Architectures for Interactive Raster Graphics

Fons Kuijk and Robert van Liere
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

For an interactive graphics system two major information streams can be recognized, information for the image synthesis process and information related to user input. In this paper, we present a logical model for the image synthesis pipeline and a logical model for input devices. Based on these abstractions we describe an interactive raster graphics system. Requirements that go with interactive systems make it important to discuss implementation of this architecture on parallel systems. For this we emphasize on rasterization hardware and system configurability.

1. Introduction

Research on raster graphics at CWI has been largely focused on satisfying the requirements for interactive environments. Graphics plays an important role in the area of user interfaces, an outstanding example of an interactive environment (see H.J. Schouten's contribution elsewhere in this issue). Requirements of such user interfaces, high resolution displays and fast interaction facilities, are becoming standard features of today's desk-top workstations. The increased capabilities of such personal workstations triggered more visual oriented communication. Consequently, user interfaces have become powerful tools. However the interactive graphics response takes up a considerable amount of computing resources.

One can foresee that the next generation of workstations would need even more demanding visualization facilities, such as rendering realistic images of three-dimensional objects that can be manipulated in real time. Realistic images involve high resolution systems and advanced shading models, including high quality illumination, distance attenuation, shadow casting and texturing.

Rendering these images in real time (typically < 0.1 sec.) takes considerably more computing resources than offered by present day workstations. The need for such systems, however, is growing in areas such as design, engineering, simulation and animation. For this, a research project at CWI is aimed at the design of a new workstation architecture. For time-critical functions in that architecture new hardware components will be designed. New and faster hardware alone however will not suffice. The interactive workstation concept should be based on an architecture which reflects the balance between carefully designed hardware resources and well-structured logical modules which

CWI Quarterly 3, 179-201

economically make use of these resources. This inherent dualism of graphics systems research will be reflected throughout this paper.

This paper is organized as follows: We first present a simplified overview of the computer graphics image synthesis pipeline and the relevant functions that operate on it. Next, we show how a logical input model interacts with various stages of the image synthesis pipeline. Finally, we show how parallel processing can be applied to implement the logical models so that real-time interaction can be achieved. The emphasis throughout the paper will be on the logical models of image synthesis and interaction.

2. Raster graphics architectures

2.1. Graphics systems

Originally, a raster graphics system merely was a dedicated section of system memory, the so-called frame buffer, connected to a video controller (sometimes called display controller). This basic system is shown in Figure 1. The host CPU had to perform all the processing needed to generate an image. Even on a fast computer however, image generation takes much more time than the persistence time of the phosphor of the screen (typically 10-60 milliseconds). Therefore, resulting colour values of individual picture elements (pixels) are stored in the frame buffer from which a screen refresh process, handled by the video controller, can be initiated.

