Looking at Workstation Architectures from the Viewpoint of Interaction

Detlef Krömker

Technical University of Darmstadt, Graphical Interactive Systems Group
Alexanderstr. 24, D-6100 Darmstadt

Introduction

Today's design of sophisticated graphics workstations may be characterized by the terms 3D-system, user driven, object-oriented user interface and multiple-windows system with the challenge to create high level interfaces for the application programmers. All these properties require a great amount of computing power, especially if we look at 3D-systems with high images quality. On the other hand it is well-known that speed, which means system response time, is the most important aspect of interactive systems. More than any other attributes, speed decides whether a new system or technique is acceptable or not. *"Not only did the speed make the user happier, but productivity went up."* /Brad-85/ This will be the first point of discussion treated in this article followed by a preview of current architectures, a short analysis of interaction, an observation of implementation techniques and finally pointing out a new hardware approach for the implementation of very fast interactive systems.

Response Time Requirements for Interactive Workstations

The following reflections base on studies done within IBM /Brad-85/. They indicate that one of the prime parameters of productivity, while using an interactive computing system, is the response time. One of the key questions in workstation design from the economic point of view is: *Does a minimal system response time exist, so that further investments in hardware or software make no more sense, because they will not increase the productivity of the users? And if it exists, what is it's magnitude?*

The answers are very interesting. To the first question, measurements showed a very substantial increase of productivity when response time decreases to below 1 second and even more if it goes down to 0.3 seconds. Fig. 1 shows the relationships. By using cognitive studies Brady made productivity projections for response times lower than 0.3 seconds (Fig. 2).

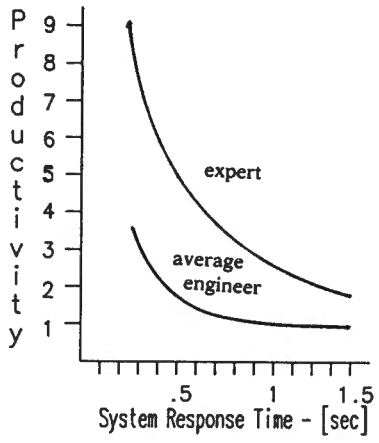

Fig. 1: The productivity of interactive engineering users in a high-function graphics application as a function of system response time /Brad-85/.

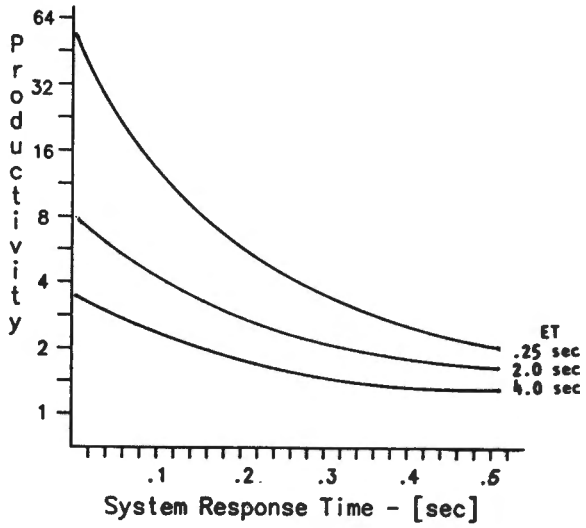

Fig. 2: Productivity projections as a function of response time and transaction entry time (ET) /Brad-85/.

As the second important parameter of productivity the transaction entry time was exposed. Out of the results we conclude:

- decreasing system response time down to 0.1 seconds (or even lower) seems to be most economically,
- the interactive system has to give as much assistance as possible to the user, to reduce the entry time.

Comparing these requirements with today's systems we have to make an important note: the typical response time of a CGI-System (Computer-Generated-Imagery) in simulator enviroments is 0.1 seconds. Such a system with the performance of several thousands of faces and an image resolution of one million pixels needs about 100 Mips for symbolic computation tasks (scene managing and geometric processing) and about 10,000 Mips for pixel computation (raster scan, color computation and hidden surface removal) /Yan-85/. The symbolic computation rate depends mostly on the number of primitives, while the pixel computation rate depends on the type of primitives, the image resolution and the quality requirements. This high processing rate will keep the pixel computation part of CGI-Systems in the domain of special purposes hardware for the forseeable future /Yan-85/.

