Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

P.J.W. ten Hagen, A.A.M. Kuijk, C.G. Trienekens

Display Architecture for VLSI-based graphics workstations

Computer Science/Department of Interactive Systems Report CS-R8637 November

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Display Architecture for VLSI-based Graphics Workstations

P.J.W. ten Hagen, A.A.M. Kuijk and C.G. Trienekens

Centre for Mathematics and Computer Science
P.O. Box 4079, 1009AB Amsterdam, The Netherlands

At present, two popular development areas in computer graphics are improvement of interaction behaviour and more realistic graphics.
The architecture for a high quality interactive workstation proposed in this work is designed such that both demanding and in a sense competing needs can be served.
Calculations for generating realistic full 3-D scenes with lighting, transparency, reflection, and refraction effects, are done on the workstation itself. Intermediate results are stored to locally serve high level interaction mechanisms.

1980 Math. Subject Classification : 69K31, 69K33, 69K37.

Key Words & Phrases :Workstation architecture, computer graphics, interaction, raster, VLSI.

INTRODUCTION

Image generation of three dimensional objects still is one of the major topics in computer graphics research. At the expense of sufficient processing power or time it is feasible to generate realistic three dimensional scenes, having shading, lighting, reflection and refraction effects.
For letting interactive workstations generate such high quality pictures, time is not available, so sufficient processing power has to be made available. Although the cost of processing power has reduced drastically in the past decade, it is still too high to put unlimited resources of processing power in a popular priced workstation. To offer both high quality pictures and good interaction behaviour, the commonly used strategy -speeding up the entire image generation pipeline- is too expensive.

The functional model of an arbitrary image generation pipeline following Carlbom[1] and Foley[2] is shown in fig. 1. Although this model applies to both vector and raster devices, only the latter type will be capable of producing the high quality

Report CS-R8637
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

pictures envisaged.
The model shows the different logical representation levels of objects that exist in the pipeline.

- The Application Model is the representation of a hierarchical object as determined by the application program. This representation may also contain non-graphical information.
- The Structured Display File contains purely graphical information of objects structured in a hierarchical manner.
- The Linear Display File contains the representation of objects in a form designed for optimal refresh speed. In a vector display this representation level may be absent. In a raster display this usually is the so-called frame buffer, in which the image is represented in a bit pattern.
- The Display which contains the resulting visible representation of the image.

Between each representation level, there is a logical processor that maps one representation to the next. A logical processor can consist of one or more physical processors or several logical processors can share one physical processor.
Each logical processor accepts input that influences the mapping process. In this way input can affect a representation level. The input is restricted by the type of changes it may cause on that level.

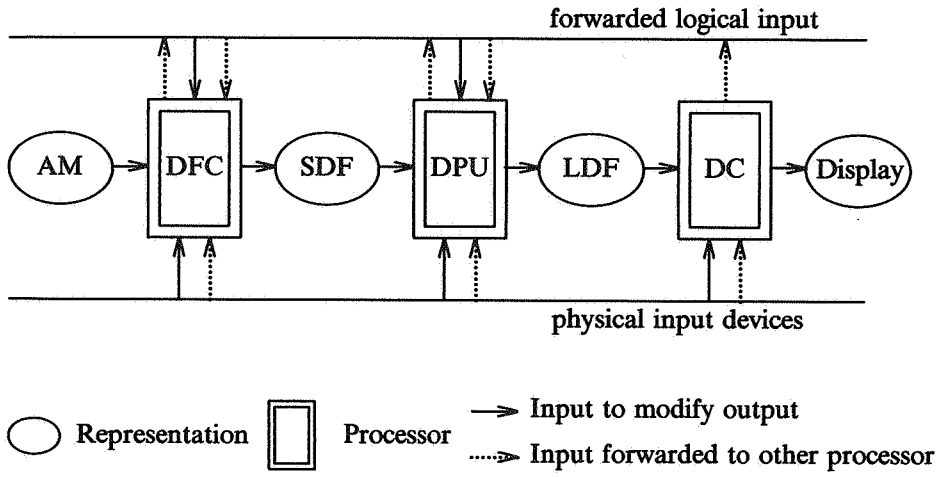

Fig. 1 Functional model of image generation pipeline (Carlbom[1]);
AM - application model, DFC - display file compiler, SDF - structured display file, DPU - display processing unit, LDF - linear display file, DC - display controller.

