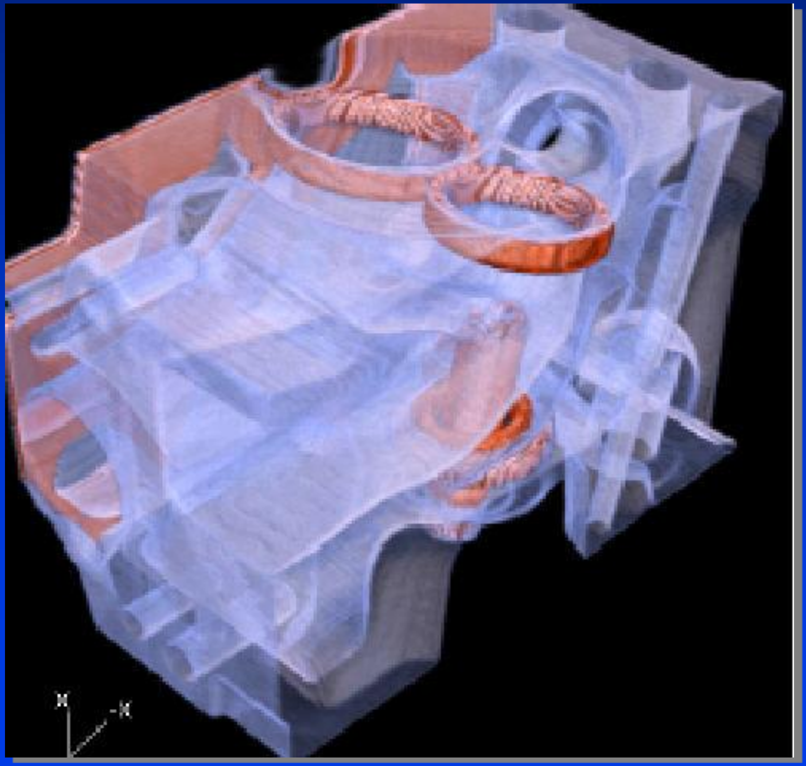


Volume Rendering

- As the rays leave the data, they comprise a sheet of accumulated values
- These values represent the volumetric data *projected* onto a two-dimensional image (the screen)
- Special mapping functions convert the grayscale values from the CT/MRI into color

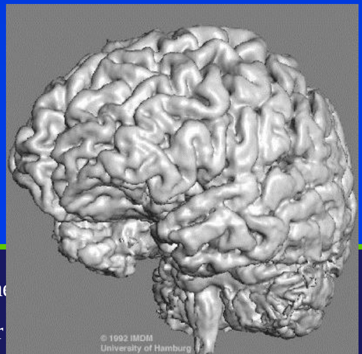
Volume Rendering

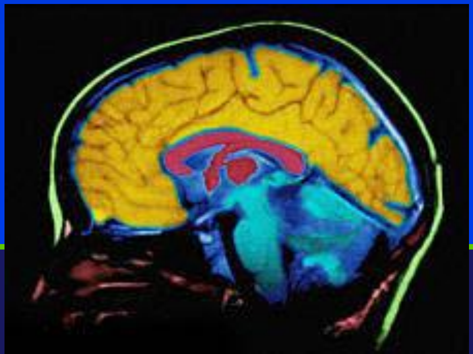
- *Semi-transparent rendering*

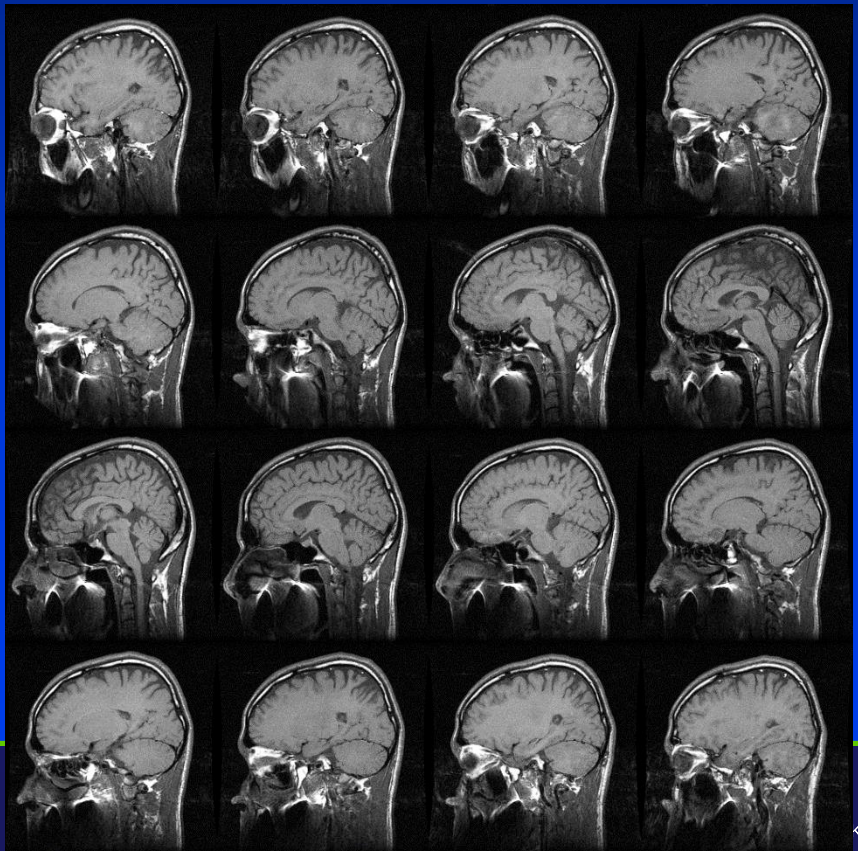


Volume Rendering

- Volume rendering is a method of displaying volumetric data as a two-dimensional image
- For instance, the volume may be a rectangular grid of *samples* acquired using an MRI scanner
- It is the task of volume rendering to convert this complex 3D data into an effective 2D visualization
- There exist several distinct classes of volume rendering algorithms, some of which use traditional surface-based 3D graphics





