Unit 6: Graphical Standards

Vector and raster displays are two substantially different hardware technologies fro creating images on the screen

Raster display now dominant hardware technology because they support several modern applications like:

a. fill area with uniform colors repeated patterns in two or more colors vector can only simulate filled areas with closely spaced sequences of parallel vectors
b. raster display stores images in pixel forms and can read or write moved copies or any manipulations

Primary goal of standardized graphics software is portability

6.1 Need for Machine Independent Graphical Languages

When packages are designed with standard graphics functions software can be moved easily from one hardware to another without standards, program designed for one hardware often cant be transferred to another system with out extensive rewriting of the programs.

In international and national standards organizations in many countries have cooperated in an effort to develop a generally accepted standards for computer graphics

"Graphical Kernel System(GKS)" was adopted as the first graphics software standard by the international standard organization and others like ANSI

Original GKS was 2D and later on 3D extension was developed

Second standard "Programmers Hierarchical interactive graphics standard (PHIGS) is an extension of GKS with increased capabilities for object modeling, color specifications surface rendering and picture manipulations.

An extension of PHIGS called PHIGS+ was developed to provide 3D surface shading capabilities standard graphics functions are defined as a set of specifications that is independent of any programming language

6.3 Graphics Software Standards and Language Binding

Gives the syntax for accessing the various standard graphics functions from this language

e.g. PHIGS, GKS function for specifying a sequence of n-1 connected 2D straight line segment polyline (n,x,y)

- FORTRAN polyline can be drawn with GPLCALLGPR(n,x,y) where x, y are end points
- C gpolyline(n,pts) where pts is list of end points

6.2 Graphical Standards: PHIGS, GKS

GKS (the Graphical Kernel System) is an ANSI and ISO standard.

GKS **standardizes two-dimensional graphics functionality** at a relatively low level.

The main objective of the Graphical Kernel System, GKS, is the **production and manipulation of pictures** in a computer or graphical **device independent way**.

The primary purposes of the standard are:

To **provide for portability** of graphics application programs.
To **aid in the understanding of graphics methods** by application programmers.
To **provide guidelines for manufacturers** in describing useful graphics capabilities.

It consists of **three basic parts**:

1. An **informal exposition** of the contents of the standard which includes such things as how text is positioned, how polygonal areas are to be filled, and so forth.

2. A **formalization of the expository material** by way of abstracting the ideas into discrete functional descriptions. These functional descriptions **contain such information as descriptions of input and output parameters, precise descriptions of the effect each function should have, references into the expository material, and a description of error conditions**. The functional descriptions in this section are language independent.

3. **Language bindings**. These bindings are an implementation of these abstract functions in a specific computer language such as Fortran or Ada or C.

In GKS, pictures are considered to be constructed from a number of **basic building blocks**. These basic building blocks are of a number of types each of which can be used to describe a different component of a picture.
The five main primitives in GKS are:
Polyline: which draws a sequence of connected line segments.
Polymarker: which marks a sequence of points with the same symbol.
Fill area: which displays a specified area.
Text: which draws a string of characters.
Cell array: which displays an image composed of a variety of colours or grey scales.

Associated with each primitive is a **set of parameters which is used to define particular instances** of that primitive.
For example, the parameters of the **Text** primitive are the **string or characters to be drawn** and the **starting position of that string**. Thus:
TEXT(X, Y, 'ABC') will draw the characters ABC at the position (X, Y).
Although the parameters enable the form of the primitives to be specified, additional data are necessary to describe the actual appearance (or aspects) of the primitives.

For example, GKS needs to know the **height of a character string** and **the angle** at which it is to be drawn. These additional data are known as **attributes**.

GKS Output Primitives

GKS standardizes a reasonably complete set of functions for displaying 2D images. It contains functions for drawing lines, markers, filled areas, text, and a function for representing raster like images in a device-independent manner.

In addition to these basic functions, GKS contains many **functions** for **changing the appearance** of the output primitives, such as changing **colors**, changing line **thicknesses**, changing **marker types** and **sizes** etc. The functions for changing the appearance of the fundamental drawing functions (output primitives) are called **attribute setting functions**.

Polylines

The GKS function for **drawing line segments** is called Polyline. The polyline function **takes an array of X-Y coordinates** and draws line segments connecting them. The attributes that control the appearance of a polyline are:
Linetype, which controls whether the polyline is drawn as a **solid, dashed, dotted,** or **dash-dotted line**.
Linewidth scale factor, which controls **how thick** the line is.
Polyline color index, which controls **what color** the line is.

The main line drawing primitive of GKS is the polyline which is generated by calling the function:
POLYLINE(N, XPTS, YPTS)
where XPTS and YPTS are arrays giving the N points (XPTS(1), YPTS(1)) to (XPTS(N), YPTS(N)). The polyline generated consists of N - 1 line segments joining adjacent points starting with the first point and ending with the last.

Polymarkers

The GKS polymarker function **allows drawing marker symbols** centered at coordinate points specified.
The attributes that control the appearance of polymarkers are:
Marker, which specifies **one of five standardized symmetric characters** to be used for the marker.
The five characters are **dot**, **plus**, **asterisk**, **circle**, and **cross**.

Marker size scale factor, which controls **how large** each marker is (except for the dot marker).

Polymarker color index, which specifies **what color** the marker is.

GKS provides the primitive polymarker marks a set of points, instead of drawing lines through a set of points.

A polymarker is generated by the function:
POLYMARKER(N, XPTS, YPTS)

where the arguments are the **same as for the Polyline function**, namely XPTS and YPTS are arrays giving the N points (XPTS(1), YPTS(1)) to (XPTS(N), YPTS(N)).
Polymarker **places a centered marker** at each point.
GKS recognizes the common use of markers to identify a set of points in addition to marking single points and so the marker function is a polymarker.
POLYMARKER(l9, XDK, YDK)

Text

The GKS text function **allows drawing a text string** at a specified coordinate position.
The attributes that control the appearance of text are:
Text font and **precision**, which specifies **what text font** should be used for the characters and **how precisely** their representation should adhere to the settings of the other text attributes.

Character expansion factor, which **controls the height-to-width ratio** of each plotted character.

Character spacing, which specifies **how much additional white space** should be inserted between characters in a string.

Text color index, which specifies **what color** the text string should be.

Character height, which specifies **how large** the characters should be.

Character up vector, which specifies at **what angle** the text should be drawn.

Text path, which specifies in what direction the text should be written (right, left, up, or down).

Text alignment, which specifies **vertical and horizontal centering** options for the text string.

A **text string** may be generated by invoking the function:

TEXT(X, Y, STRING)   where (X, Y) is the text position and STRING is a string of characters.

The **character height attribute** determines the height of the characters in the string. Since a character in a font will have a designed aspect ratio, the character height also determines the character width. The character height is set by the function:
SET CHARACTER HEIGHT(H)
where H is the character height.