Even with the support of very fast development of new VLSI technologies, we will not be able to bring such a computing power to a commonly used workstations in the near future. This forces the following questions: ***Are there different requirements for CGI-Systems and (editing-) workstations? Furthermore, are future system architectures both the same for CGI-Systems and workstations or not?***

A Preview of Current Architectures

For further discussion on architectural concepts, we must keep in mind the enormous difference between the costs of symbolic and the costs of pixel processing. Fig. 3 shows a primitive model of interactive graphics systems. The most time consuming processes in these systems are raster scan process, illumination process, and hidden surfaces removal process. They are located at the raster scan processor. For realtime generation of moving images the raster scan processor needs more than 95% of the overall computing power. This leads to the use of highly specialized processors for only a few and simple primitives. Contrary to early CGI-Systems, nowadays there is the trend frame buffer based systems /Yan-85/. The main reason is the possibility of better balancing the multiple scan processors and therefore less ***bad*** images with the same computing

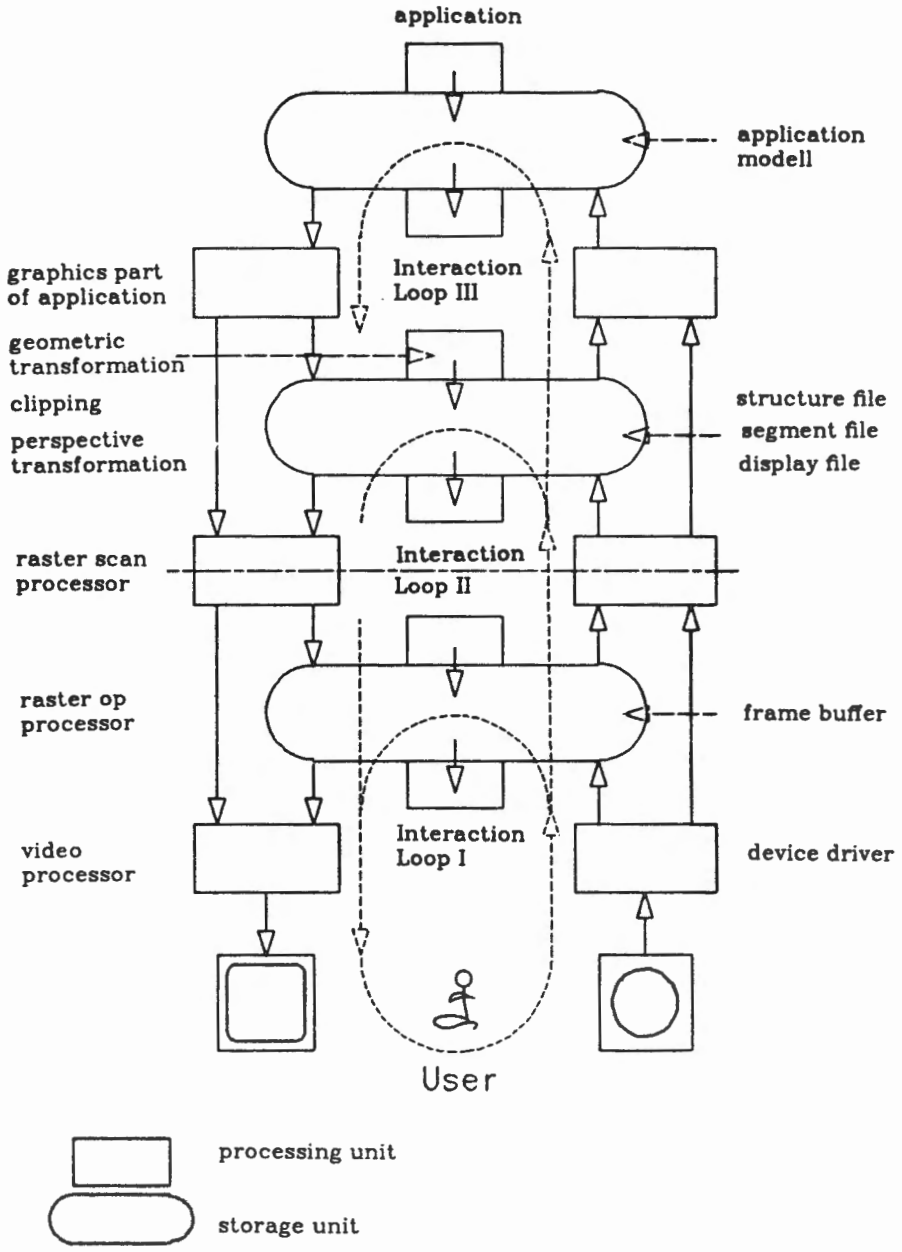

Fig. 3: Model of a interactive graphics system.

power. Due to the frame buffer, one can generate high quality images with antialiased edges, transparency and textures using quite simple frame buffer algorithms.