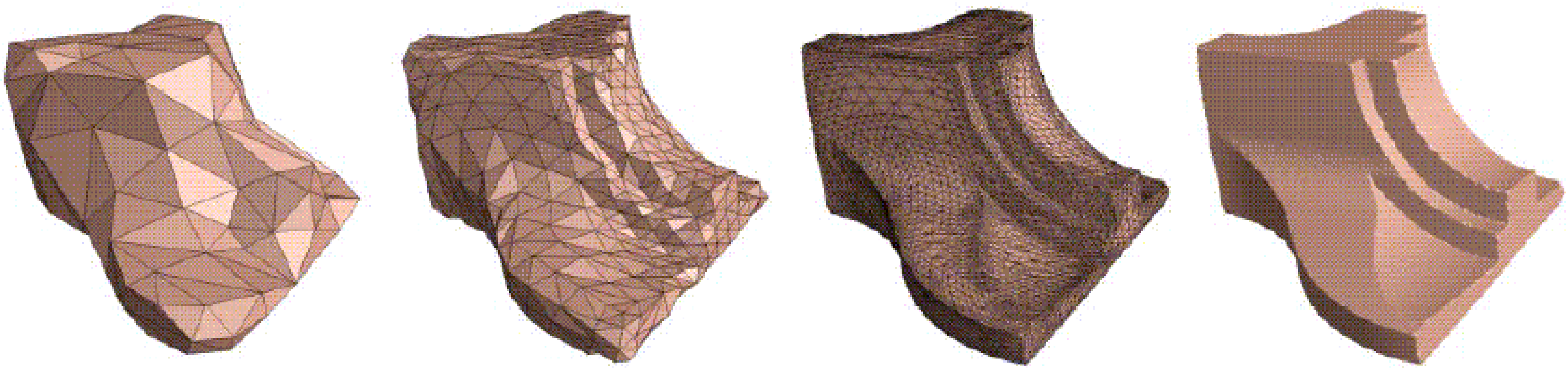


Volume Visualization with Direct Rendering

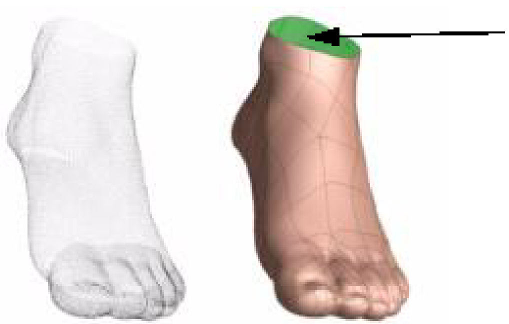
- Good: maintains a representation that is close to the underlying fully-3D object (but discrete)
- Good: can achieve a level of realism (and "hyper-realism") that is unmatched by surface graphics
- Good: allows easy and natural exploration of volumetric datasets
- Bad: extremely computationally expensive (high rendering complexity)!
- Bad: hardware acceleration is complex and very costly ($3000+ vs $200+ for surface rendering)

Surface Modeling and Rendering

- a mesh of polygons:



200 polys 1,000 polys 15,000 polys



an “empty” foot

Surface – Pros and Cons

- Pros:
  - fast rendering algorithms are available
  - acceleration in special hardware is relatively easy and cheap (many $100 game boards)
  - Rich programming libraries such as OpenGL, Direct3D make it easy to develop surface graphics applications
  - surface realism can be added via texture mapping

Surface – Pros and Cons

- Cons:
  - discards the interior of the object and just maintains the object's shell
  - does not facilitate real-world operations such cutting, slicing, direction
  - does not enable artificial viewing modes such as semi-transparencies, X-ray
  - surface-less phenomena such as clouds, fog, gas are hard to model and represent

Surface Rendering vs. Volume Visualization

- Suppose we wish to animate a cartoon character on the screen
- Should we use surface rendering or volume rendering?
- Suppose we want to visualize the inside of a person's body?
- Now what approach should we use? Why?
- Could we use the other approach as well? How?
- We could visualize body as collection of surfaces

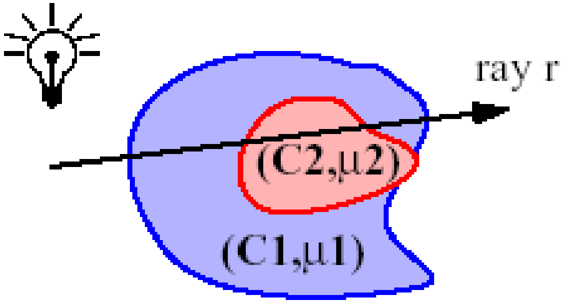
Volume Visualization with Direct Rendering

- Maintains a representation that is close to the underlying fully-3D object (but discrete)
- Models the object as a magic gel that can change its properties at any time.
- Different aspects of the dataset can be emphasized via changes in the functions that translate raw densities into colors and transparencies
- Volume rendering is a formidable technique for the *exploration* of datasets, since when the nature of the data is not known, it is difficult to create the right polygonal mesh

Ray-casting

- Consider a volume consisting of particles:

  - each has color C and light attenuating density μ



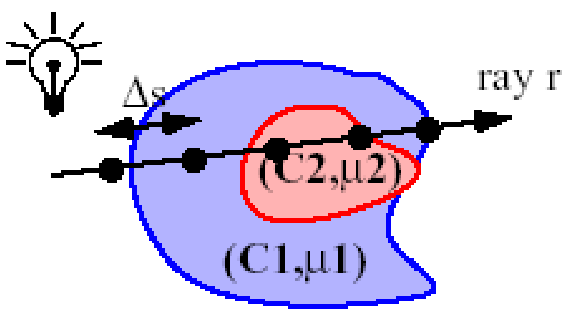
- A rendering ray accumulates attenuated colors
- We write the continuous volume rendering integral:

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

(this is generally not solvable analytically)

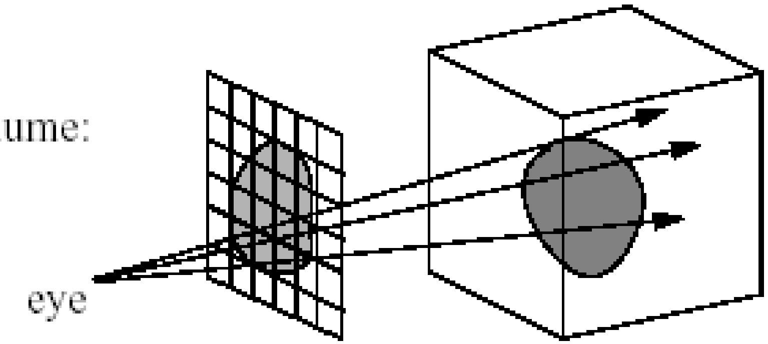
- We can approximate it by discretizing it into sampling intervals of width Δs:

$$I_\lambda(x, r) = \sum_{i=0}^{L/\Delta s} C_\lambda(i\Delta s)\mu(i\Delta s)\Delta s \cdot \prod_{j=0}^{i-1} e^{(-\mu(j\Delta s)\Delta s)}$$



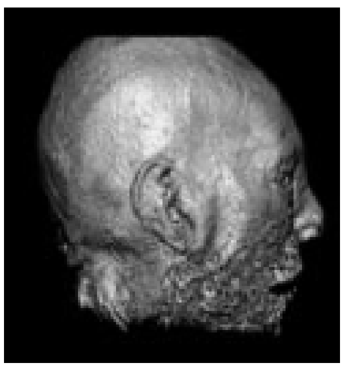
Volume Rendering Modes

- For each pixel in the image, a ray is cast into the volume:

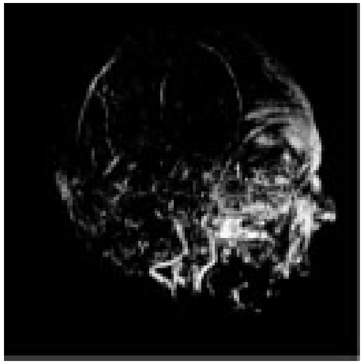


- Four main volume rendering modes exist:

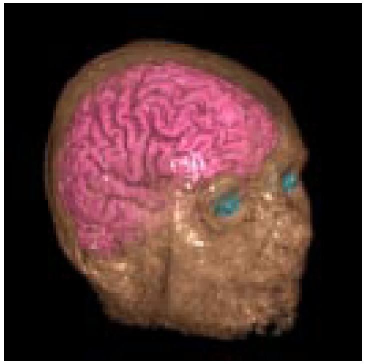
X-ray:
rays sum volume contributions along their linear paths



Iso-surface:
rays look for the object surfaces, defined by a certain volume value



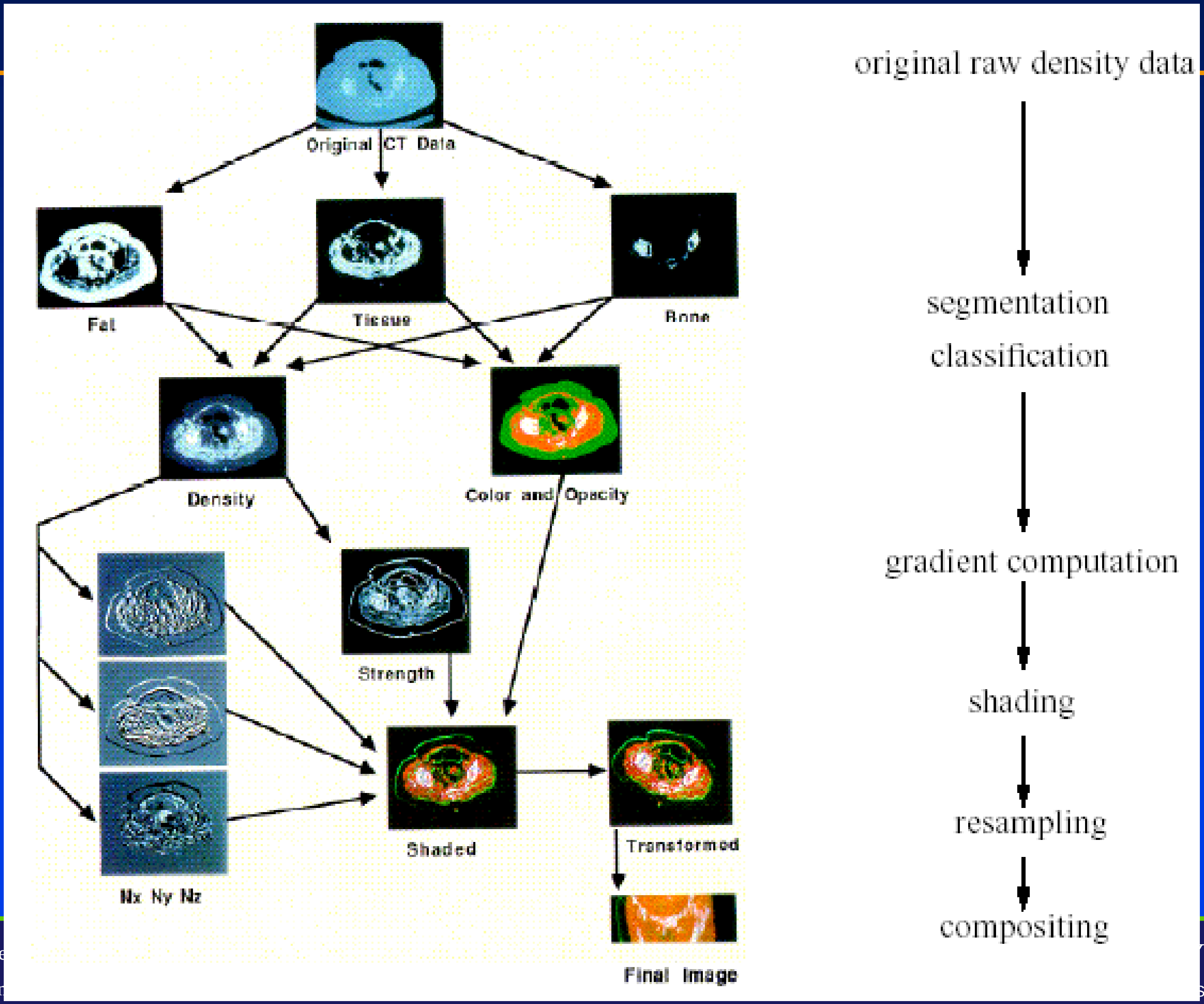
Maximum Intensity Projection (MIP):
a pixel value stores the largest volume value along its ray