The **character up vector** is perhaps the most important text attribute. Its main purpose is to determine the orientation of the characters. However, it also sets a reference direction which is used in the determination of text path and text alignment. The character up vector specifies the up direction of the individual characters. It also specifies the orientation of the character string in that. By default, the characters are placed along the line perpendicular to the character up vector. The function:
SET CHARACTER UP VECTOR(X, Y)

Fill Area

The GKS fill area function **allows specifying a polygonal shape of an area to be filled** with various interior styles.
The attributes that control the appearance of fill areas are:
Fill area interior style, which specifies **how the polygonal area should be filled**: with solid colors or various hatch patterns, or with nothing, that is, a line is drawn to connect the points of the polygon, so to get only a border.
Fill area style index. If the fill area style is hatch, this index specifies **which hatch pattern** is to be used: horizontal lines; vertical lines; left slant lines; right slant lines; horizontal and vertical lines; or left slant and right slant lines.
Fill area color index, which specifies **the color of the fill patterns** or solid areas.

At the same time, there are now many devices which have the concept of an area which may be filled in some way. These vary from intelligent pen plotters which can cross-hatch an area to raster displays which can completely fill an area with a single colour or in some cases fill an area by repeating a pattern.

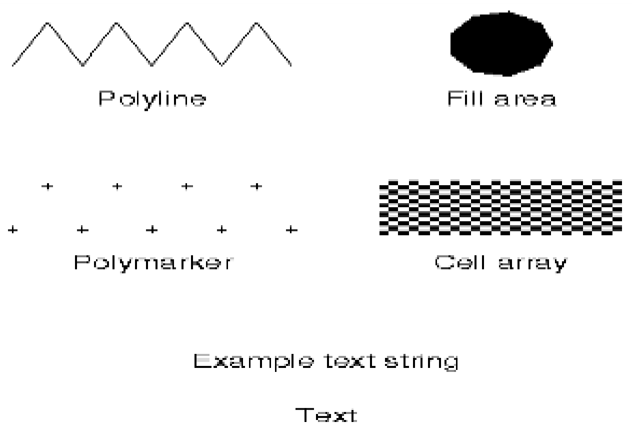
GKS provides a fill area function to satisfy the application needs which can use the varying device capabilities. Defining an area is a fairly simple extension of defining a polyline. An array of points is specified which defines the boundary of the area. If the area is not closed (i.e. the first point is not the same as the last point), the boundary is the polyline defined by the points but extended to join the last point to the first point. A fill area may be generated by invoking the function:

FILL AREA(N, XPTS, YPTS)
where, as usual XPTS and YPTS are arrays giving the N points (XPTS(1), YPTS(1)) to (XPTS(N), YPTS(N)).

Cell Array

The GKS cell array function **displays raster like images** in a device-independent manner.

The cell array function takes the two corner points of a rectangle specified, a number of divisions (M) in the X direction and a number of divisions (N) in the Y direction. It then partitions the rectangle into M x N subrectangles called cells. Assign each cell a color and create the final cell array by coloring each individual cell with its assigned color.



PHIGS

PHIGS is an international **ISO standard of functional interface** between an application program and a configuration of graphical input and output devices.

This interface contains basic functions for dynamic interactive 2D and 3D graphics on wide variety of graphics equipment

1. PHIGS concept and graphical workstation

PHIGS is a high level graphics library **with over 400 functions**.

It allows an application programmer **to describe a model of a scene, to display the model on workstation, to manipulate and to edit the model interactively**

The models are stored in a graphical database known as **centralized structure store** (CSS).

The fundamental entity of data is a **structure element** and these are grouped together into units called **structures**.

Structures are organized as acyclic directed graphics called **structure networks**

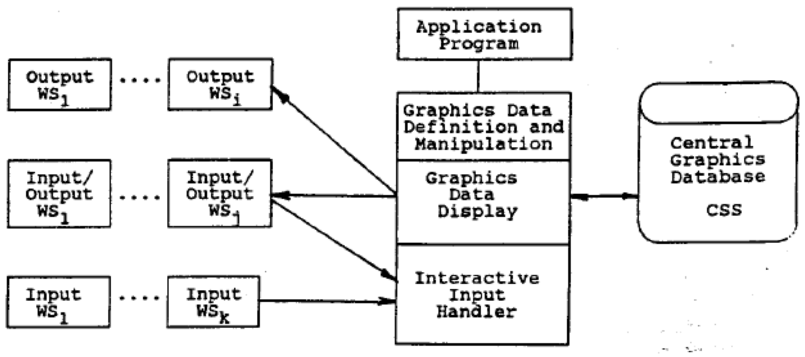


Fig. 1 Structure of the PHIGS

The two abstract concepts (input and output) are the building blocks of an **abstract workstation**.

A **PHIGS workstation** represents a unit consisting of zero or one **display surfaces** and zero or more **input devices** such as keyboard, tablet , mouse and light pen.

The workstation presents this devices to the application program as a configuration of abstract devices thereby **shielding the hardware peculiarities**

Each workstation has a type falling into one of five categories

```
OUTPUT – supports output only

INPUT – supports input only
```

`OUTIN – supports output and input`

`MO – supports output graphical and application data to the external storage`

`MI- supports input graphical and application data from the external storage to the application`

For every type of workstation present in a PHIGS implementation**, there exists a generic workstation description table** which describes the standard capabilities and characteristics of the workstation.

When the workstation is opened, a **new specification workstation description table is created** for that workstation description table obtained from the device itself. And possibly from other implementation dependent source.

The content of the specific workstation description table may change at any time while the workstation is open.

The application program can inquire which generic capabilities are available before the workstation is open.

The specific capabilities may be inquired while the workstation is open by first inquiring the workstation type of an open workstation to obtain the workstation type of the specific workstation description table, and then using this workstation type as a parameter to the inquiry functions which query the workstation description table. This information may be used by the application program to adapt its behavior accordingly

The application program references a workstation by means of a workstation identifier. Connection to a particular workstation is established by the function `OPEN WORKSTATION` which associates the workstation identifier with a generic workstation type and a connection identifier.

The current state of each open workstation . state values of the WSSL maybe set up by the “set functions” and may be inquired by “inquire functions”.

2. Structure Entity

A structure element is the fundamental entity of data. Structure elements are used to represent application specified graphics data for output primitives, attribute selections, modeling transformations and clipping , invocation of other structures, and to represent application data.

The following types of structure elements are defined in PHIGS

- Output primitive structure elements
- Attribute specification structure elements
- Modeling transformation and clipping structure elements

- Control structure element
- Editing structure element
- Generalized structure element
- Application data
- Graphical output

Graphical Output

Picture generated by PHIGS are build up of basic pieces called output primitives. Output primitives are generated from structure elements by structural traversal

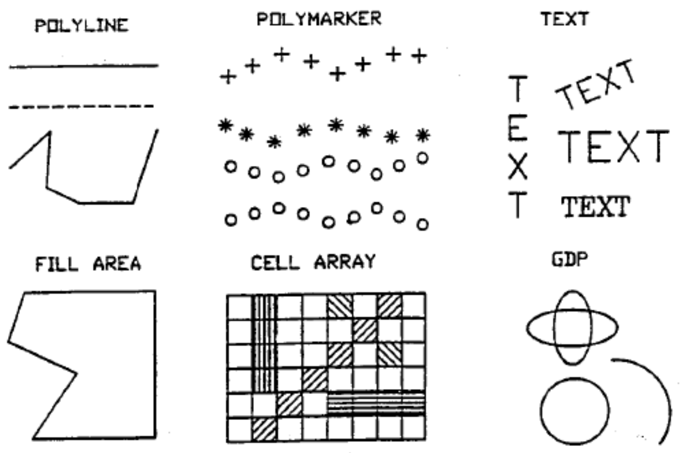


Fig. 2 Examples of output primitives

POLYLINE set of connected lines defined by point sequences POLYLINE3

POLYMARKER set of symbols of one type centered at a given position POLYMARKER 3

TEXT character string at a given position on an arbitrary plane in modeling coordinate space

