UNIK-4660/TMA-4235, Summary day 1-2

Introduction and Basic computer graphics

Øyvind Andreassen and Anders Helgeland

oya@ffi.no and ahe@ffi.no

Forsvarets Forskningsinstitutt

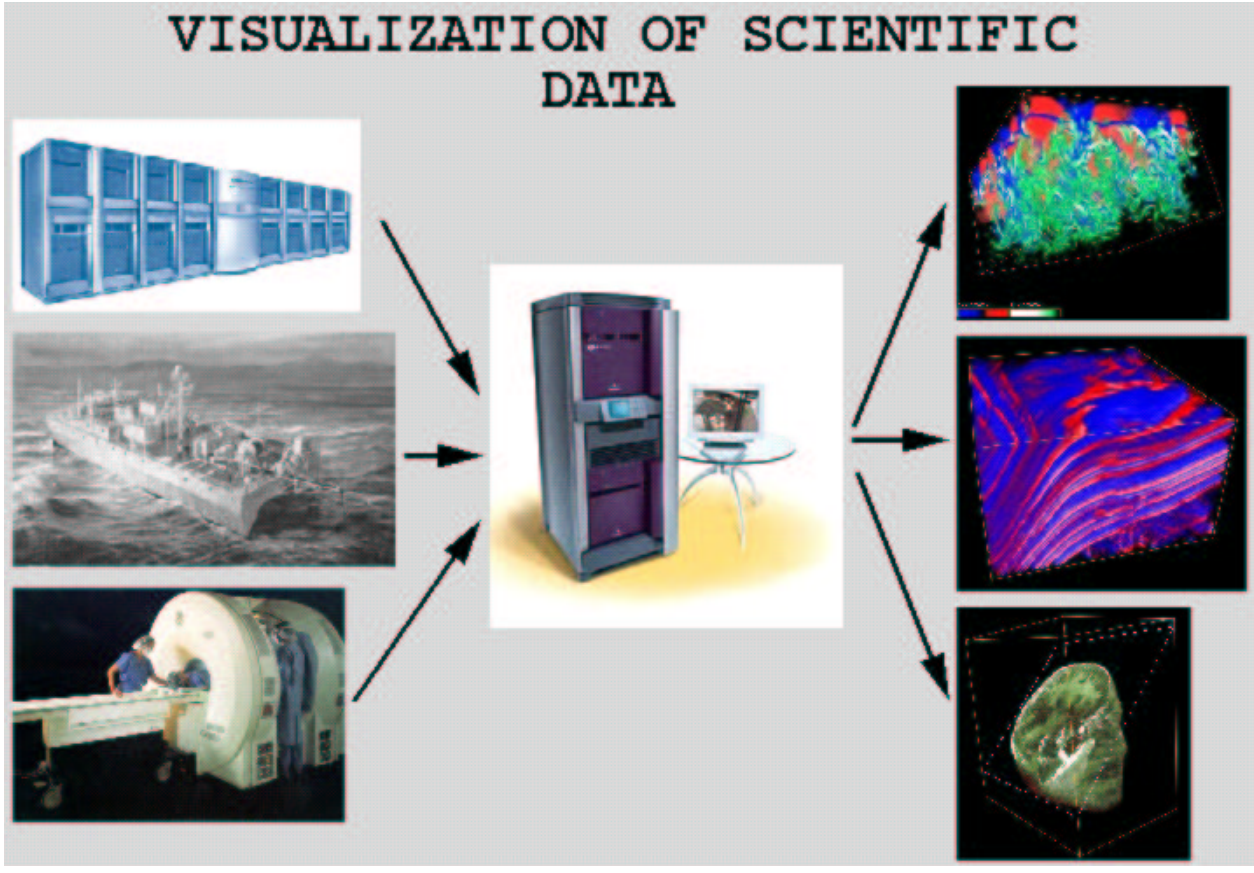
Why visualize?

Our eyes and brain have an amazing ability to identify geometrical patterns and to efficiently extract key visual information. This ability makes visualization to a powerful tool.

- Increasing computing capabilities/improved sensor performance $\Rightarrow$
- Generation of huge data sets $\Rightarrow$
- A challenge to extract key information from the data sets

The primary goal of scientific visualization is to communicate relevant physical information contained in a given dataset.