Full volume rendering:
rays *composite* volume contributions along their linear paths

Direct Volume Rendering

Volume Rendering Pipeline

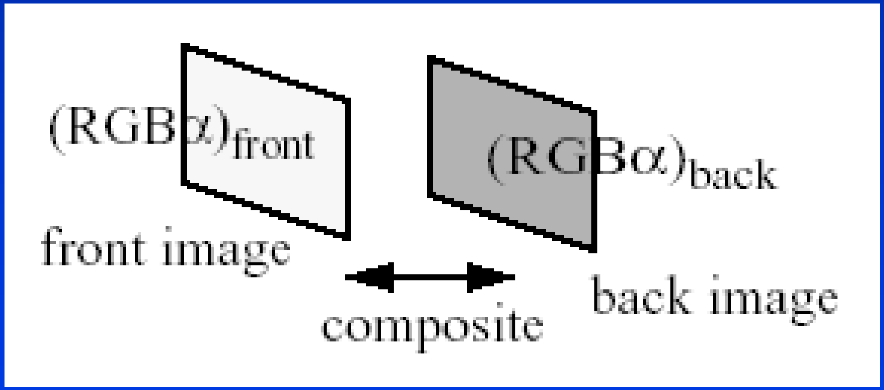


Full Volume Rendering

Ray-casting

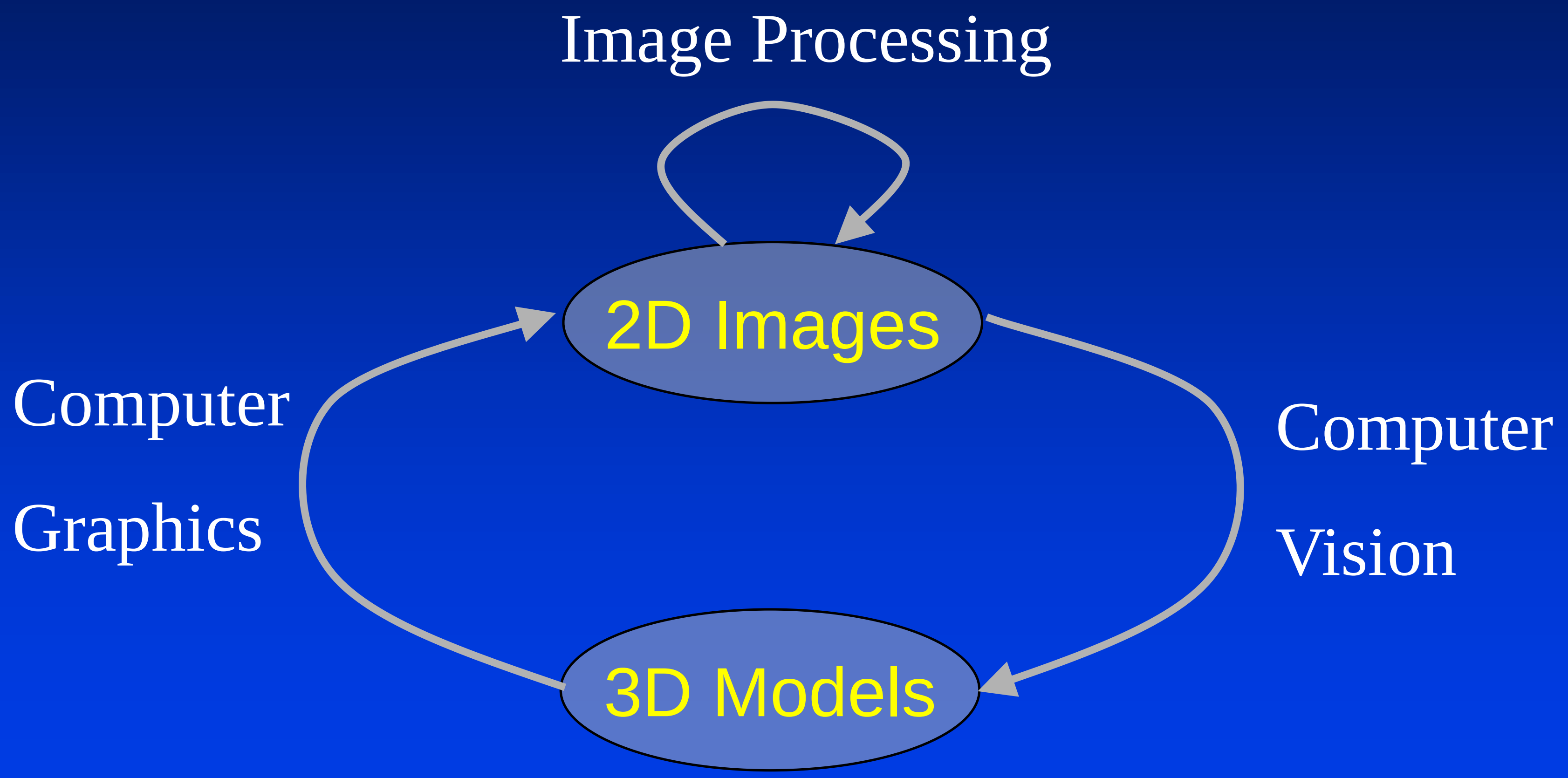
- It is the accumulation of colors weighted by opacities
- Colors and opacities of back pixels are attenuated by opacities of front pixels:

$$rgb = RGB_{back} \cdot \alpha_{back} (1 - \alpha_{front}) + RGB_{front} \cdot \alpha_{front}$$



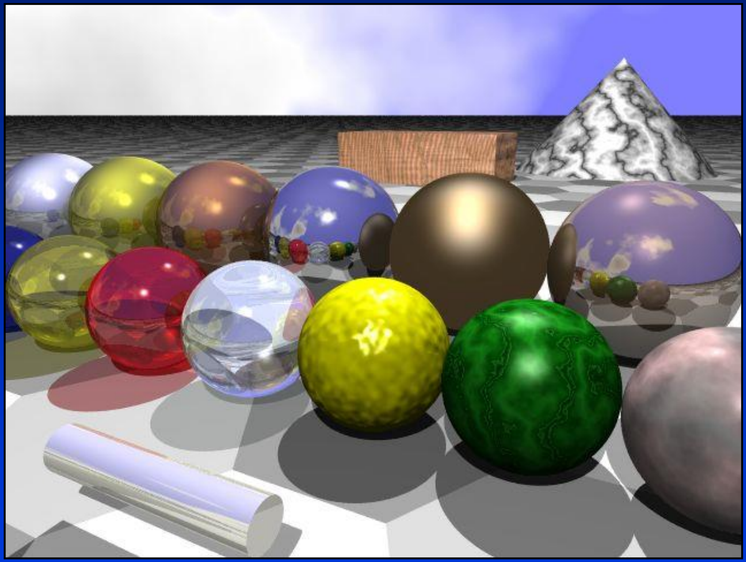
- Volume rendering uses this recursive expression to combine (=composite) the samples taken along the ray

A Classical Classification



Traditional Computer Graphics

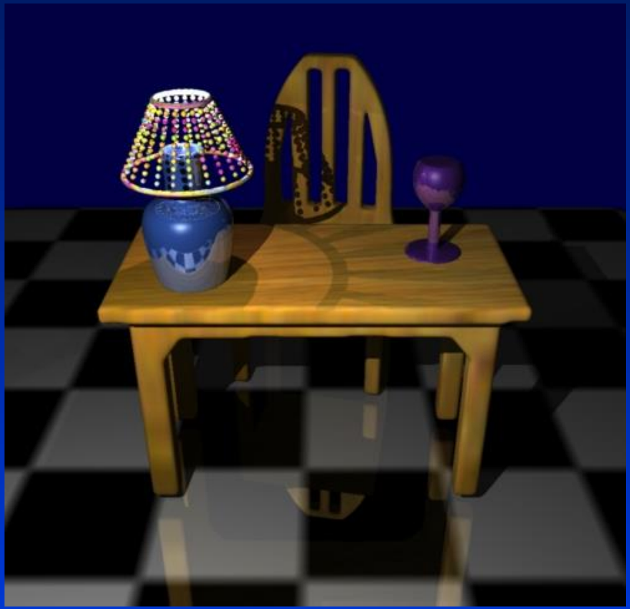
- Computer graphics deals primarily with *surface representations and rendering*
- Objects are defined by a surface or boundary representation
- Explicit distinction between the inside and the outside
- But the inside in empty – it has no substance



- A surface is infinitesimally thin – zero thickness
- This is just an approximation of reality – even a sheet of paper or a human hair has a thickness, however small
- Volume graphics includes a set of techniques for rendering and visualizing *volumetric data*, data that have interior information