ANNOTATION TEXT RELATIVE character string at specified position in an x y plane parallel to view plane ANNOTATION TEXT RELATIVE 3

FILL AREA single polygonal area which may be hollow or filled with uniform color, pattern or hatch style

FILL AREA SET a set of polygonal areas which may be filled similar as FILL AREA

CELL ARRAY two dimensional array of cells with individual colo9ur4s

GENERALIZED DRAWING PRIMITIVE special geometrical output capabilities of a workstation such as the drawing of spline curves, circular arcs, elliptic arcs

for individual specifications of aspects, there is a separate attribute for each aspect. These attributes are workstation independent

bundled aspects are selected by a bundle index into a bundle table each entry of which contains non geometric aspects of a primitive. The non geometric aspects are workstation dependent in that each workstation has its own set of bundle tables (stored in the workstation state list) the values in a particular bundle may be different for different workstations

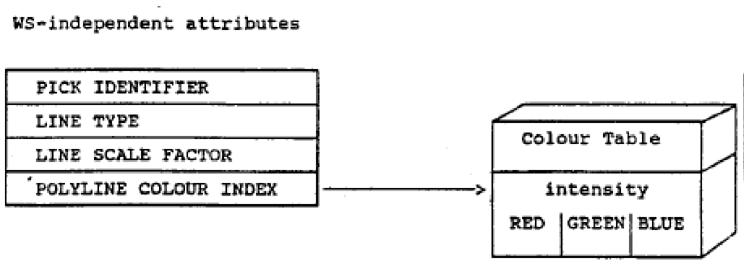


Fig. 3 Individual specification of attributes

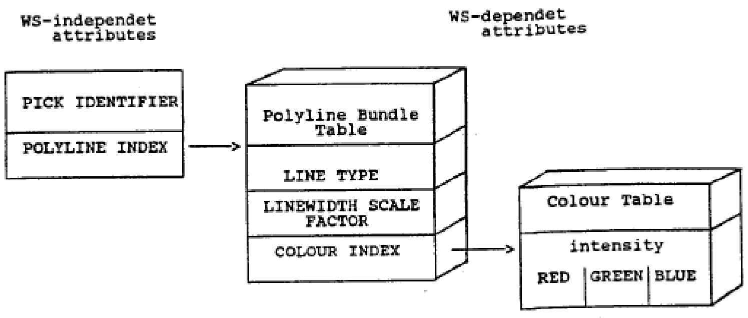


Fig. 4 Bundled specification of attributes

Output primitive attributes

Attributes of the first type control the geometric aspects of primitives. These are aspects that affect the shape or size for the entire primitive (e.g. CHARACTER HEIGHT for TEXT)

Hence they are sometimes referred to as geometric attributes. Geometric attributes are workstation independent and if they represent coordinate data they are expressed in modeling coordinates

Attributes of the second type control the non geometric aspects of primitives these are aspects that affect a primitive's appearance(for example. Line type for POLYLINE, or color index for all primitives except CELL ARRAY) or the shape or size of the component parts of the primitive (for example , marker size scale factor for POLYMARKER)

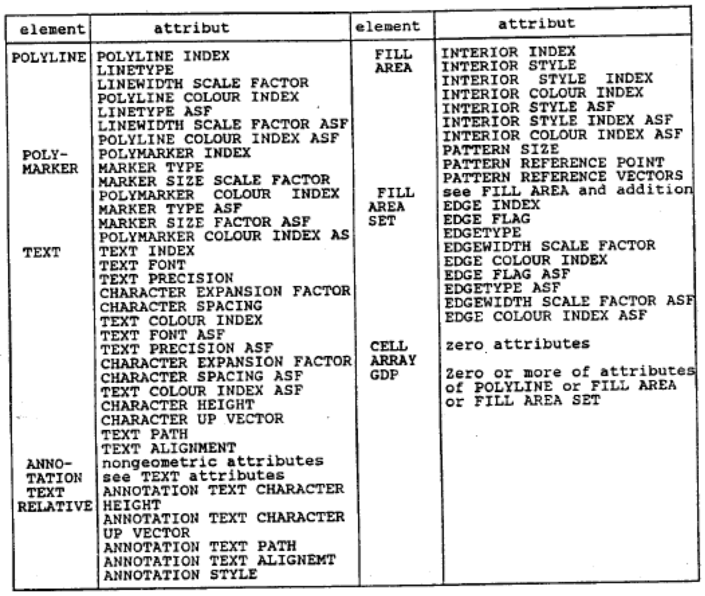
Non geometric aspects do not represent coordinate data

The non geometric aspects of primitive may be specified either via a bundle or individually the geometric aspects only individually

3. Structure and structure networks

PHIGS supports the storage and manipulation of data in CSS the CSS contains graphical and application data organized into units called **structures** which may be related each other hierarchically to form structure networks

Each structure is identified by unique name which is specified by the application



element	attribut	element	attribut
POLYLINE	POLYLINE INDEX LINETYPE LINEWIDTH SCALE FACTOR POLYLINE COLOUR INDEX LINETYPE ASF LINEWIDTH SCALE FACTOR ASF POLYLINE COLOUR INDEX ASF	FILL AREA	INTERIOR INDEX INTERIOR STYLE INTERIOR STYLE INDEX INTERIOR COLOUR INDEX INTERIOR STYLE ASF INTERIOR STYLE INDEX ASF INTERIOR COLOUR INDEX ASF PATTERN SIZE PATTERN REFERENCE POINT PATTERN REFERENCE VECTORS
POLY-MARKER	POLYMARKER INDEX MARKER TYPE MARKER SIZE SCALE FACTOR POLYMARKER COLOUR INDEX MARKER TYPE ASF MARKER SIZE FACTOR ASF POLYMARKER COLOUR INDEX AS	FILL AREA SET	see FILL AREA and addition EDGE INDEX EDGE FLAG EDGETYPE EDGEWIDTH SCALE FACTOR EDGE COLOUR INDEX EDGE FLAG ASF EDGETYPE ASF EDGEWIDTH SCALE FACTOR ASF EDGE COLOUR INDEX ASF
TEXT	TEXT INDEX TEXT FONT TEXT PRECISION CHARACTER EXPANSION FACTOR CHARACTER SPACING TEXT COLOUR INDEX TEXT FONT ASF TEXT PRECISION ASF CHARACTER EXPANSION FACTOR CHARACTER SPACING ASF TEXT COLOUR INDEX ASF CHARACTER HEIGHT CHARACTER UP VECTOR TEXT PATH TEXT ALIGNMENT	CELL ARRAY	zero attributes
ANNO-TATION TEXT RELATIVE	nongeometric attributes see TEXT attributes ANNOTATION TEXT CHARACTER HEIGHT ANNOTATION TEXT CHARACTER UP VECTOR ANNOTATION TEXT PATH ANNOTATION TEXT ALIGNEMT ANNOTATION STYLE	GDP	Zero or more of attributes of POLYLINE or FILL AREA or FILL AREA SET

Structure networks are organized as directed acyclic graphs

So a structure may contain invocations of other structures containing in CSS. The invocation of a structure is achieved using the **execute structure element**. Such an invocation is known as a **structure reference**. Structure can not be referenced recursively