A typical scenario



Situations where visualization is commonly used.

How did we do it in the past?

The capacity and power of computers have increased dramatically during the last years.

- Until 30 years ago, simulations were mostly one-dimensional
  - Output were printed on line printers or plotted on graph paper
- Until 15 years ago, simulations were mostly up to two-dimensional
  - Visual output were given as contour plots or as color coded images on CTRs

How did we do it recently?

- 15 years ago, a turning point came with the Cray X-MP/Cray Y-MP computer systems
- Low resolution 3D simulations $(\sim 50^3)$ became now feasible
- “Plotting” of 3D data became requested and the field of data visualization grew out of this need
- The need of showing the temporal evolution of 3D data became urgent
- Some degree of interactivity was requested
- Dedicated graphics hardware and software (OpenGL from SGI) designed by the simulator/entertainment industry became utilized for data visualization

How we are doing it today?

- Tflops computer systems are currently available
  - They have 1-Tbytes of memory and 500-Tbytes mass stores
  - Presently the largest Navier-Stokes DNS simulations ever done are for Reynolds number $R_e = UL/\nu \sim 10^4$
- We have still far to go before "useful" work can be done
  - For a cruising B747, $R_e \sim 10^8$
  - The Number of grid points required for this case is at least $N \sim R_e^{9/4} \quad \Rightarrow$
  - There is a need of computer systems 10^9 times larger than current supercomputers

Do we need visualization systems?

Current supercomputers generate a vast amount of data.

Visualization systems must balance computing systems.

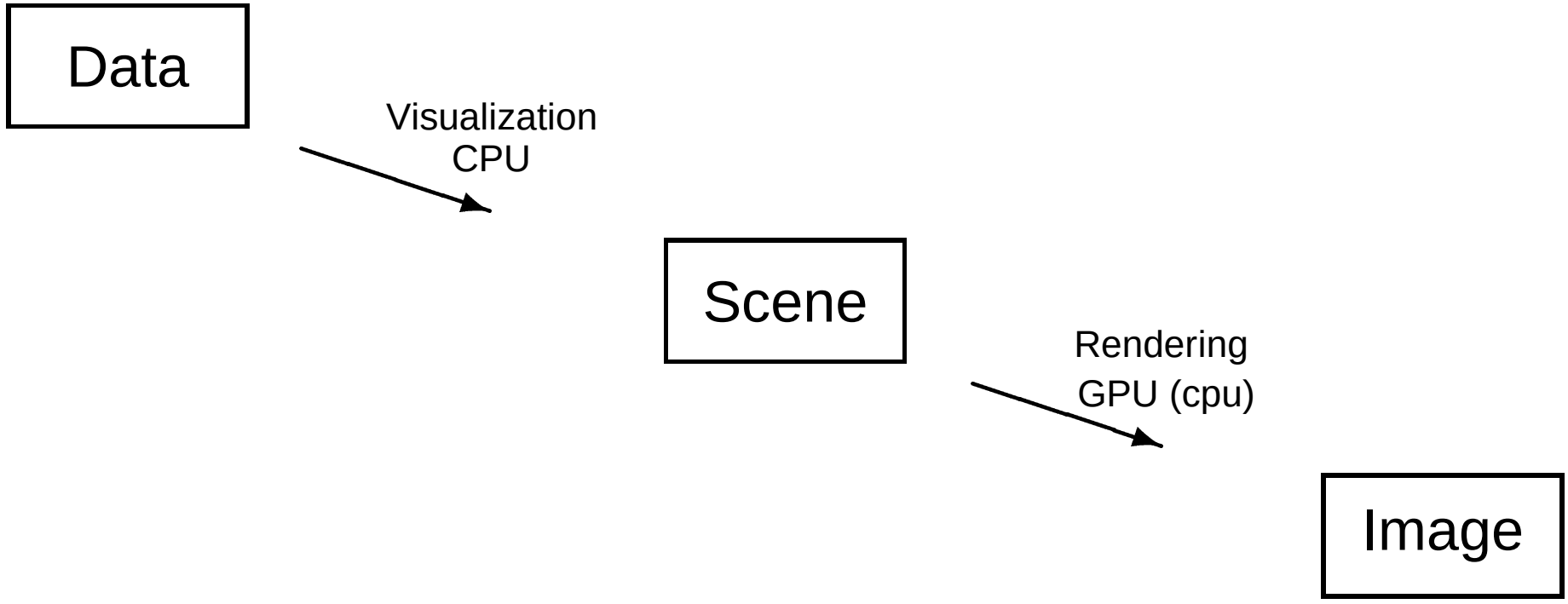
What is needed in visualization

In the visualization process, numerical data are “turned” into a suitable visual form. The process is multi disciplinary and involves among others