Volume Graphics

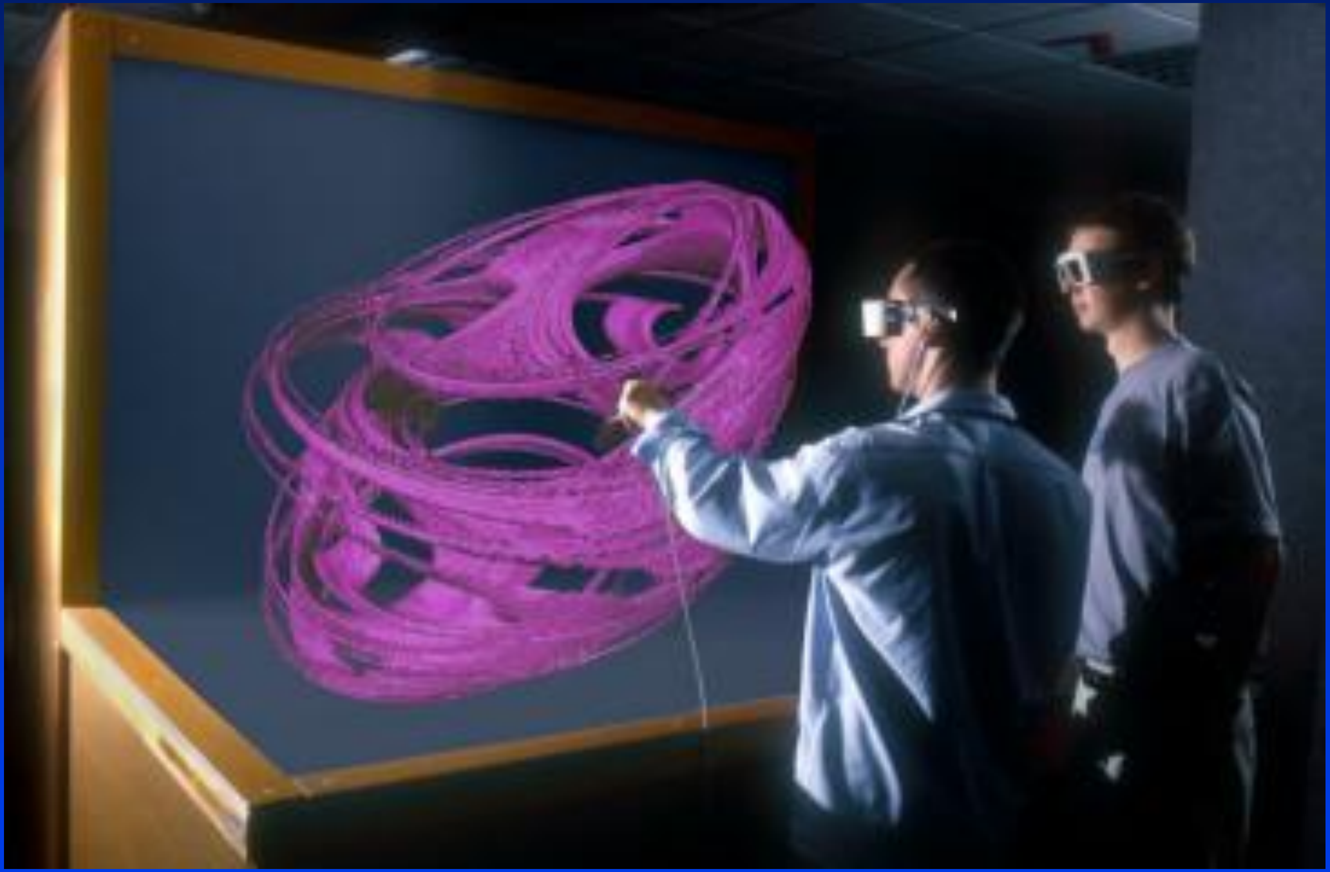
- Not exactly the same thing as volume visualization
- Properly, a sub-area of volume visualization
- Idea: exploit advantages of volumetric techniques in traditional computer graphics applications
  - CAD (computer-aided design)
  - flight simulation
  - virtual sculpting/design