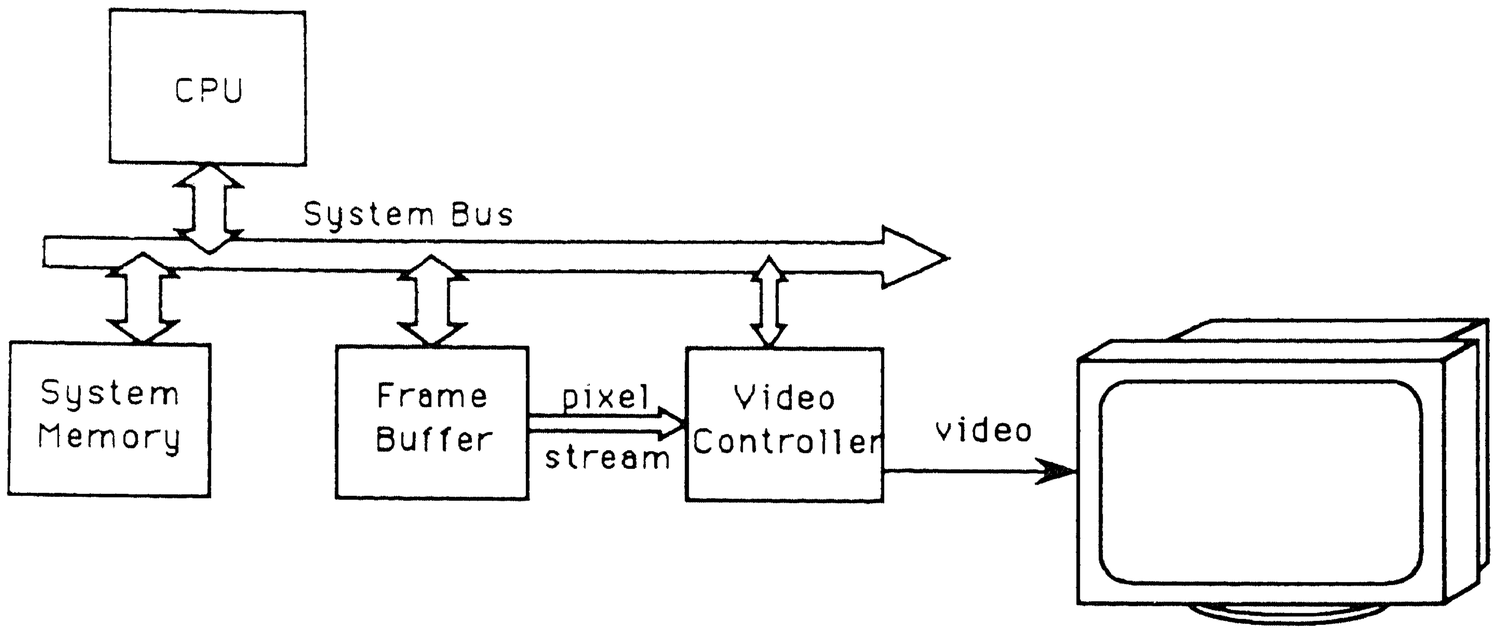

Figure 1. Simple graphics system.
Frame buffer access takes place via the system bus.

At a sufficiently fast rate, typically 60-100 Hz, the video controller reads in the digital pixel data from the frame buffer and generates from that data the analogue video signal. The pixel data varies depending on the kind of system: a monochrome system has one bit per pixel, a full colour system may have up to 36 bits per pixel (12 bits for each of the red, green and blue video signals).