- “Physics” - system knowledge
- Numerical mathematics
- Computer graphics software
- Dedicated computer graphics hardware

High degree of interactivity is a requirement for best utilization of our short-time visual memory.

The visualization process



Some components of visualization

Visualization pipeline

- Computation of derived fields, vector magnitudes etc
- Thresholding by data value, clipping in space, and many others
- Color assignment
- Scalar/vector/tensor visualization techniques

Rendering pipeline

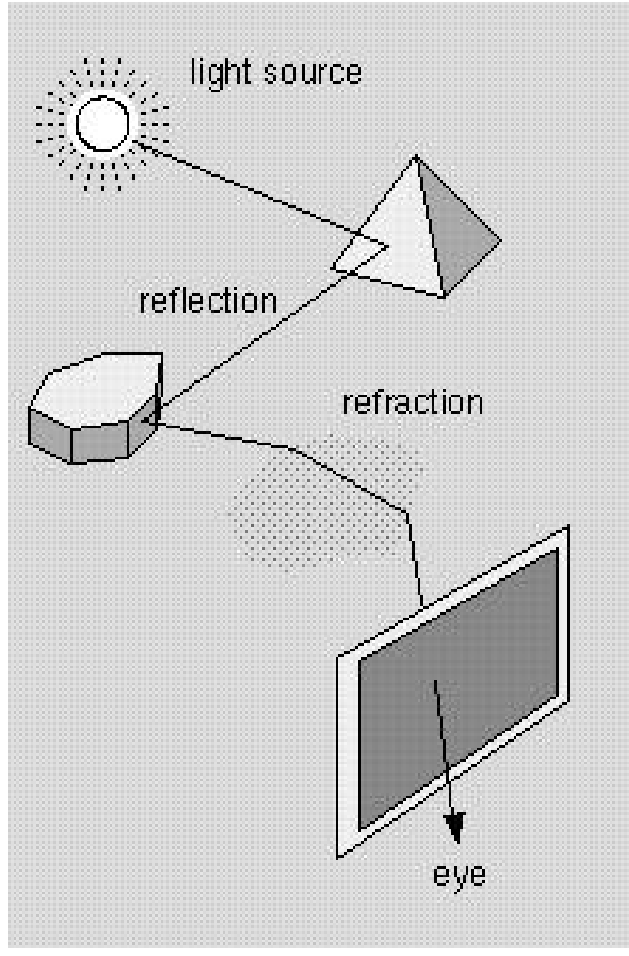
- Vertex transformations
- Lighting
- Rasterization
- Depth sorting (Z-buffering), antialiazing
- Texture mapping

Rendering

Render means represent or portray. In computer graphics, a render is the program that makes the scene objects visible as an image. Common render techniques are:

- Ray-tracing or ray casting
- Radiosity
- Texture based direct rendering

Ray-tracing



Ray-tracing

Ray-tracing

The rays are computed from an eye-point, through each pixel on the screen, refracted/reflected around until a light source or back plane of the scene is hit.

- It gives a high degree of realism
- It is unfortunately difficult to implement in hardware
- It is presently not suited for interactive rendering

Radiosity

For Radiosity, the following equations are solved for each ray composing the image

- $A_i B_i = A_i E_i + R_i \Sigma_{j=1}^{n} B_j F_{ij} A_j$
- Here B_i is the energy per time of ray i passing through area A_i. R_i is a diffuse reflection and scattering cross-section. F_{ij} is the fraction of energy in ray j that is contributing to ray i. B_j is the incoming energy per time through area A_j prior to scattering
- Radiosity gives a high degree of realism
- It is less compute-intensive compared to ray-tracing, but as ray-tracing it is difficult to implement in hardware
- Without hardware support, it is not suitable for interactive rendering

Texture based direct rendering

Texture based direct rendering is the simplest of the render techniques discussed here.