A structure can be created in one of the following ways:

- When a reference to the nonexistent structure is inserted into a structure in the CSS
- When the structure is opened for the first time (function OPEN STRUCTURE)
- When the structure is posted for display on a workstation (POST STRUCTURE)
- When the structure is referenced in any function changing the structure identifier
- When the not existing structure in CSS is retrieved from an archive (RETRIEVE STRUCTURE)
- When the not existing structure is emptied (EMPTY STRUCTURE)

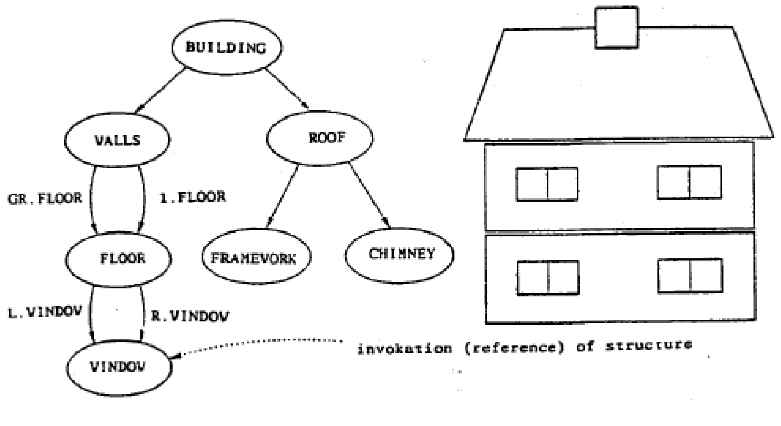


Fig. 5 Building and its structure network

Structure traversal and display

- A structure network is identified for display on a workstation by posting function POST STRUCTURE

A structure may be displayed only if it is a member of a posted structure network

To display a network the structure elements have to be extracted from the CSS and processed . that process is called traversal process

The traversal process interprets each structure element in the structure network sequentially starting at the first element of the top of the network

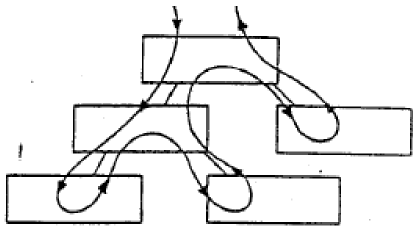


Fig. 6 Traversal process

A traversal state list is associated with each traversal process. Values in this state list ;may be accessed when the structure elements are interpreted

Each time a traversal is initiated the associated list is initialized

Structure editing

Each element within a structure can be accessed and modified individually with editing functions to inset new elements, replace elements with new structure elements, delete structure elements. Navigate within the structure and inquire structure element content

A structure is identified for editing an element pointer is established which points at the lat element in the structure . functions for positioning element pointer

SET ELEMENT POINTER – set to an absolute position

SET OFFSET ELEMENT POINTER – set relative to current position

SET ELEMENT POINT AT LABEL – set a position of the specified label structure element

The edit mode defined by the function SET EDIT MODE defines whether new elements replace the element pointed to by the element pointer or are added after the element pointed to by the element pointer

Functions for editing:

COPY ALL ELEMENTS FROM STRUCTURE- copy all the elements of a structure into the open structure

DELETE ELEMENT – delete element at which the element pointer is pointing

DELETE ELEMENT RANGE- delete a group of elements between two element positions

DELETE ELEMENT LABELS-delete group of elements delimited by labels

EMPTY STRUCTURE-delete all elements of the structure

Manipulation of structures in CSS

Operations for manipulation of structure

DELETE STRUCTURE- delete structure ;and all references to it

DELETE ALL STRUCTURES- delete all structure from the CSS

DELETE STRUCTURE NETWORK-delete the indicated structure and all its ancestors

CHANGE STRUCTURE IDENTIFIER-change identifier specified structure

CHANGE STRUCTURE REFERENCES-change all references of specified structure

ELEMENT SEARCH –search within a single structure for an element of a particular element type

Structure archival and retrieval

OPEN ARCHIVE FILE , CLOSE ARCHIVE FILE-initiates or terminates access to archive file

ARCHIVE ALL STRUCTURES-storing of structure to archive file

RETRIEVE ALL STRUCTURES- recovers structures from an archive file

DELETE STRUCTURES FROM ARCHIVE-deletes structure in an archive file

4. Coordinate systems and transformations

Structures represent parts of a hierarchical model of modeling scene. Each of these parts has own world space represented by modeling coordinate system .

The relative positioning of the separate parts is achieved by having a single World coordinate space onto which all the defined modeling coordinate systems are mapped by a composite modeling transformation during the traversal process

The world coordinate space can be regarded as a workstation independent abstract viewing space

The workstation dependent stage then performs a transformation on the geometrical information contained in output primitives , attributes and logical input values

These transformations perform mapping between four coordinate systems:

- **World coordinates** used to define uniform coordinate system for all abstract workstation
- **View reference coordinate** used to define a view
- **Normalized projection coordinates**; used to facilitate assemblies of different views

- **Device coordinates**: one coordinate system per workstation representing its display space

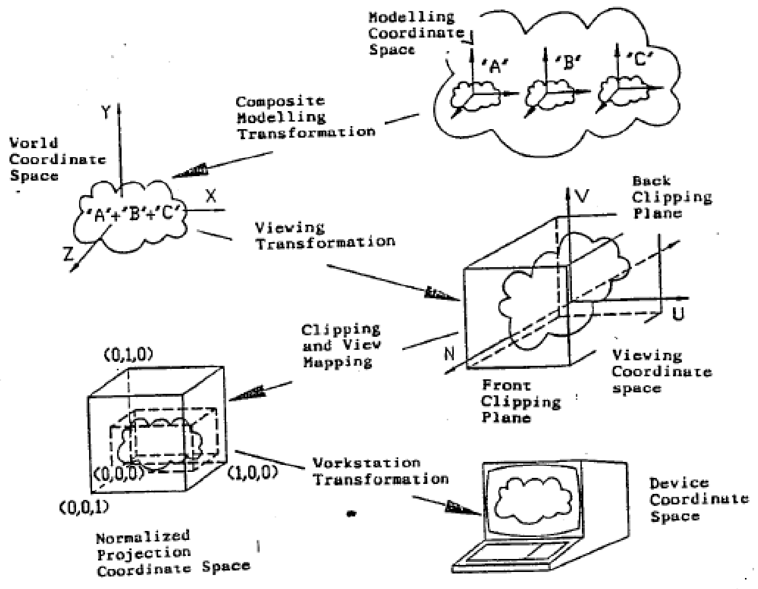


Fig. 7 Coordinate systems and transformations in PHIGS

Output primitives and attributes are mapped from world coordinate to view reference coordinate by the **view orientation transformation**, from view reference coordinates to normalized projection coordinates by the **view mapping transformation** and from normalized projection coordinates to device coordinates by the **workstation transformation**. Hidden lines removals and clippings are also done during the mapping process