Speeding up of this pipeline is usually pursued by using multi-processor systems on the most time critical parts of the pipeline (DPU).

We will show that by strategically subdividing the image generating pipeline, we can relax timing requirements for the entire pipeline and still be able to offer -at moderate cost- good response behaviour for the high quality feedback mechanisms required for typical editing functions on an interactive workstation.

1. Display file and interaction support

It is clear that for a truly interactive display, the amount of calculations needed for system response, has to be minimized. From the functional model shown above one can see, that a (structured) display file, from which the interaction should be supported, should offer ways to minimize calculations. On one hand the logical level of such a structured display file should be high, say as GKS-segments, to be able to support high level interactions, such as segment dragging. On the other hand a lower level could be required for the purpose of fast redraw for low level interactions such as pick operations.

While looking after necessary features for achieving a fast real-time, interactive, display of 2/3-D scenes on a rasterscreen, some pilot goals for a display file are formulated:

- fast execution, when traversing the file. Especially for the primitives (and their attributes) it is very important that they don't need time-consuming processes when the file is traversed during display update.
- partitioned file, so as not to need to re-execute all of the file, when only one element of it changes.
- the file should be structured in such a manner that changing of objects represented in that file, can be handled easy and fast.

Fast execution. Fast execution results from fast execution of *each* individual primitive (and its attributes). It is up to the instructions/representations provided for each individual primitive to achieve fast execution (with or without using VLSI). As an example of a bad representation in this sense, one can think of a filled area, represented by its boundary. It is obvious that mapping of such a representation to a scan converted representation is expensive, but this will be even more expensive if self-crossing boundaries are allowed.

The performance of an interactive system will be limited by the slowest primitive. Hence, to assure an optimal interaction, each primitive execution should be fast.
The kind of primitives envisaged (RGF^3) are surface elements which can describe a 3D-object from its boundary representation. Each surface element is a pair consisting of a domain and a colourfunction, which assigns a colour to each point in the domain. The 3D-domains typically are flat areas bounded by polygons or bounded by B-spline curves, or similar such areas slightly bended (e.g. patches).

In this paper no detail is given about domain and colourfunctions and how they are represented. This will be described in a forthcoming paper. Both domain and colourfunctions are subjected to manipulations during interactions. It is obvious that such manipulations, in order to be dynamic, require fast execution of the transformed domain and colourfunction.

PARTITIONING. In order to be able to do partitioning, additional information about the environment is required, for instance, about the range of a picture. This however is rather high level information, that needs to be passed via the virtual terminal manager, who is responsible for terminal resource management. Although this is considered an important aspect of interactive workstations that can have a great impact on response, it is not discussed in this paper.

STRUCTURING. We concentrate in this paper on the second goal, display file structuring: what is needed is to make an inventory of kinds of changes that are wanted and/or required during interaction, and which representation levels of an object are needed to achieve fast and consistent execution of these changes.
Regardless of the primitives used, it is the interaction support which determines how the display file is to be structured and how it is to be converted into the linear display file. In the sequel the inventory of changes will be related to the display file structure, in order to obtain the proper structure requirements.

2. INVENTORY OF SUPPORTED CHANGES

Typically for interactions, changes are *incremental*, appearing mostly one at a time. Incremental changes suggests that the change is relative to an already existing value. Values of this kind should be in the display file explicitly. In this way, a change can be represented by updating this value. The change is effectuated by first letting this change propagate through the SDF (consistency!) and next generate from this a new LDF.

A further frequent property of interaction change is *locality*. This means that the change affects a relatively small part of the picture. If the corresponding part in the display file can be easily identified, it will be possible to minimise both the update efforts in the SDF as well as the generation of the corresponding LDF.