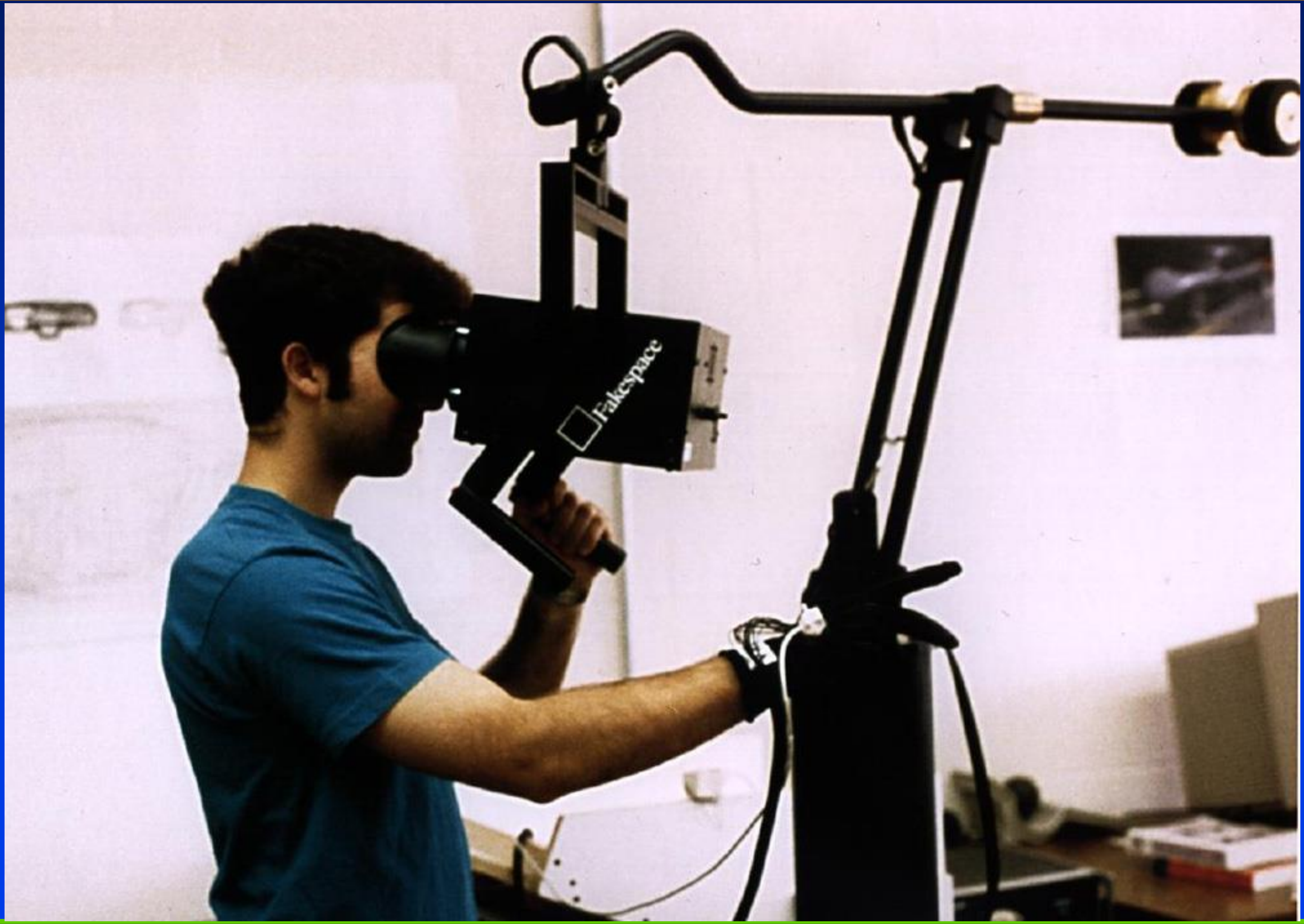
- A critical issue is, however, when we should use volume graphics instead of traditional graphics
- Discussion question: Can you think of some reasons why we might wish to use a volumetric representation in CAD? Or, what would a volumetric representation give us that a surface representation could not?