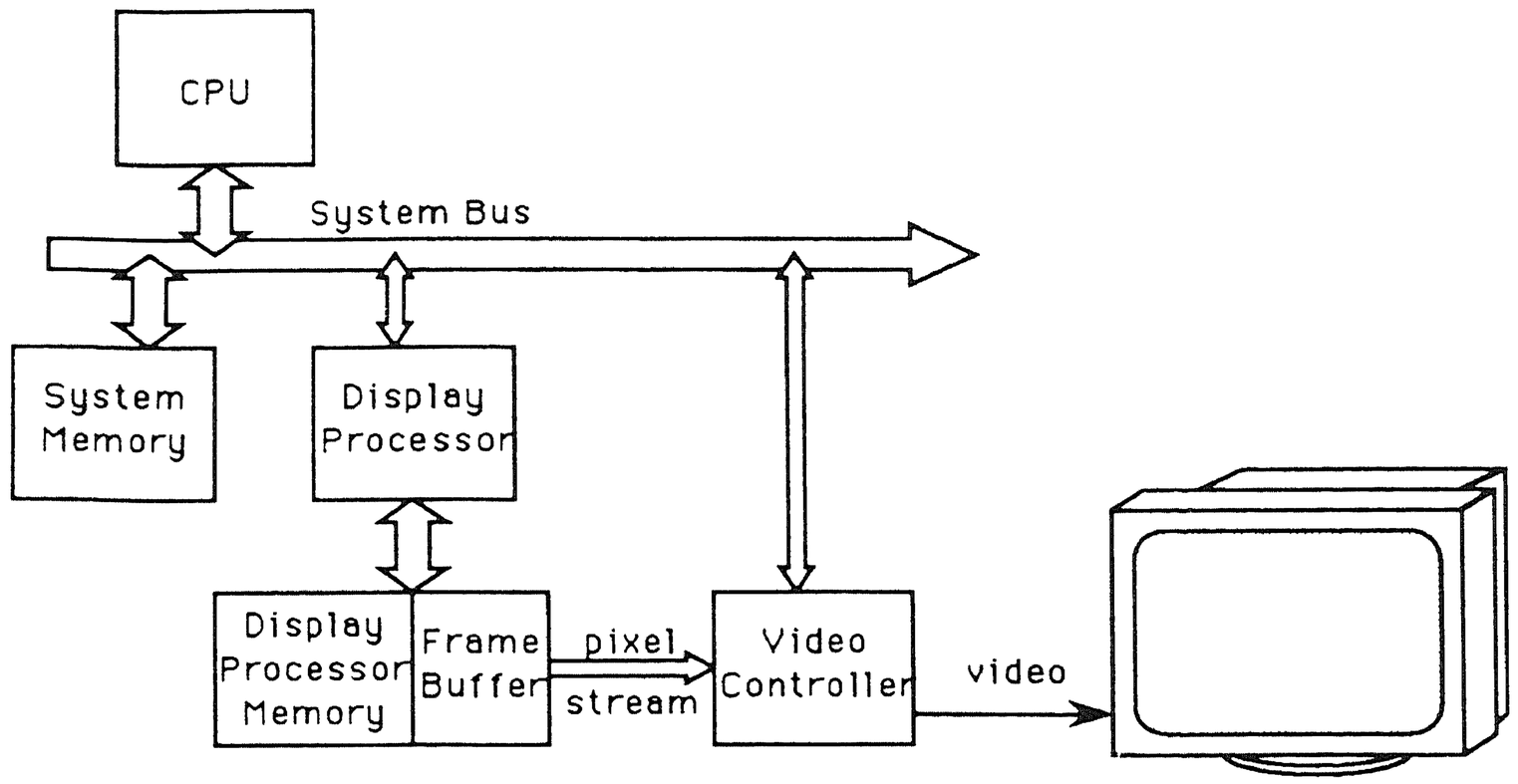

FIGURE 2. Graphics system centered around a display processor. The display processor off-loads the host CPU and reduces system bus contention.

From this elementary architecture, the display processor architecture (see Fig. 2) emerged, triggered by the ever-increasing need for extra performance and the availability of inexpensive microprocessors [1-3]. In this architecture, some of the image generation tasks are handled by a dedicated display processor, thereby off-loading the host CPU. Typical for this architecture is that frame buffer access is handled by the display processor via a private bus, thereby reducing system bus contention. Even for display processor architectures frame storage is still needed. Most of the systems found today adhere to this concept, although we can distinguish a great variety in the level of functionality handled by the display processor. Throughout the years we have seen several 'waves' of moving functionality in the direction of display processors and moving functionality back to the host CPU. These tidal waves on the ocean of computer graphics are caused by changes in hardware technology (e.g. the RISC versus CISC competition) as well as changes in software technology and de facto standards (graphics languages versus pixel based window systems).

2.2. Logical model for image synthesis

A logical model of the image synthesis pipeline is shown in Figure 3. This model [4] distinguishes between the different logical representation levels of objects that exist in the image synthesis pipeline and the operations applicable on these representations.

The application model (AM) is the representation of an object as

determined by the application program. In this representation the application data to be displayed is presented as a model from which specific graphical images can be composed. The application program can change the object representation by editing it. Naming of parts of the representation is allowed and can be used to aid editing.

The structured display file (SDF) contains only graphical information of objects and is composed of a list of output primitives. The graphics system manages the content of the structured display file. The application program does not have direct access to it.

The linear display file (LDF) contains the representation of objects in a form designed for optimal refreshing speed. With raster displays this is the frame buffer, in which an image is represented as a pixel pattern. The display contains the resulting visible representation of the image.