If an interaction change has a more global nature, it might well be of one type only, for example shift the position of all objects, or, increase the illumination. The structuring should be particularly helpful in supporting such changes. This type of change is called a *single aspect* change.

The inventory of changes will be further characterised, using the concepts incremental, locality and single aspect. For interaction, it is considered sufficient when such restricted changes can be dealt with in real-time. Changes requiring a complete recalculation of the picture are permitted to take longer (in measures of seconds rather than hours).

For dynamic interactive raster picture changes it is required that the following kinds of manipulations are supported:

- geometrical transformations, affecting geometrical aspects of both domain and colour function mappings
- colour control, merging colours for simulating shading, reflection, transparency
- appearance control attributes like highlighting, blinking, visibility, priority, depth
- delete or insert objects
- picking manipulations
- input feedback

Every manipulation, whether its effect is big or small, is considered a change. This change has to be shown on the screen instantly. In second instance, update of all higher representation levels to keep them consistent with the changed one can be done. This however only has to be done if the change is permanent.
Different requirements exist for manipulations of 3D scenes than for 2D. Calculations and representations in 3D are more complex and time-consuming than in 2D, because more in-between-steps are needed (e.g. hidden surface removal).

GEOMETRICAL TRANSFORMATIONS. The geometry of objects is represented by coordinate values, which denote points or vectors. How the set of points of a given entity is interpreted further, depends on the entity type. In any case, after changing the coordinate values (i.e. transforming), the entity description must be recalculated. We are especially interested in dynamic transformations, these are transformations that are executed and visualised in real time. Such transformations, typically are applied to groups of primitives (e.g. a segment) and one transformation at the time. The display file must have a level where such groups can be addressed and manipulated. Then the actual processing can be optimised.
The interactive environment should be able to characterise the requested change as being a particular transformation to a particular group only. In addition, the object properties that are to be affected should be identified. Associated with such requirements are hidden surface calculations, which can take advantages of the fact that it concerns a limited change. This excludes solutions such as z-buffer or quad-tree representations, where every change requires complete recalculation.

COLOUR CONTROL. Little attention has been given so far to provide firmware for dynamic control of colours. This is because current raster based 3D systems do not provide dynamics at all or exclude colour (e.g. in case of providing an extra pixel-plane for dynamic effects). Fast regeneration in our cases is intended to include colour function evaluation. This can be done if in the process of mapping the higher level display file on to the lower level, the area information and the colour function per area can be preserved. Then the effect of shading etc. can be realised by colour function compositions.

APPEARANCE CONTROL. The attributes which traditionally (c.f. GKS) are used for creating (unnatural) dynamic effects, remain equally useful. They are primarily for fast low level feedback. They affect either groups or individual output primitives. In both cases the corresponding elements must be easily tractable in the display file. This introduces no further requirements beyond those already encountered for transformations.

DELETE AND INSERT. Deletion and insertion (as well as priority and visibility changes) require local rearrangements for proper hidden surface removal and lighting effects. In order for such rearrangements to be calculated quickly, followed by a reexecution of (part of) the hidden surface removal which also must be fast, a special hidden surface removal system will have to be developed.

PICKING MANIPULATIONS. Picking and input feedback will be best served by functionality that identifies the relevant elements in real-time. Low level feedback will assist during picking or other input. Higher level feedback, indicating restored consistency after an input completion will use upward references realised either by an explicit administration or by fast searching traversal.

3. REPRESENTATION LEVELS AND DISPLAY FILES

Displaying 3D-objects in graphical packages like GKS[4] is effectuated by one intermediate representation: the segment representation. From this segment representation, the picture generation is initiated. In terms of the functional model this representation is the so-called Structured Display File.
To support the picture generation is not the only reason of existence of the segment representation in GKS. Segments are not thrown away after their generation but are kept stored in the workstation memory to allow for segment manipulations such as copying, transformations, deletion, change of priority etc.