One has to note that the most important lack of all realtime approaches which avoid the use of a frame buffer is the limited complexity of displayable images. This fact, and the few primitives supported in such systems, make a frame buffer approach necessary for general and limited purpose workstations (see also /Engl-86/).

Therefore, frame buffer is the lowest representation level of the model in Fig. 3. A medium representation level is a structure file, and/or a segment file, and/or a display file. In advanced systems, there are often more than one storage units as medium representations, which will extend the pipeline only and will not change the principles as shown.

A short Analysis of Interaction

Ten Hagen, Kuijk and Trienekens /THKT-87/ gave a very good inventory of changes required in editing applications which were discussed on the Hardware Workshop EG 86. They identified three major characteristics of interaction: ***incremental***, ***locality*** and ***single aspect***.

These attributes of interactive changes must be seen with respect to a given representation level and the implementation of this level. E.g., in the 2D-case deleting of an object is incremental to a structured display file but ***not*** incremental to a normal frame buffer if this object was written in REPLACE-Mode. But of course it is incremental to the frame buffer if the object was written in EXOR-Mode. If a change is not incremental to a given representation, it is necessary to reconstruct this representation completely from the next higher level, which is called update. Depending on the processing task between these two representation levels this may require very high computational costs, e.g. if this is a update of the frame buffer from a symbolic display file.

From the implementer's point of view, it is very useful to add an additional characterization. We may distinguish between three types of changes. One type has a ***temporary*** nature and will not affect the application model. These are changes like moving of cursors (maybe with rubberbanding etc.), highlighting of objects, dragging of objects, popping up/down menu fields etc.. Another type of changes has a ***persistant***

nature. A temporary change may be seen as an intermediate state within a user transaction, whereas a persistant change marks the end of a user transaction with a change in the application model. A third group of changes are *assistance functions*, i.e. grids, scales, arrangement of different views (windows), etc., which have great power to reduce the entry time.

By using the attributes of interaction mentioned above we are able to identify the main differences beetween CGI-Systems and editing workstations: Most interactions in CGI-Systems change multiple aspects, forcing global changes of the image which are not incremental to the lowest representation level (the frame buffer), and have a persistant nature. For editing applications, a general characterization as above is impossible. but there are important differences compared with CGI-Systems. Nearly all inputs change single aspects, the requested changes are often local. A lot of inputs force changes with temporary nature or request assistance functions, and some are incremental to the frame buffer.

To answer the first question noted above, we found that the requirements for CGI-Systems and editing workstations are not the same. To answer the second question we must concentrate on the attribute *incremental*, especially with respect to the frame buffer. (Because the raster scan process needs the highest computing power.)

Implementation Techniques

Up to now many techniques have been developed to assist a quick update of the frame buffer. We cannot discuss them in full details but we will identify their main benefits and disadvantages. There are two main approaches:

- making more updates incremental to the frame buffer and
- using the video processor to manipulate the video output stream.

Typical implementations of the first approach are:

— Information Conserving Write Modes, i.e. EXOR, ADD/SUBTRACT, etc. Since these operations are reversible, a selective erase of objects is possible. They are useful e.g. for highlighting, but in general, only if the backround is wellknown, otherwise their uses may cause bad color effects.

- Independent Bit-Planes: They allow writing to the frame buffer without effecting others than the enabled planes. This technique is very useful for displaying a restricted number of objects with limited colors, e.g. for temporal changes (echo planes).
- Raster Ops (BitBlts,PixBlts): They are known as the implementation technique used commonly for multiple windows systems. If there is enough free space (i.e. parts of the frame buffer which are not visible on the sreen), the use of raster ops allows virtually an incremental update of the frame buffer (with a save and reload of the affected part) almost in realtime. Additionally they allow quick generation of a limited set of symbols, e.g. text or icons, but no ***free*** graphics. They are often used with information conserving write modes having the same disadvantages as mentioned above.

Typical for the second main approach are techniques like *Hardware Windowing* (e.g. INTEL 82786), *sprites* (Commodore C64, AMIGA, Atari VCS /WiER-85/), *color lookup tables*, *realtime cross hair*, etc. All these functions are usefull to implement realtime response, but they are restricted to very few system response types and they are not able to solve the general problem.

By this short survey of implementation techniques we find:

- All techniques are restricted to special tasks and they do not generally solve the problem, although we have combinations of different techniques.
- Especially for 3D-systems they only allow a very few persistant changes incremental to the frame buffer.
- None of these techniques assist the important function of picking identification of objects.

The main disadvantage of the frame buffer is the unstructured storage of information. This leads to an approach, in which structure information is added to the frame buffer.