Between each pair of successive representations, there is a logical processor that can map one representation to the other. Each logical processor accepts input parameters that influence the mapping process. These input parameters consist of a previous representation and, optionally, of parameters that are controlled by either the user or the application program.

The display file compiler (DFC) implements the logical process that maps the application model to the structured display file. This mapping process can be parameterized by, for example, surface properties of the objects.

The display processing unit (DPU) implements the logical process that maps the structured display file to the linear display file. Functions that are executed by the display processing unit are geometric transformations, hidden surface removal calculations, illumination calculations and clipping. The display processing unit can be parameterized by, for example, viewing and illumination parameters (cf. E.H. Blake's contribution on 'Shading' in the Appendix). Needless to say, it is this process that is the most time critical of the image synthesis pipeline.

The display controller (DC) implements the logical process that maps the linear display file onto the screen. The display controller can be parameterized so that only portions of the linear display file are mapped on the screen; i.e. parameterized by window management functions.

3. Interaction

The basis of interaction lies in the effectiveness of how a user can operate the image synthesis pipeline. To explain this, we first develop a logical input model which encapsulates all types of physical input devices and possible user actions.

3.1. Logical input devices

A logical model of a graphical input device is shown diagrammatically in Figure 4.

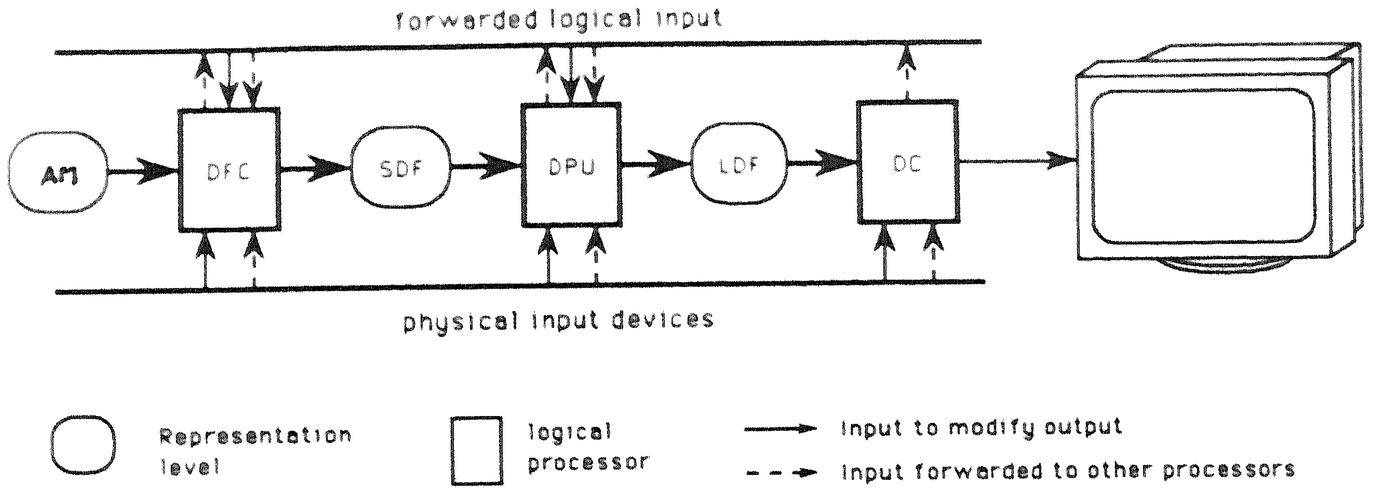

FIGURE 3. Logical model of image generation pipeline. AM - Application Model, DFC - Display File Compiler, SDF - Structured Display File, DPU - Display Processing Unit, LDF - Linear Display File, DC - Display Controller.

3.1.1. Overview. The output of a logical input device is an input primitive. Different classes of logical input devices exist corresponding to different types of input primitives. Each class has a well-defined data type for the measure reported by devices of that class. For example, current computer graphics standards support the following basic logical input device classes: locator, stroke, valuator, choice, pick, string [5].