In the display architecture we propose here, we distinguish between (at least) three intermediate representations of an object. For each of these representations one can think of similar arguments to justify permanent storage of these representations. These representation levels show up when performing the complex mapping of 3D-objects. On its way through the image generating pipeline, each object subsequently passes through these representations. It remains to be investigated if all three intermediate representations of an object really have to be stored in memory during the lifetime of that object.

Next we will present the three levels subsequently called high, medium and low.

3.1. High Level

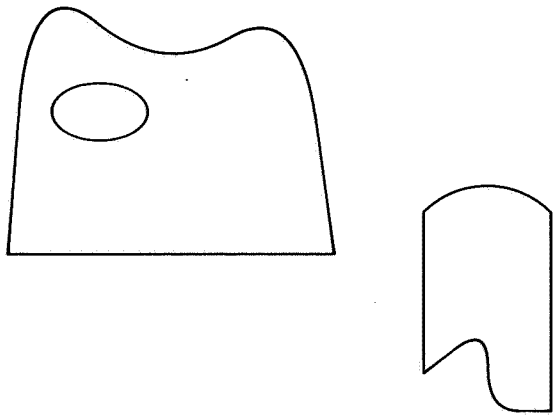

Fig. 2 High level;
On this level several primitive types are allowed, although just fill-areas are shown. Complex primitives defined by domains and colourfunctions are supported. The representation is in continous coordinates.

The highest level of our three intermediate levels is the level that contains an object representation directly resulting from the user-definition, and his own coordinate system. This level is similar to the segment representation in GKS. In the functional model this would be the Structured Display File. We will not discuss the higher level Application Model representation, because that representation level is not a part of the actual graphical package, but a part of the application using the graphical package.
Viewing and segmentation are done on this level. All objects in this level need to be in one coordinate system. This is still a continuous coordinate system to avoid loss of information and introduction of resolution dependencies.

On this level there is in principle no limit on the types of primitives that can be represented. Simple primitives such as filled areas, lines, characters, markers but also more complex primitives such as B-splines etc. Also simple and complex attributes such as colours and colour functions (i.e. functions that map colours on a domain) can be represented on this level.

3.2. Medium Level

Here, each stored element represents a domain with just one colour function. Disjunct domains are split up and stored in separate, continuous elements. As an example, a text string containing individual characters will on this level be mapped

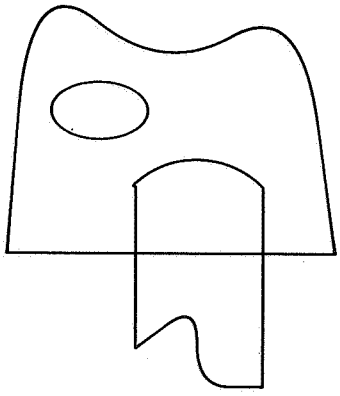

Fig. 3 Medium level;
On this level just one type of primitive, a domain with a colour function, is allowed. Primitives may overlap on this level. Coordinates are continuous.

on as many domains as there are characters in the string (provided the characters themselve are described as continuous areas). On this level the primitives can be overlapping. The representation is specially chosen to allow for efficient hidden surface removal.

Summing up the properties of this level:

- it contains only one type of primitive, which fills an area
- each area is one connected region
- each area corresponds to exactly one primitive from the high level, but the reverse is not necessarily the case
- the element representation is such that hidden-surface-removal calculations can be done very efficiently.

3.3. Low Level

This representation level is the finally resulting file of (at least partly) visible, connected areas from the original objects. In the functional model it is called the Linear Display File. This level acts as refresh buffer that directly supplies the information to the refresh process. As such, the representation has to be optimized for this purpose.