- It is supported in hardware
- Direct rendering is carried out back to front or front to back
- It is suited for volume rendering
- Its quality is less good compared to ray-tracing/radiosity rendering but fully adequate for data visualization
- Texture based direct rendering is used for what we call volume rendering or voxel rendering (voxel is the 3D equivalent of a pixel)

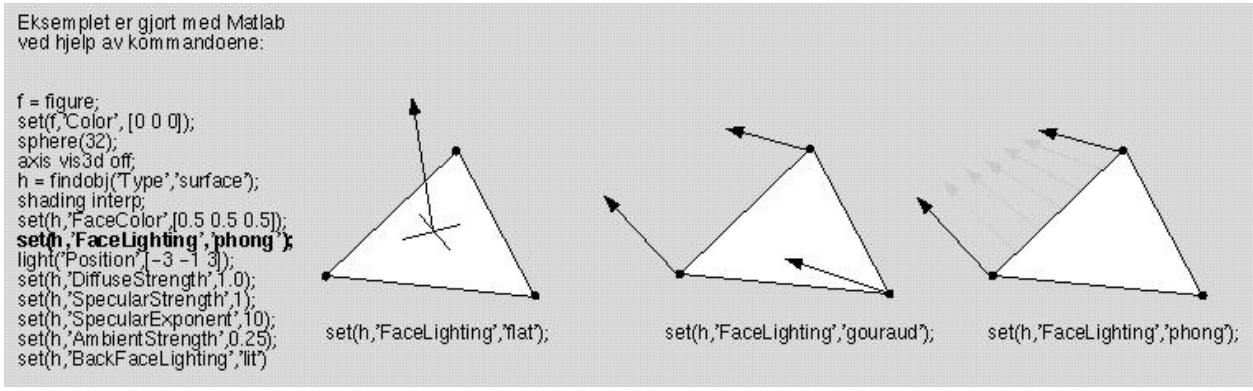
Rasterization

Geometric transformations (rotation, translation and scaling) projects the 3D scene into a 2D image (mapped to screen). The image is composed of polygons in 2D which vertices has a coordinate (x, y) and a color/opacity (R, B, G, α). In rasterization, the polygons are converted into a raster image (pixels). This process is made up of two parts:

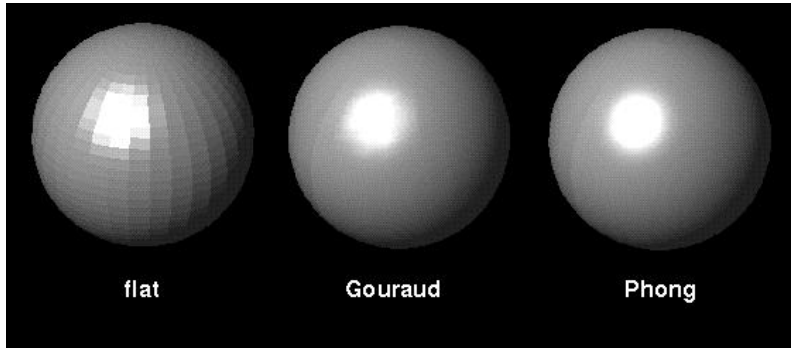
- Computation of color and opacity
- Computation of reflection of light
  - Three reflection models are used (flat, Goraud and Phong)
  - Goraud is implemented in hardware. Some of the platforms support Phong shading

Shading

The figures below show the different shading techniques and the results when applied to a sphere.



Concepts for flat, Goraud and Phong shading



Flat, Goraud and Phong shading of a sphere

Colors

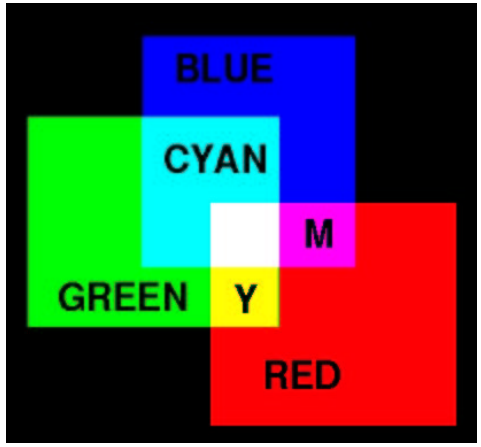
Several color models are available in computer graphics. In visualization the most important are:

1. RGB (red, green, blue) which is an additive color model
2. CMY (cyan, magenta, yellow) is a subtractive model
3. HSV (hue, saturation, value) is most suitable for visualization

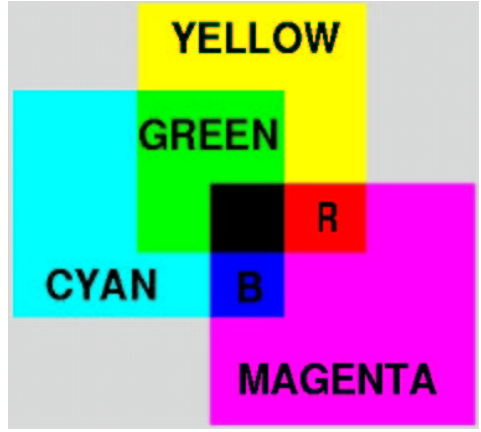
R-C, G-M and B-Y are complementary colors. They are related as

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

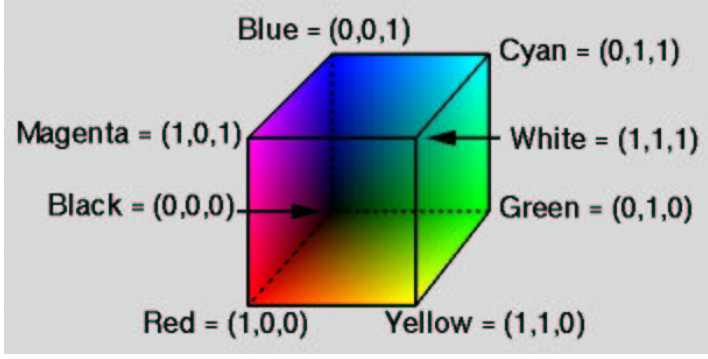
The RGB, CMY and HSV models



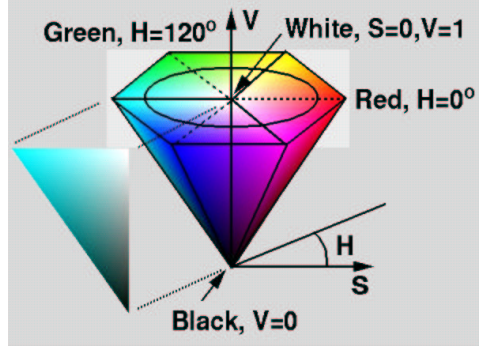
The RGB model is additive



The CMY model is subtractive



The RGB cube



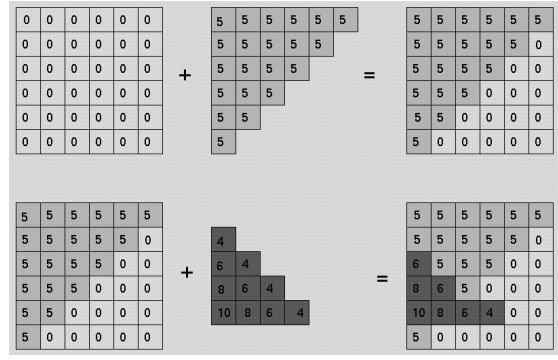
The HSV cone

Computer graphics terms

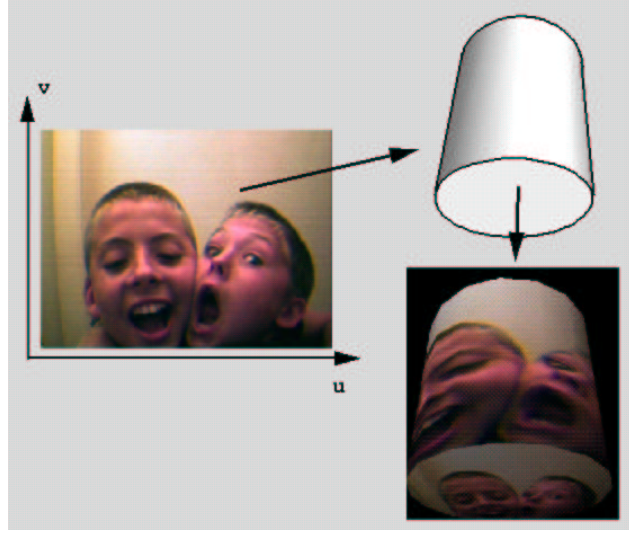
- Color, shading, blending, lighting
- Transparency t and opacity α are complementary quantities
  - $t = \alpha - 1$
- Hidden surface removal / Z-buffer
- Anti aliasing
- Frame buffer, multi sample anti-aliasing
- Texture/image mapping