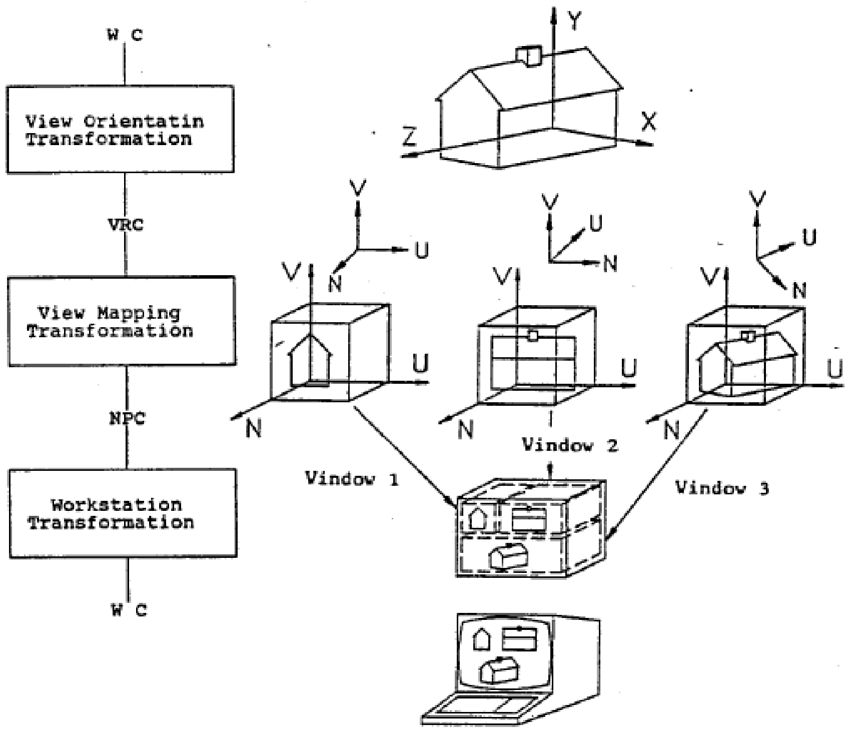

W C
View Orientatin Transformation
VRC
View Mapping Transformation
NPC
Workstation Transformation
W C
Y
Z
X
V
U
N
Vindow 1
Vindow 2
Vindow 3

Fig.8 Viewing transformations in PHIGS

5. Graphical input

An application program gets graphical input from an operator by controlling the activity of one or more logical input devices.

6. Language Interfaces

PHIGS defines only a language independent nucleus of a graphics system

For integration into a language, PHIGS is embedded in a language dependent layer containing the language conventions, e.g. parameter and name assignment

In case of the layer model, each layer may call the functions of the adjoining lower layers. In general the application program uses the application oriented layer, the language dependent layer, other application dependent layers and operations system resources. There are standards of the language dependent layers for the language FORTRAN, PASCAL and C

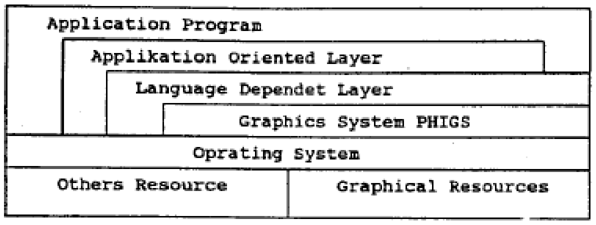


Fig. 9. Layer model of PHIGS

7. Graphics system PHIGS +

PHIGS does not support a shading of pictures and modeling a non uniform rational B Spline curves and surfaces (NURBS)

An extension of PHIGS for these functionalities is PHIGS+

PHIGS+ enables to specify light sources

Parameters of the light sources are described and application program s may set the up

The predefined light sources are ambient, directional, positional, spot light source.

Shading method is an attribute of the graphics primitives.

NONE- No shading

COLOUR-color interpolation shading

NORMAL-normal interpolation shading

DOT PRODUCT- dot product interpolation shading

Light type accepted for shading

NONE – no reflectance calculation performed

AMBIENT-use ambient term

AMB_DIF-use ambient and diffuse terms

AMB_DIF_SPEC- use ambient, diffuse and specular terms

PHIGS+ offers additional output primitives and functions;

COMPUTE FILL AREA SET GEOMETRIC NORMAL-compute geometric normal of the fill area set

FILL AREA SET3 WITH DATA-creates a 3D fill area set structure element that includes color and shading data

QUADRILATERAL MESH3 WITH DATA-creates a 3D quadrilateral mesh primitive with color and shading data

TRIANGLE STRIP3 WITH DATA-creates a 3D triangle strip primitive with color and shading data

NON UNIFORM B-SPLINE CURVE NURBS-crates a structure element containing the definition of a non-uniform B-Spline Curve

Significance of PHIGS

Portability of program

- PHIGS is computer and device independent graphics system. The application program that utilize PHIGS can be easily transported between host processors and graphics devices

Sophisticated capabilities save development time

- PHIGS manages the storage and display of 2D and 3D graphical data, crates and maintains a hierarchical database

Increased program performance because of fewer error conditions

- Application using PHIGS have well defined inputs and outputs that minimizes errors

6.4 Overview of Graphics File Formats

A graphics format file reader is responsible for opening a file, determining its validity and interpreting the information contained within it.

A reader may take its input , source image data either from a file or from the data stream of an input device, such as a scanner or frame buffer.

There are two types of reader designs:
i. Filter ii. Scanner.

A filter reads a data source one character at a time.

The Scanners randomly access across the entire data source. They can read and reread data anywhere within the file.

Filters require small amount of memory while the Scanners require a larger amount of memory.
Image data are read from a Graphics File.

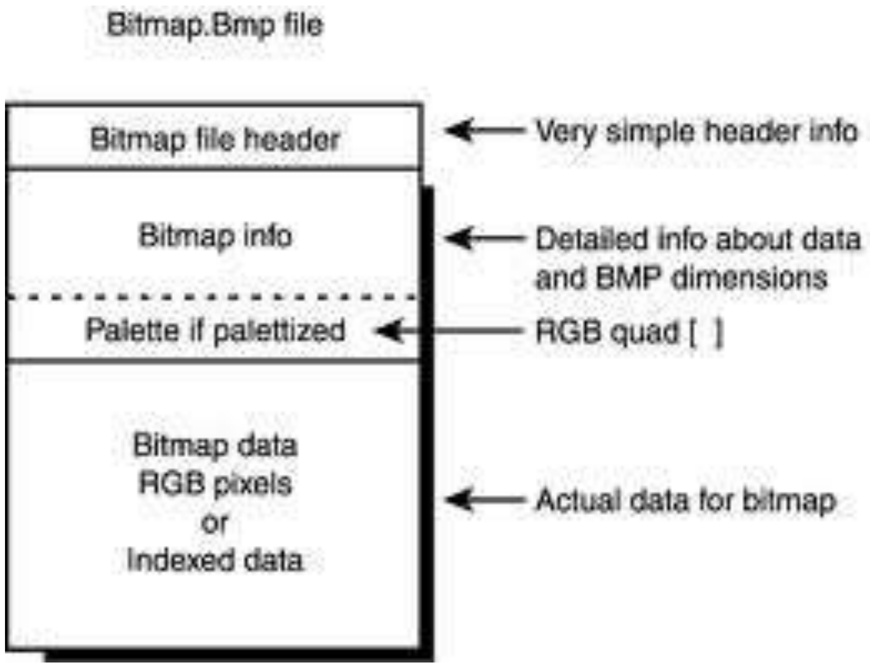
Compressed image data is read one byte at a time from file and into memory before it is decompressed.