This display file may contain either only disjunct areas or the elements in the display file are ordered in such a way that the last element encountered in an overlapping situation sets the colourfunction.
If this level allows only disjunct areas, the hidden surface algorithm has to generate

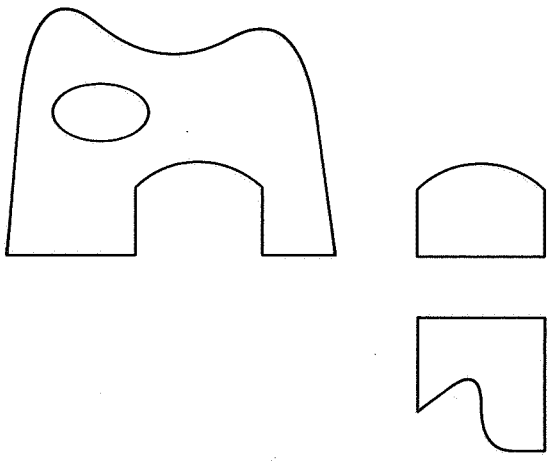

Fig. 4 Low level;
On this level all the areas are disjunct. No more hidden surfaces exist. The colour function for each area is a result of a combination of the colour functions of all the primitives that contribute to that area.

only noncovered areas, but there is not need for these areas to be sorted.
On the other hand, if overlapping areas are allowed, the hidden surface algorithm is relieved of having to remove parts of areas that are overlapped. In this case however, the areas will have to be sorted in such manner that the overlapping area overrules the pixel settings of the overlapped parts. This can cause the next stage -the mapping to pixel values as performed by the Display Controller- to do more than actually needed.

We prefer the disjunct area approach; The low level is intended to serve as refresh buffer, hence we cannot afford to have to go through superfluous information and let the refresh process do more than actually needed. Furthermore, we intend to allow the merging of colour functions to handle transparent objects. To do this, we will need disjunct areas.

At this moment we will not decide yet if the representation on this level is already in discrete or still in continuous coordinates.

3.4. Comments

Originally on the high level all primitives have their own intrinsic colourfunction. On the medium level there is an alternative representation of a primitive but still with its original colourfunction. However, on the low level, the colourfunction could have been changed, because of effects such as merging objects for transparancy, shading and the like as defined by the application/user.

Given the three intermediate representation levels as discussed, following Carlbom[1] the functional model of this architecture will be as shown in fig. 5. Compared to the model shown in fig. 1, just one level has been added. The important difference with conventional raster display architectures however, is that the lowest level -being the refresh buffer- in our model is a structured display file, whereas in most other architectures this is a frame buffer.

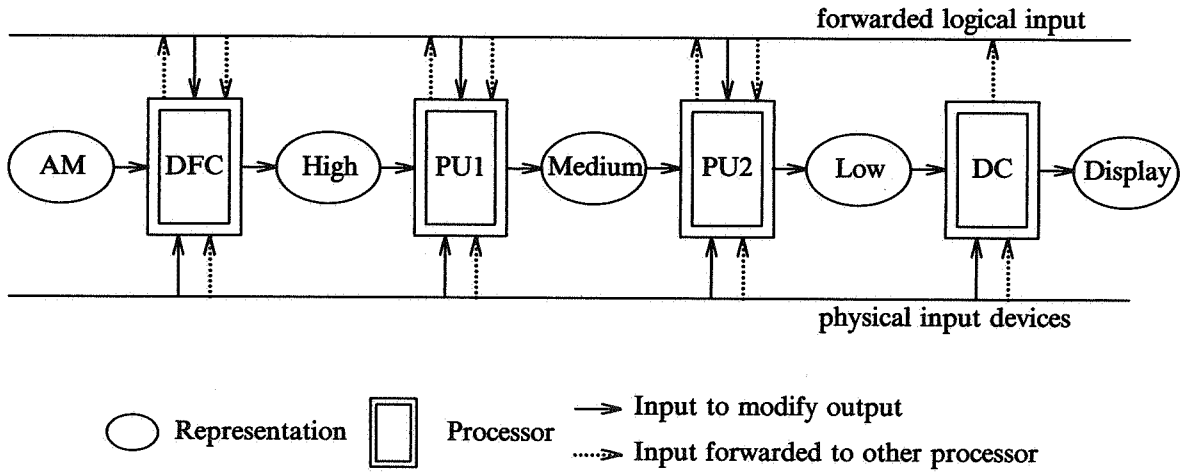

Fig. 5 Functional model of the proposed architecture;
AM - application model, DFC - display file compiler, High - structured display file, PU1 - processing unit to convert primitives, Medium - structured display file prepared for HSR calculations, PU2 - processing unit to handle hidden surface removal, Low - structured display file that acts as refresh buffer, DC - display controller.