Structure Storage Approach

Fig. 4 shows the basic idea. Parallel to the frame buffer (and in case 3D-systems to the z-buffer, too), we have one additional storage element for every pixel, the so-called ***structure storage***. It stores the combination of objects covering the corresponding pixel. The content may be interpreted as structure information, then the background frame buffer holds the image information, or directly as additional image buffer which holds an overlay image to the backround frame buffer. In this case we need an additional lookup table which is controlled by the structure storage.

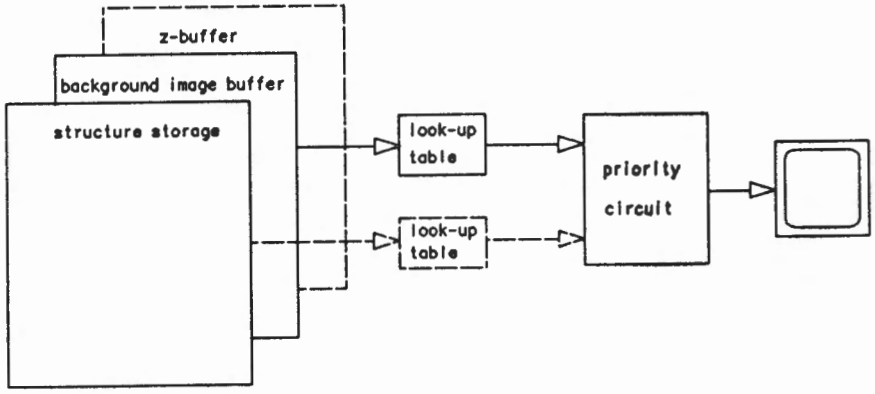

Fig. 4: Basic approach of a structure storage.

The basic operations on a structure storage are

- write (an object) with id x,
- erase an (object) with id x,
- output of the id's of objects which are affected during the last write/erase operations,

— output the id's of objects covering a given pixel.

The main problem of this approach is the amount of required bits for each pixel. This amount will be shown by the following analysis.

Assuming the priority sequence of objects is fixed (2½D-case) or there is no priority attached. Tab. 1 shows the number of combinations to be stored in the worst case. The number of bits to represent any of these cases is ld (γ_2) = n. This leads to a natural coding N, which requires one bit for each object:

1: object covers the pixel

0: object does not cover the pixel.

For the 3D-case, where the priority of objects may change the complexity is $\gamma_3 = n \cdot 2^n$.

n Objects	number of Combinations
0 Objects(transparent)	1
1 Object	n
2 Objects	$\binom{n}{2}$
.	.
.	.
.	.
n Objects	$\binom{n}{n}$
$\gamma_2 =$	2^N

Tab. 1: Complexity of the potential covering of objects.

The number of bits to represent any existing case is too high to use a direct (e.g. the natural) coding. Normally, the number of objects covering a given pixel is much smaller than γ. Therefore, we use a hash function to compute an index I which is stored in the structure storage. The key for the hash function H is the natural code N. A useful hash function is the division by a polynom $P(x)$.

$$H(N): \quad I = N \bmod P(x).$$

The minimal storage costs for some implementations are as follows.

structure storage	hash table	max. number of objects
8 bits	256 × 24 bits	32
16 bits	65K × 48 bits	64
16 bits	65K × 128 bits	142

Conclusion

The architectures of editing workstations and CGI-Systems are not necessarily the same, even both systems have the same response time requirements. These results are from different interaction requirements which are typical for these applications. Future research has to realize these differences in the development of new architectures for editing purposes. (As it was the introduction of raster ops some years ago.) The proposed method is able to support a system to achieve a very short response time for most interaction steps. This promises a higher reduction of costs than it is given by the common approaches to speed up the output pipeline.

References

/Brad-86/ J.T. Brady, *A Theory of Productivity in the Creative Process*
in: Proceedings of the 1st International Conference on Computer Workstations; San Jose, CA; Nov. 1985
reprinted in: IEEE Computer Graphics & Applications; Vol.6, No.5, pp. 25-34; 1986

/Engl-86/ N. England, *A Graphics System Architecture for Interactive Application-Specific Display Functions*
in: IEEE Computer Graphics & Applications; Vol.6, No.1, pp. 60-70; 1986

/THKT-86/ P.J.W. ten Hagen, A.A.M. Kuijk, C.G. Trienekens, *Display Architecture for VLSI-based Graphics Workstations*
Within this book.

/WiER-86/ G. Williams, J. Edwards, P. Robinson, *The AMIGA Personal Computer*
in: BYTE; August 1985; pp. 83-100

/Yan-85/ J.K. Yan, *Advances in Computer Generated Imagery for Flight Simulation*
in: IEEE Computer Graphics & Applications; Vol.5, No.8, pp. 37-51; 1986