The input primitive produced by the logical device is the measure of the device at the moment a trigger fires. The operating mode of the device determines when an input primitive is delivered to the user of the device.

An input device logically comprises measure, echo, prompt, trigger, and acknowledgement processes overseen by a single control process.

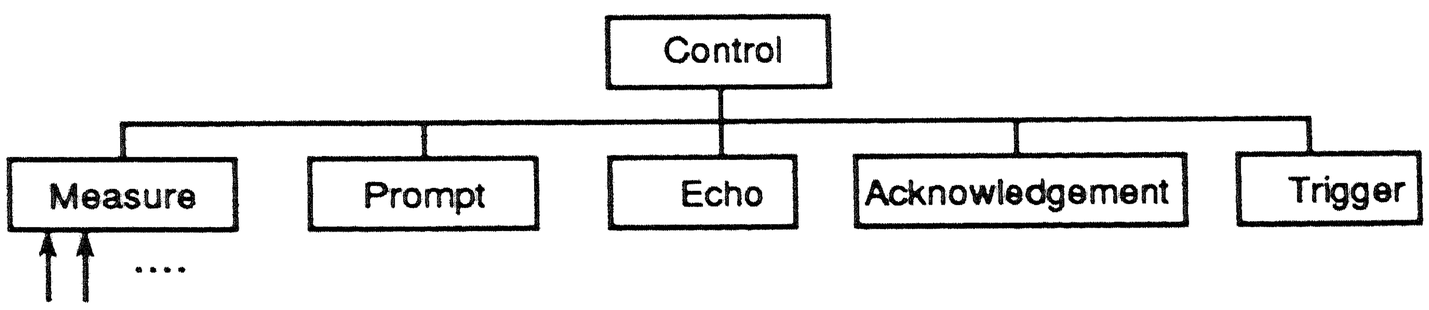

FIGURE 4. Logical model of a graphical input device.

Each of the component processes has certain responsibilities which, when acting in concert, can result in the production of an input primitive derived from manipulations of the input tool by an actuating agent:

- *Measure process.* When the measure process is activated, it continuously maintains a measure value derived from the inputs to the process. The process is responsible for performing all computations necessary to map the inputs to the measure value. The measure may then be accessed by the echo and control processes as necessary.
- *Echo process.* When the echo process is activated, it provides continual feedback of the current value of the measure. This feedback is expressed in terms of the output facilities available at the level of abstraction at which the logical input device is defined. The echo process may also use output storage facilities maintained at this level of abstraction.
- *Prompt process.* The prompt process is responsible for indicating the availability of its associated logical input device. The prompt appears when the prompt process is activated. The prompt is used both as an indication and to provide instructions on the use of the logical input device.
- *Trigger process.* While a trigger process is active, it continuously monitors one or more sets of conditions. When any one of these sets of conditions is satisfied, the trigger is fired. The trigger firing is reported to the control process.
- *Control process.* The control process is responsible for the overall operation of the logical input device and its associated component processes. When the logical input device is activated, the control process activates the associated measure, echo, prompt, and trigger processes. These processes then function independently. When the trigger firing report is received by the control process, it produces an appropriate input primitive that may contain the measure which was current at the time of the trigger firing. The input primitive is then reported to the agent that is using the logical input device.
- *Acknowledgement process.* The acknowledgement process is activated upon acceptance or rejection of the input primitive by the using agent. The acknowledgement process provides feedback according to whether the input report was accepted or rejected. Either one, but not both, of an acceptance acknowledgement or a rejection (negative) acknowledgement can be a null feedback.

3.1.2. Operations on logical input devices. There are seven types of operations which may be performed on logical input devices and their component processes. These are described below:

- *Definition.* The definition operations form logical input devices as collections of component processes. The component processes themselves may either be predefined or provided by the agent that is defining the logical input devices. Typically, a realization of this model will constrain the control process to be predefined. However, this is not an intrinsic requirement of the model.
- *Initialization.* Initialization operations prepare the logical input device for activation by defining the initial state to be used by each of the component processes. These initial values are applied at each activation of the

logical input device. At this time static echo and prompt objects may be created for display at activation.