Uncompressed image data is often stored only as bytes, even when the pixels are two three or four bytes in size.
Inverse of Reading a graphics file

Writers may send data directly to an output device, such as a printer or they may create image files and store data in them.

BMP
A Bitmap File (BMP) contains an exact pixel mapping of an image which can then be reconstructed by rendering application on the display surface of an output device.

BMP file consists of:
Header   Bitmap Data   Color Palette   other Data



The Header
Consists of binary or ASCII format data located at the beginning of the file., but the information about the bitmap data is stored elsewhere in the file.

A Bitmap Header is composed of
i. File Identifier
A header starts with a unique identification value called a File Identifier, file IP or ID.

ID values are assigned arbitrarily by the creator of the file format, and can be a series of ASCII characters

ii. File Version
Successive version of bitmap formats differ in header size, bitmap data supported and color capability.
After verifying the file format, an application examine the version value, determine whether it can handle the image data located in the file.
iii. Bitmap Data
In many bit map files Bitmap Data is found immediately after the end of the file header.

Bitmap data is composed of pixel values.

Sequence of values that logically maps bitmap data in a file to an image on the display surface of an output device

Pixels on an output device are usually drawn in scan lines corresponding to rows spanning the width of the display surface.

Image data is usually first assembled in one or more blocks of memory.

Exactly how the data is arranged then depends on a number of factors, including the amount of memory installed, the amount available to the application, and the specifics of the data acquisition or file write operation in use.

TIFF
Acronym of Tag Image File Format

TIFF specification was intended for storing black and white images, created by scanners

There are three sections in a TIFF file:
- Image File Header (IFH)
- Image File Directory (IFD)
- Bitmap Data

Image File Header is always at the beginning of the TIFF file, occupying the first eight bytes in every TIFF file.

IFH contains three fields of information

An identifier, used as byte order identifier and having a size of a word

A version , contains version number of the TIFF format.

IFD offset is a 32 bit value. It is offset position of the first image file directory in TIFF file. We use this value to identify the starting position of the image file information

One way to verify whether the file is a TIFF file is by checking the first four bytes of the file

Image File Directory is used to describe the bitmapped data to which it is linked

It contains information of height , width, and depth of image colors and compression types used and doesn't have a fixed size and position in a TIFF file

JPG

Standardized by Joint Photographic Experts Group
Used for storing, transmitting photographic images on web
Header section

- defines start of image data and
- sets parameters like image dimension and color format

Data Section

- contains compressed bitmap data itself

Uses lossy compression.
Output image, as result of compression, is a trade-off between storage size and image quality
Users can adjust compression level to achieve desired quality level and reduce storage size
Higher the compression value, higher the degradation in image quality
Markers
Differentiate various segments within a file, guiding software through decoding process.
Markers can

- indicate beginning and end of an image
- define a segment where metadata such as the color profile is stored.

This modular approach allows JPEG files to be highly flexible in storing and transmitting data, facilitating easy manipulation and retrieval of information without decoding entire file.
Color Profile in JPEG Image Formats

- Ensures that colors in your images appear consistent across different devices.
- Describes image's color characteristics, which helps maintain color fidelity when viewed on different displays or printed.

6.5 Visualization of Data Sets

Scientific Visualization:

The use of graphical methods as an aid in scientific and engineering analysis

For example, dealing with the output of high-volume data sources such as supercomputers, satellite and spacecraft scanners, radio-astronomy telescopes, and medical scanners.

Visualization techniques are useful for analyzing processes that

occur over a long time period

cannot be observed directly

Such as

quantum-mechanical phenomena

special-relativity effects produced by objects traveling near the speed of light

To visually display, enhance, and manipulate information to allow better understanding of the data, Scientific visualization uses methods from

computer graphics

image processing

computer vision

Business Visualization

Visualization methods employed by commerce, industry

Data Set Classification

Data sets are classification according to

Their spatial distribution

Data type.

Two-dimensional data sets have values distributed over a surface

Three-dimensional data sets have values distributed over the interior of a cube, a sphere

Data types include scalars, vectors, tensors, and multivariate data.

Visual Representations for Scalar Fields

A scalar quantity is one that has a single value.

Scalar data sets contain values that may be distributed in time, as well as over spatial positions.

The data values may be functions of other scalar parameters.

Some examples of physical scalar quantities:

energy density mass temperature pressure charge
resistance reflectivity frequency water content

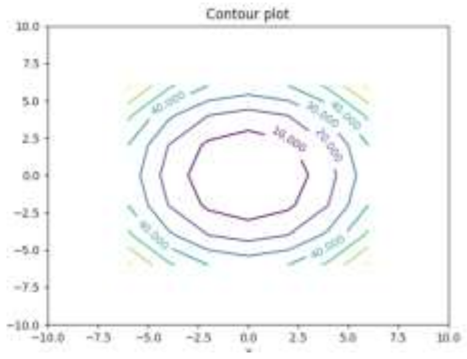
A common method for visualizing a scalar data set is to use graphs or charts that show the **distribution of data values** as a function of other parameters, such as **position** and **time**.

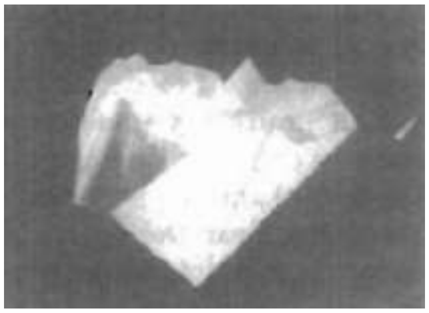
If the data are distributed over a surface, we could plot the data values as vertical bars rising up from the surface, or we can **interpolate the data values** to display a smooth surface.

Pseudo-color methods are also used to **distinguish different values** in a scalar data set, and **color-coding techniques** can be combined with graph and chart methods.

To color code a scalar data set, we choose a **range of colour** and map the range of data values to the color range.

For example, blue could be assigned to the lowest scalar value, and red could be assigned to the highest value.



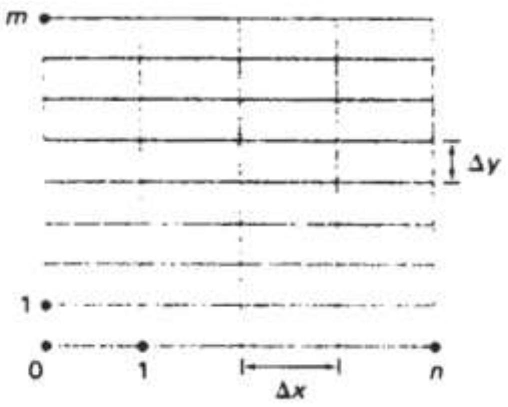


However, some color combinations can lead to misinterpretations of the data.

Contour plots are used to display isolines (lines of constant scalar value) for a data set distributed over a surface.

The isolines are spaced at some convenient interval to show the range and variation of the data values over the region of space.

A typical application is a contour plot of elevations over a ground plane.

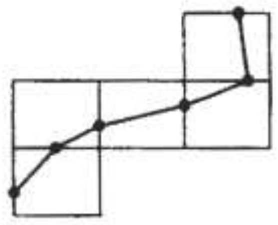


Contouring methods applied to a set of data values that is distributed over a regular grid

Regular grids have equally spaced grid lines, and data values are known at the grid intersections. Numerical solutions of computer simulations are usually set up to produce data distributions on a regular grid, while observed data sets are often irregularly spaced.