Graphics Hardware

Virtual Reality Systems

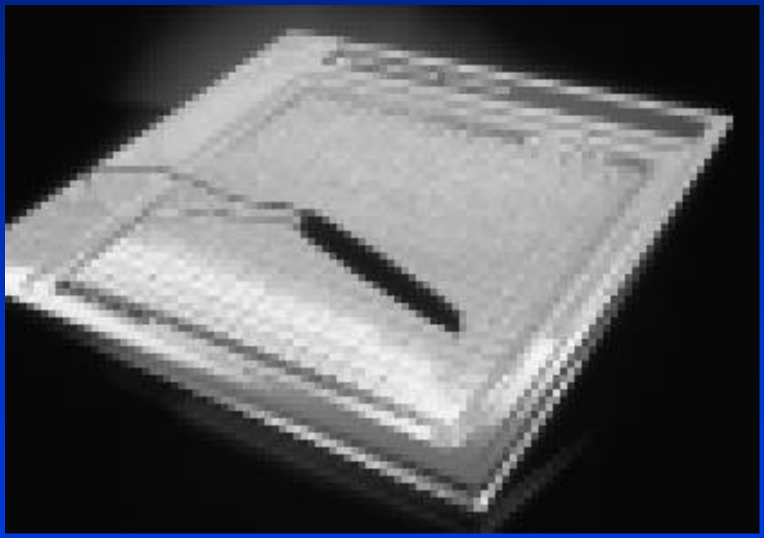


Virtual Reality Systems

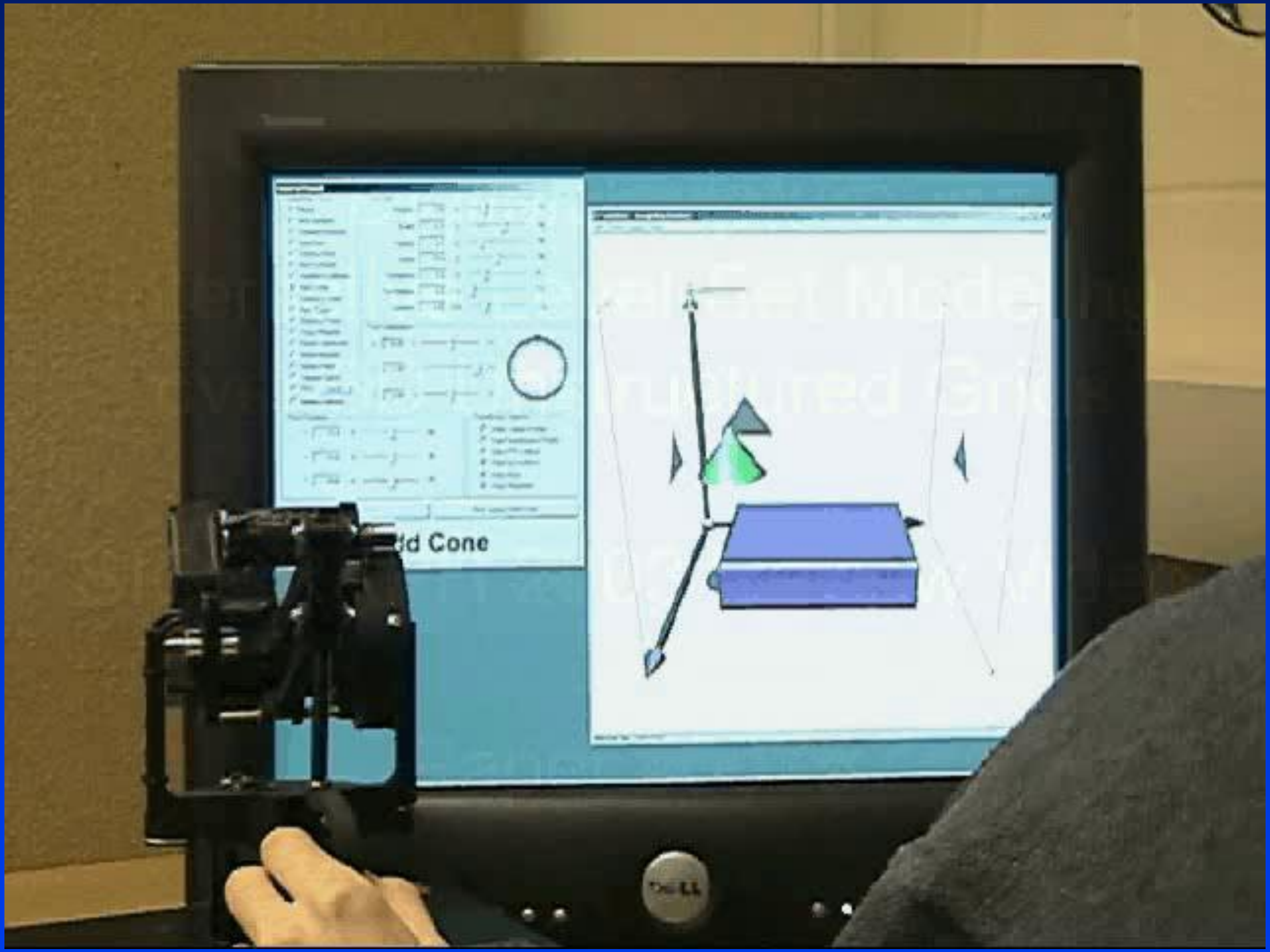


Virtual Reality Systems

Trackball, Joystick, Touch Pad

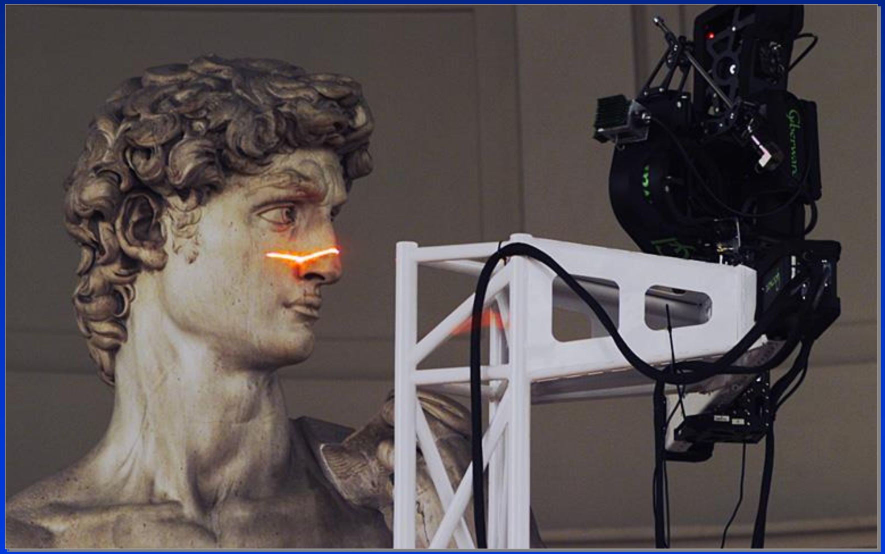


Haptics Device (Phantom 1.0)



3D Laser Range Scanner

3D Laser Range Scanner



3D Camera

Digital Fringe Projector

OpenGL

- Most widely used 3D graphics Application Program Interface (API).
- Truly open, independent of system platforms.
- Reliable, easy to use and well-documented.
- Default language is C/C++.

OpenGL

- The **GL** library is the core OpenGL system:
  - – modeling, viewing, lighting, clipping
- The **GLU** library (GL Utility) simplifies common tasks:
  - – creation of common objects (e.g. spheres, quadrics)
  - – specification of standard views (e.g. perspective, orthographic)
- The **GLUT** library (GL Utility Toolkit) provides the interface with the window system.
  - – window management, menus, mouse interaction

OpenGL

- To create a red polygon with 4 vertices:

```
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_POLYGON);
    glVertex3f(0.0, 0.0, 3.0);
    glVertex3f(1.0, 0.0, 3.0);
    glVertex3f(1.0, 1.0, 3.0);
    glVertex3f(0.0, 1.0, 3.0);
glEnd();
```

- `glBegin` defines a geometric primitive:

  `GL_POINTS`, `GL_LINES`, `GL_LINE_LOOP`, `GL_TRIANGLES`, `GL_QUADS`, `GL_POLYGON`…

- All vertices are 3D and defined using `glVertex`

FLTK

- Fast Light Tool Kit (FLTK)
- www.fltk.org
- C++ oriented
  - A set of UI classes such as Window, box, etc.
- Can mix use with GLUT
- FLUID: fast light UI Designer
  - Fast creation of GUI
  - Automatically writes parts of GUI code from a graphical spec
  - Good for elaborate interfaces

Comments on Programming

- OpenGL, VTK, plus Glui
  - Simple, easy to program, limitations
- OpenGL, VTK, plus FLTK
  - Cross platform, more powerful
- OpenGL, VTK, plus Visual C++
  - Super!
  - Only run under windows system