- *Activation*. When a logical input device is activated, the states of the component processes are set to their initial values and the component processes are activated. At this time, the logical input device may be utilized (see below).
- *Utilization*. After a logical input device has been activated, its various component processes execute independently under the coordination of the control process. At this time, any input tool associated with the logical input device may be manipulated by the operator to modify the input values from which the measure is constructed. Feedback of the current value of the measure is produced by the echo process. Factors involved in the evaluation of the sets of conditions associated with trigger firings are monitored by the trigger process. As appropriate, trigger firing reports are provided by the trigger process to the control process. As trigger reports are received by the control process, input primitives are produced and delivered to the using agent. Validation of acceptance or rejection of the input primitives is feedback to the operator by momentary activation of the acknowledgement process.
- *Deactivation*. When the control process determines that the logical input device should no longer be available (usually by direction of the using agent), the component processes are terminated. This will result in elimination of the echo and the removal of the state of the logical input device. Manipulation of any associated input tools will no longer have any effect on this logical input device.
- *Termination*. At termination of a logical input device, resources allocated to the logical input device are released. Such resources include the initial state values and any precomputed echo and prompt objects.
- *Undefinition*. When a logical input device is undefined, all knowledge of the composition of the logical input device is removed from the system. The identifier of the logical input device may then be reused for another definition.

3.1.3. Operating modes. A logical input device may operate in any one of several operating modes. An operating mode determines the behavior of the device as seen by the operator of the device and the agent using the device. Examples of operating modes commonly supported include the conventional request, sample, and event modes. However, these are not intended to be the only possible operating modes. Support for other modes can be defined so long as they operate within the confines of the model.

3.1.4. Relationship between input and output. Logical input devices may need to make use of output facilities in order to provide prompting, echoing, and acknowledgement. The input model does not contain special output functionality for this purpose. Instead the model uses the output functionality provided at the same level of abstraction (see section 2.2) at which the logical input device

is defined. The using agent of a logical input device may be the output environment at its level of abstraction. For instance, a logical input device may directly control components of the state of this environment rather than through a path involving the transfer of input primitives to a higher level of abstraction followed by an invocation of an output function which causes appropriate changes at the lower level. An example of this is a logical input device which provides a transformation matrix used directly by the output environment, or which provides a colour representation that is used directly in a colour table.

The model also allows the output environment to be either a part of the state of the component processes of a logical input device, or input to a process. An example of this is a pick logical input device for which the input data are a location in the coordinate space at a certain level of abstraction and the current collection of graphical information at that level.

This balanced relationship between input and output is called input/output symmetry. The diagram below illustrates this relationship. The logical input device (box on the left) gets its input from its using agent. It can pass on the input through to a higher level, or invoke a function $f(input)$ which causes changes in an output environment (box on the right). Similarly, the output environment can invoke the function which causes a change of state in the logical input device.

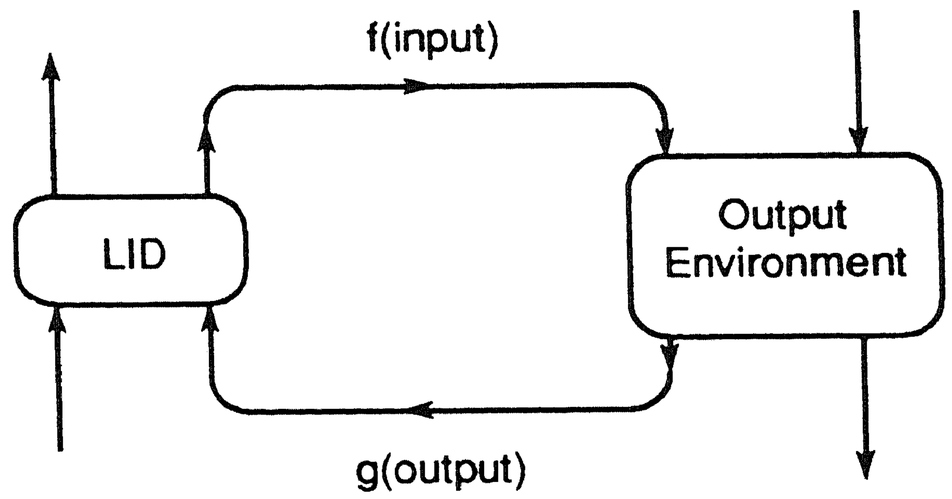

FIGURE 5. Balanced relationship between input and output.