Contouring methods have been devised for various kinds of nonregular grids, but often nonregular data distributions are converted to regular grids.

A two-dimensional contouring algorithm traces the isolines from cell to cell within the grid by checking the four corners of grid cells to determine which cell edges are crossed by a particular isoline.

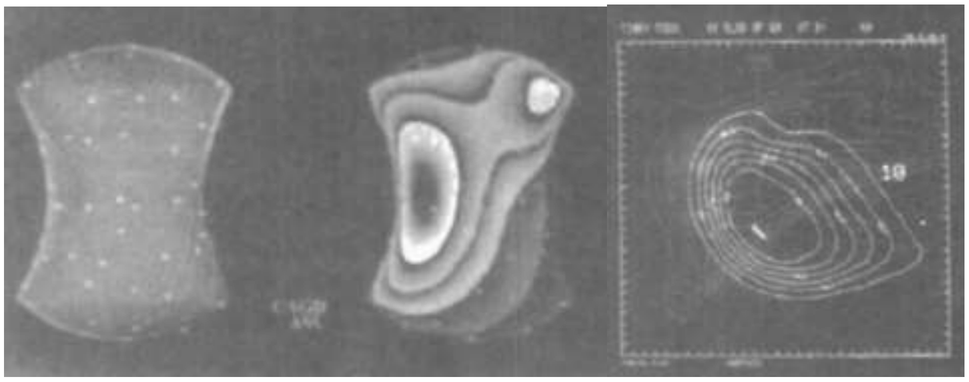


Isolines plotted as straight-line sections across each cell

Sometimes isolines are plotted with spline curves, but spline fitting can lead to inconsistencies and misinterpretation of a data set.

For example, two spline isolines could cross, or curved isoline paths might not be a true indicator of the data trends since data values are known only at the cell corners.

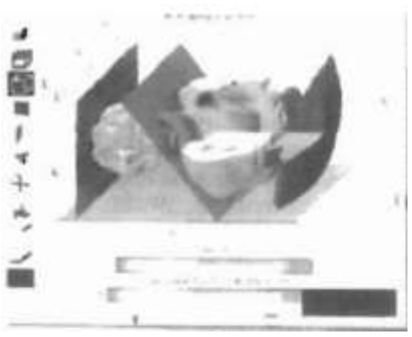
Contouring packages can allow interactive adjustment of isolines by a researcher to correct any inconsistencies.



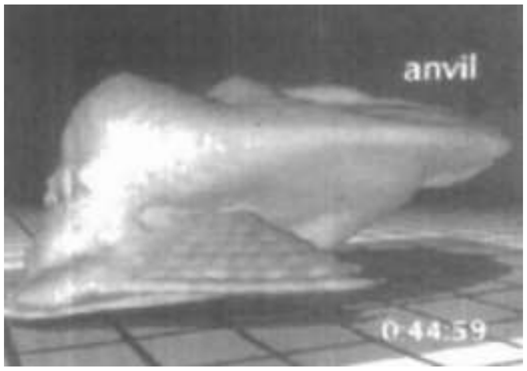
Contour lines and color coding for an irregularly shaped space.

For three-dimensional scalar data fields, we can take cross sectional slices and display the two-dimensional data distributions over the slices. We could either color code the data values over a slice, or we could display isolines.

Visualization packages typically provide a slicer routine that allows cross sections to be taken at any angle.



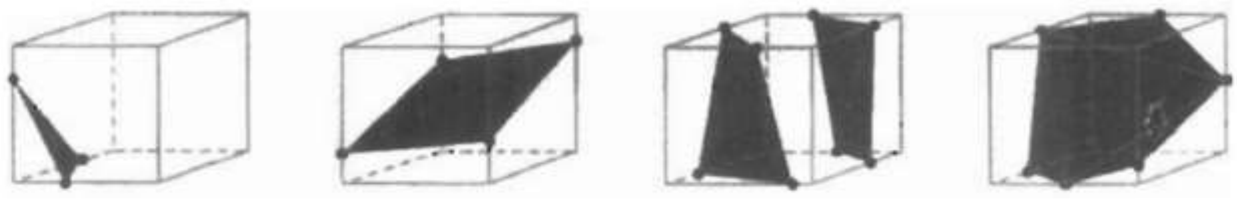
Slicerdicer package.



Three-dimensional contour plots IsoSurfaces

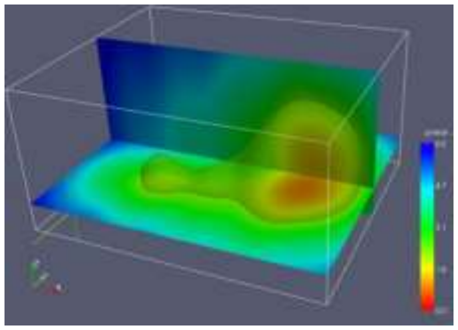
When two overlapping isosurfaces are displayed, the outer surface is made transparent so that we can view the shape of both isosurfaces.

Constructing an isosurface is similar to plotting isolines, except now we have three-dimensional grid cells and we need to check the values of the eight comers of a cell to locate sections of an isosurface.



lsosurface intersections with grid cells, modeled with triangle patches

Isosurfaces are modeled with triangle meshes, then surface-rendering algorithms are applied to display the final shape

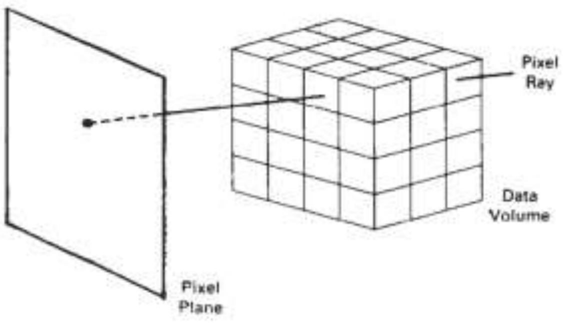


Isosurfaces

Volume rendering, which is often somewhat like an X-ray picture, is another method for visualizing a three-dimensional data set.

The interior information about a data set is projected to a display screen using the ray-casting methods

Along the ray path from each screen pixel, interior data values are examined and encoded for display.



Ray casting to Examine Interior Data Values

Often, data values at the grid positions are averaged so that one value is stored for each voxel of the data space.

How the data are encoded for display depends on the application. Seismic data, for example, is often examined to find the maximum and minimum values along each ray.

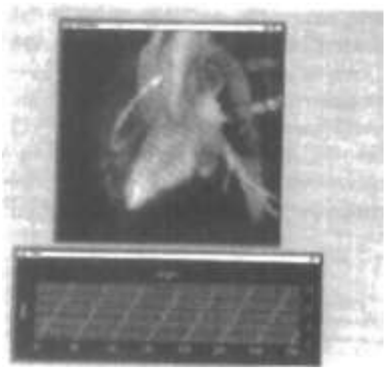
The values can then be color coded to give information about the width of the interval and the minimum value.

In medical applications, the data values are opacity factors in the range from 0 to 1 for the tissue and bone layers.

Bone layers are completely opaque, while tissue is somewhat transparent (low opacity).

Along each ray, the opacity factors are accumulated until either the total is greater than or equal to 1, or until the ray exits at the back of the three-dimensional data grid.