Technical terms, figures

Antialiazing



Z-buffer

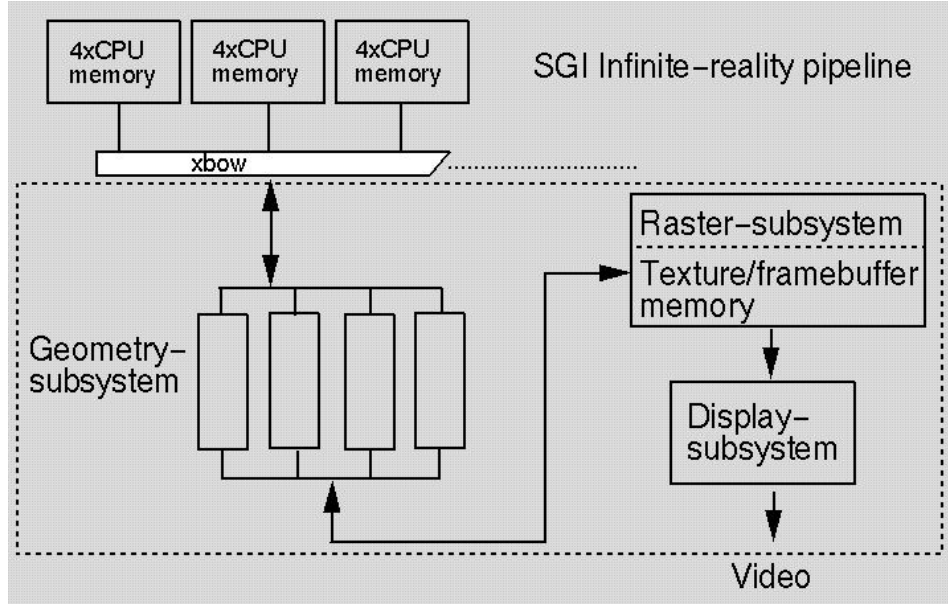


2D Texture mapping

Graphics hardware

Computer graphics hardware is designed as a pipeline with three functional units:

- Geometry engine
- Raster manager
- Display subsystem



Graphics hardware

Graphics hardware, geometry engine

The geometry subsystem has the following functionality:

- Vertex transformations
- Lighting
- Projection to screen
- Convolutions
- Histograms
- Scaling
- Lookup tables

Graphics hardware, raster manager

The raster subsystem has the following functionality:

- Converts triangles, lines and points into pixels
- Z-buffering
- Multi sample anti-aliasing
- Texture mapping
- 3D textures for volume rendering
- Tri-linear interpolation
- HDW supported cut planes
- Various image-processing functions