3.2. Display files

From the logical model one can see that appropriate structuring of the display files can aid the logical processes to traverse and/or edit these files. On one hand the structure of a display file should support high level editing, for instance, to facilitate dragging of compound structures. On the other hand, a low level structuring seems to be necessary for the rapid redrawing of low level editing such as pick operations.

Display file structure. To achieve interactive display of images, a well-tured display file must have the following properties:
Performance. Performance results from fast execution of each primitive in the display file. The algorithms that implement each logical process must emphasize the execution speed. The performance of the system will be determined by the slowest algorithm. Graphics algorithms can be imple-mented either in software or in silicon.
Structuring. The primitives are structured into groups to aid editing and traversing of the display file. In the next section we list a number of typi-cal editing operations on the structured and logical display files. The point here is that an appropriate structuring of the display file will aid the logi-cal process of regenerating a display file. It will be this regeneration speed which determines how the display file should be structured.

. *Editing display files.* Two basic notions related to editing of structured ay files are those of incremental and local changes.
eally, algorithms within the display processing unit should be able to make ges incrementally. By incremental changes we mean changes that are rela-to an already existing value. In this way, an editing operation can be esented by updating a value in the structured display file. The change s effect by first letting the change propagate through the structured display nd then generate from this a new logical display file.
gorithms should, when possible, support the notion of locality. This ns that a change will affect a relatively small portion of the display file. If corresponding part in the display file can be easily identified, it is possible inimize both the update efforts in the structured display file, as well as the ration of the corresponding logical display file. Structuring should be par-arly helpful in supporting this type of changes.
ypical editing operations on the structured and logical display files that be supported include:
Geometrical transformations. The geometry of an object is represented in the structured display file by a group of primitives. A particularly interest-ing editing operation is the transformation of the object. The structured display file must have a level where a complete group of primitives can be addressed and manipulated. A simple example of a geometrical transfor-mation would be the dragging of an object. With an appropriate structure in the structured display file this editing operation would need to change only one value.
Colour control. Little attention has been given to provide firmware for dynamic control of colours. Fast regeneration should include colour func-tion 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 reflection and transparency can be realized by colour function compositions.
Appearance control. The attributes that are traditionally used for creating dynamic effects, such as highlighting, are primarily used 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. Appearance control introduces no further requirements beyond those already encountered for transformations.

- *Insert and delete.* Insertion and deletion of objects in the display file require local rearrangements for proper hidden surface removal and lighting effects. Such rearrangements must be calculated quickly, followed by a re-execution of the hidden surface removal algorithm.
- *Picking manipulations.* Picking an output primitive is an editing operation by the user which essentially involves a search of the display file. Usually, the graphics system will provide some form of feedback to identify the picked primitive by, for example, high-lighting. Picking and feedback will be best served by functionality that identifies the relevant elements in real-time. Low level feedback will assist during picking. Higher level feedback, indicating restored consistency after an input completion, will use upward references realized either by an explicit administration or by fast searching traversal.

3.3. An interaction based architecture

The aspects of interactions as listed above, determine the structuring of a display file, a data structure in which information needed to generate an image is stored. An inventory of manipulations which can be expected to occur by user interactions showed us that all interactions are initiated based on geometrical aspects. We concluded [6,7] that apart from the application model, three geometry based levels of image representation must be accessible for interaction purposes. These levels are: visible parts only, graphical objects or compound graphical objects, and the scene as a whole.