The accumulated opacity value is then displayed as a pixel-intensity level, which can be gray scale or color.



Volume Visualization of Dog Heart.

For this volume visualization, a color-coded plot of the distance to the maximum voxel value along each pixel ray was displayed

Visual Representations for Vector Fields

A vector quantity V in three-dimensional space has three scalar values (V_x, V_y, V_z) one for each coordinate direction, and a two-dimensional vector has two components (V_x, V_y).

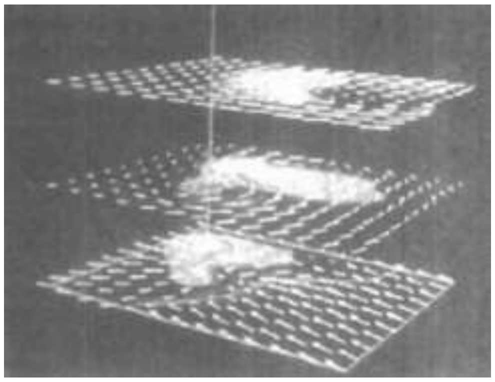
Another way to describe a vector quantity is by giving its magnitude IVI and its direction as a unit vector u.

As with scalars, vector quantities may be functions of position, time, and other parameters.

Some examples of physical vector quantities are velocity, acceleration, force, electric fields, magnetic fields, gravitational fields, and electric current.

One way to visualize a vector field is to plot each data point as a small arrow that shows the magnitude and direction of the vector.

This method is most often used with cross-sectional slices, since it can be difficult to see the data trends in a three-dimensional region cluttered with overlapping arrows.



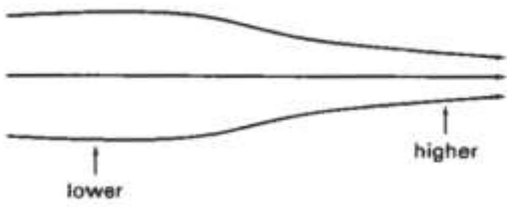
Visualization with Cross-sectional Slices

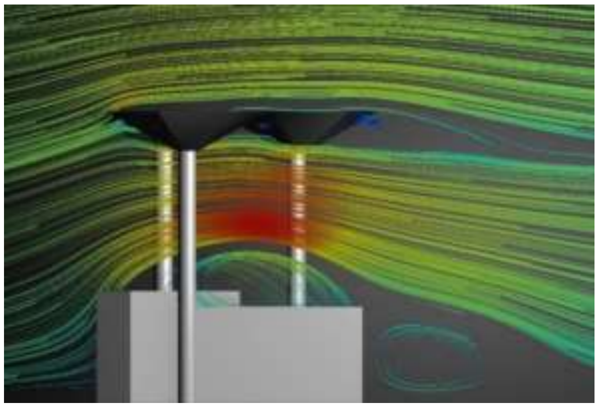
Magnitudes for the vector values can be shown by varying the lengths of the arrows, or we can make all arrows the same size, but make the arrows different colors according to a selected color coding for the vector magnitudes.

We can also represent vector values by plotting field lines or streamlines.

Field lines are commonly used for electric, magnetic, and gravitational fields.

The magnitude of the vector values is indicated by the spacing between field lines, and the direction is the tangent to the field.





Streamline Plot of a vector field

Streamlines can be displayed as wide arrows, particularly when a whirlpool, or vortex, effect is present. For animations of fluid flow, the behavior of the vector field can be visualized by tracking particles along the flow direction.

Visual Representations for Tensor Fields

A tensor quantity in three-dimensional space has nine components and can be represented with a 3 by 3 matrix.

Actually, this representation is used for a second-order tensor, and higher-order tensors do occur in some applications, particularly general relativity.

Some examples of physical, second-order tensors are stress and strain in a material subjected to external forces, conductivity (or resistivity) of an electrical conductor, and the metric tensor, which gives the properties of a particular coordinate space.

The stress tensor in Cartesian coordinates, for example, can be represented as

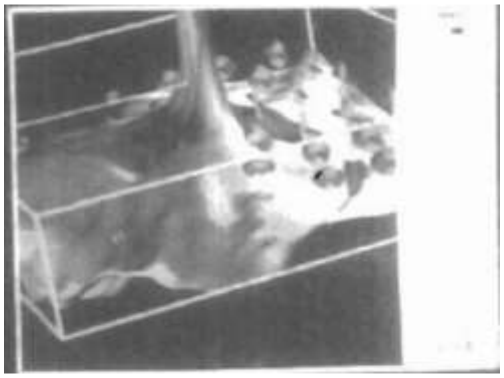
$$\begin{bmatrix} \sigma_x & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z \end{bmatrix}$$

Tensor quantities are frequently encountered in anisotropic materials, which have different properties in different directions.

The x, xy, and xz elements of the conductivity tensor, for example, describe the contributions of electric field components in the x, y, and z directions to the current in the x direction.

Usually, physical tensor quantities are symmetric, so that the tensor has only six distinct values. For instance, the xy and yx components of the stress tensor are the same.

Visualization schemes for representing all six components of a second-order tensor quantity are based on devising shapes that have six parameters.



Graphical representation for a tensor

The three diagonal elements of the tensor are used to construct the magnitude and direction of the arrow, and the three off diagonal terms are used to set the shape and color of the elliptical disk.

Instead of trying to visualize all six components of a tensor quantity, we can reduce the tensor to a vector or a scalar.

Using a vector representation, we can simply display a vector representation for the diagonal elements of the tensor. And by applying tensor contraction operations, we can obtain a scalar representation.

For example, stress and strain tensors can be contracted to generate a scalar strain-energy density that can be plotted at points in a material subject to external forces

Visual Representations for Multivariate Data Fields

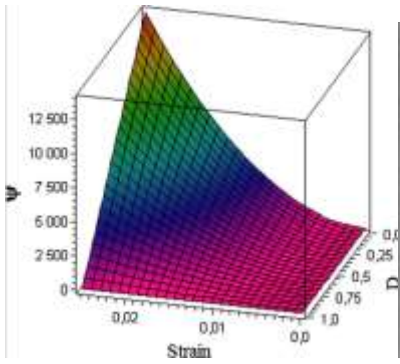
In some applications, at each grid position over some region of space, we may have multiple data values, which can be a mixture of scalar, vector, and even tensor values.

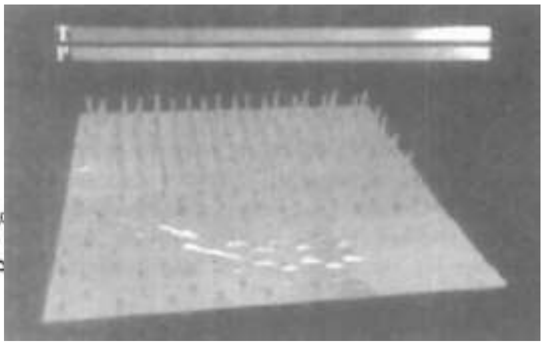
A method for displaying multivariate data fields is to construct graphical objects, sometimes referred to as glyphs, with multiple parts.

Each part of a glyph represents a physical quantity.

The size and color of each part can be used to display information about scalar magnitudes.

To give directional information for a vector field, we can use a wedge, a cone, or some other pointing shape for the glyph part representing the vector.





Visualization of a multivariate data field using a-glyph structure at selected grid positions