A frame buffer contains the raster image in a bit pattern form. Its main reason of existence is to cause a decoupling of the refresh process and the scan conversion process. Scan conversion usually cannot keep up with the refresh rate a raster diplay needs.

Advantages of a frame buffer are:

- Simple decoupling of refresh rate and scan conversion process. This allows the refresh process to continue independent of the complexity of the image.
- Frame buffers lend themselve to separately mask off bitplanes. This feature can be used, for instance, to use one of the bit planes for interaction response, or to do image switching.

The price one has to pay if a frame buffer is included in an architecture is rather high:

- The interaction time that can be offered by such a system depends totally on the

rate at which the frame buffer information can be updated. Since two processes (refresh and scan conversion) compete for the access to the frame buffer memory and image storage in this form is not compact, the bandwith of such a memory has to be high. This will be even worse if higher resolution is required.

- The information in a frame buffer is unstructured. This limits the types of manipulations possible on the information.
- The resolution of the frame is tied to the resolution of the display, which limits the picture quality.
- The limited information per pixel introduces limits, for instance on available colours.
- Frame buffers cannot support accurate control of changes, e.g. a change becoming visible at once, unless a double frame buffer is used.

The nature of these defects is such that trying to improve on one point, will make things worse for others.

Advantages of using a structured display file as a refresh buffer are:

- Information in a display file is compact, which reduces the bandwidth requirements of the memory.
- Information is structured, which makes it more simple to manipulate parts of the image.
- Information is more object oriented, which makes is better suited for interaction feedback mechanisms such as object picking.
- A further set of advantages is associated with the low level display file not being a frame buffer, but rather a set of disjunc area's. This makes feedback on that level on a per (visible) primitive basis easy. Also local changes can be effectuated immidiately. At the same time correlations with encompanying objects can be made direct.

Of course also a display file has disadvantages:

- The size of the display file is related to the image complexity. The compactness of information in the display file assures that only in a worst case situation the size of the memory required for display file storage would be comparable to the size needed for frame buffer storage.
- Since the refresh rate is fixed, the complexity of a picture that can be processed within the refresh cycle is limited. By making use of scanline coherence and due to an optimized representation and nowadays processing capabilities, we expect this limit to be far above what reasonably can be expected of a now common workstation.

In what follows we will investigate the three levels of representations to decide which level is the most suitable for catching a certain picture change in order to let

this happen as fast as possible. If a representation level seems to be essential for achieving fast changes it is obvious that this representation should be kept in the workstation memory.

4. Display list representation levels and changes

To achieve a fast update of the display, changes should be initiated on the lowest level possible. In general it will hold, that the lower the level on which a change can start, the better the performance would be. Starting at a high level will generally be a much more time-consuming process. But small changes still can be fast. This due to the fact that in that case, the result of a previous and similar calculation is strongly coherent.

It is necessary to maintain consistency in the three intermediate representations. For instance, in case of 3D, to recover hidden elements after a change, going back to higher levels is necessary. Hence, when pointing to an element from the low level, its corresponding element in the medium level representation has to be known. Similarly, for each element in the medium level, the corresponding element in the high level has to be known.

In most types of interactive changing operations, the changes only need to be maintained temporarily on the screen. They belong to the lowest (real-time) feedback. For instance, a cursor moving across a primitive temporarily adds the cursor icon on top of the primitive. When the cursor moves further the original image of the primitive must be restored. These instantaneous effects need to be done on the lowest possible level. As soon as the changes are permanent in the diverse situations there may be a need to penetrate into higher level(s) to obtain consistent update of data in the existing representations.

In fig. 6 it is indicated on which representation level the various changes should be effectuated (visualised). If a change can be effectuated on a certain level it can also be undone (on an individual base) without assistance from higher levels. Thus, feedback based on such a change can be dealt with entirely on that level.