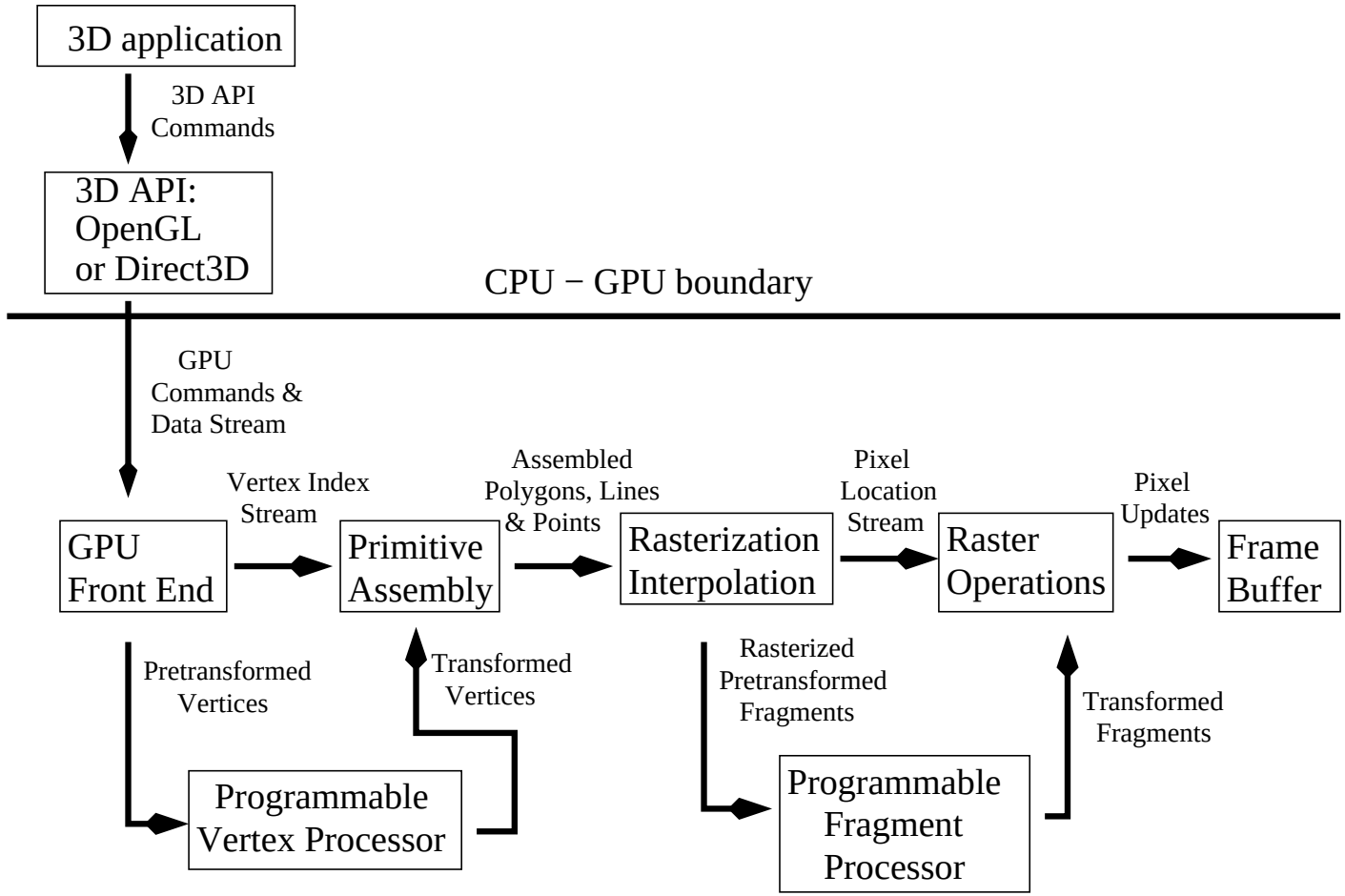
Graphics hardware, display subsystem

The display subsystem has the following functionality:

- Converting output from the raster subsystem to a variety of video formats
- Digital/analog
- NTSC/PAL
- VGA(640x480) to 1920x1200
- Support multiple output devices
- Multiple pipes can feed a single screen

Graphics hardware, extension

In modern GPUs, the vertex and fragment processors can be programmed by C syntax languages. The pipeline can be as follows:



OpenGL, driver for graphics hardware

OpenGL is a “low” level graphics software library. Some important functions are:

- Geometry and raster primitives
- RGBA or color index modes
- Display list or instant mode control
- Transformations
- Lighting and shading
- Z-buffer control
- Anti-aliasing
- Texture mapping
- Feedback and selection
- Stencil planes

Texture mapping/volume visualization

Introduction of 3D texture hardware implied volume-rendering speedup of a factor of 10^5 compared to CPU rendering. Presently (2006), rendering on inexpensive GPUs including 3D texture hardware offers great advantage compared to CPU rendering, Some numerical and signal processing algorithms fit to the GPU hardware and are currently implemented. Very high performance with up to 20 times the performance of the fastest CPUs is achieved.

- Interactive rendering (rot, pan, zoom) and interactive cut planes
- Color tables are attached to the raster hardware
- Visualization of four fields can be done in the same spacial domain
- Can “blend” geometrical objects and voxels in the same scene
- User defined fragment programs can be used on texture data to achieve a flexible and controlled blending of various datasets