A change on the low level affects only one element, or a small number of elements, i.e. one of the disjunct, connected areas, at the time. The feedback cycle consists of: select element(s), change the appearance, restore the original appearance. For instance this is the way a cursor could move across the screen or through the 3D space. The hardware should in this case be able to determine which low level element is at a given pixel or 3D-device coordinate position. Consequently the lowest level feedback can deal with the visible elements or visible parts of objects only.
Typical operations that can be handled on the low level are pick operations and a change of appearance control attributes such as: highlight, blink and depth.

As soon as an entire primitive or object is to be changed it will have to be

Operation	Representation level		
	Low	Medium	High
dynamic attribute controls:			
highlighting	●		
blinking	●		
depth	●		
pick	●		
priority		●	
visibility		●	
colour static transformations:			
transparency		●	
shading & reflection changes		●	
individual geometric transformations:			
scale, translate, rotate		●	
clip		●	
change, replace		●	
element manipulations:			
insert, delete		●	
viewing control			●
grouping			●

Fig. 6 Representation levels and changes;
This table shows from which minimal representation level the various changes can be effectuated. In principal, changes can also be effectuated from higher levels, allowing higher level feedback.

regenerated from the medium level. Hence, every manipulation that affects only one entire individual primitive must be possible on the medium level (individual object manipulation, geometry transformations on individual objects).

Changes that affects uniformly a whole group of elements, for instance a GKS-segment or PHIGS-hierarchical group, should be initiated at the high level representation.

Although changes can be effectuated from a certain level, one can think of operations that may have several modes, which, for feedback support, need to penetrate to higher levels. As an example we can take the following modes of

object picking.

- Picking of elements. Feedback on picking of elements in this mode can be done by highlighting the one (visible) element that is picked. This element can be one area of a set of areas, together forming one primitive of the higher level. Feedback of this mode of picking can be handled entirely with information stored in the low level.
- Picking of primitives. Feedback of this pick operation mode would be highlighting the entire picked primitive (i.e. all visible and invisible elements of that primitive). Clearly in this mode of operation information as present in the medium level is needed.
- Picking of segments. All the primitives forming the segment would have to be highlighted in this mode, that for that reason needs the information stored in the high level.

In all these three modes, the initial identification is done at the low level. However, the feedback cycles of all three modes differ in which levels are needed for assistance. Obviously, this difference in feedback cycle will reflect in the response time. Due to structuring of the display files and the fact that feedback usually is a change of the restricted type as mentioned in section 2, real time behaviour still can be guarantied.

5. Conclusion

In conventional raster graphic display architectures, the frame buffer update is one of the most serious bottlenecks for interactive 3D visualisation. Most of the so far proposed solutions, try to tackle this problem by parallelising the update process in some way, or by adding simple logics in the memory itself to be able to perform frame buffer manipulations.

Our approach differs radically from those solutions, in that we completely avoid the need for a frame buffer by a proper design of a display file with a sufficient efficient representation to be able to serve as a refresh buffer. The thus resulting architecture has a lot in common with the conventional highly interactive vector display's.

All of the three levels proposed in this paper, seem to be essential for efficient support of some frequent interaction mechanisms. Thus a solid ground to maintain these three levels as separate files does exist. An implementation scheme might be optimised by sharing common data elements of the various files.

The actual realisation of the proposed architecture, will consist of VLSI implemented modules (together forming the logical processors PU1, PU2 and DC in fig. 5), that will perform the mapping functions needed to generate the various display files.

References

1. I. Carlbom, *System architecture for High-Performance Vector Graphics,* Dept. of Computer Science, Brown University, Providence (1980). Ph.D. thesis
2. J.D.Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics,* Addison-Wesley (1982).
3. P.J.W. ten Hagen, M.M. de Ruiter, and C.G. Trienekens, *Raster Graphics Facilities (RGF),* CWI, Amsterdam (1986). preliminary report
4. ISO, "Information Processing - Graphical Kernel System - Functional description," International Standard DIS 7942 (1985).