Based on this notion, we arrived at the logical architecture shown in Figure 6, in which the application model is progressively processed in a number of steps, each providing facilities for interactive modification. The availability of intermediate representation levels allows changes to be initiated on the lowest level possible. It also supports incremental changes affecting a mere subset of the image information, thereby reducing the amount of computations needed and thus improves system performance. Note that based on this viewpoint pixels can be considered as artefacts of the raster technology. Interaction support as such does not justify a pixel representation of the image. For this reason, in this architecture, the refresh process is at least conceptually initiated from a domain representation level. Given the real-time requirements of the refresh process, it is clear that a display controller that actually operates from this level is quite resource intensive. On the other hand it will support the fastest possible response on interactions.

Worth noting is that colour evaluation can take place at different levels in this architecture. In this way, different shading methods and evaluation strategies can be incorporated in the same architecture.

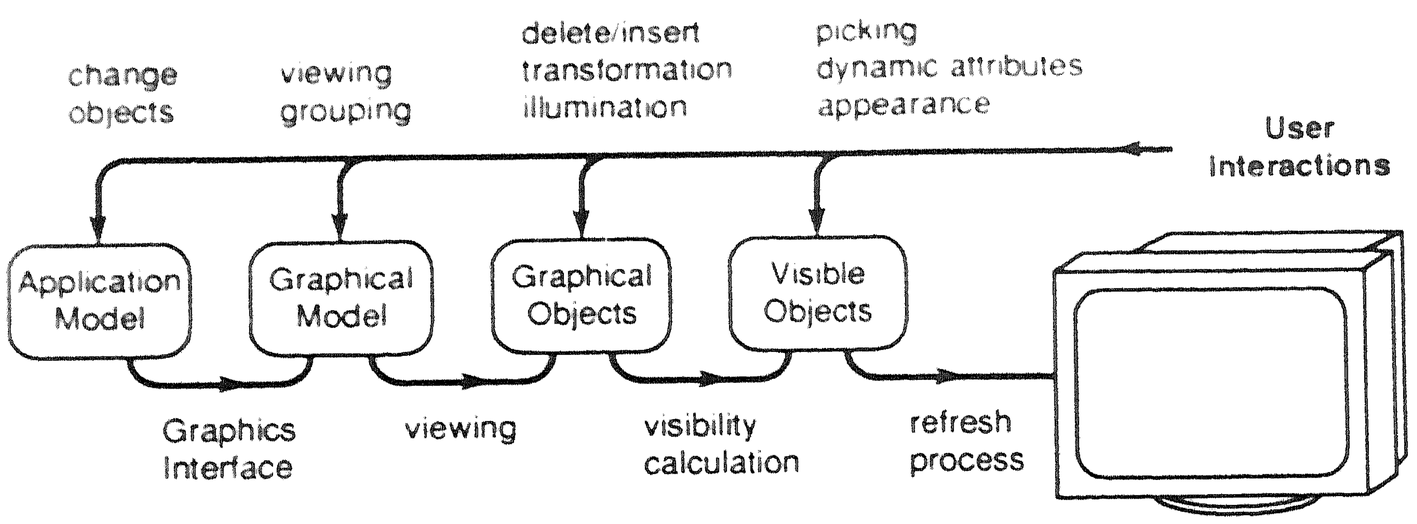

FIGURE 6. An interaction supporting architecture. User interactions can be initiated from intermediate representation levels.

4. PARALLEL PROCESSING

The logical architecture presented in the previous section provides the structuring for efficient interaction support. In spite of this, some of the processing steps involved require considerable computing resources to guarantee sufficiently fast response. Processor technology developments have resulted in a remarkable improvement of computing power. But, to satisfy the need for higher image quality, scene complexity and interactivity, experts in the field estimate that six orders of magnitude more processing power than available in present day processors is necessary. It is clear that this cannot be accomplished by improvement in processor technology and efficient structuring alone. An interactive system will have to be highly parallel. Before we discuss our investigations in this area we will first focus on how parallel systems can be classified.

4.1. Classification

When introducing parallelism the two basic aspects that must be considered are: 'how can the problem be distributed?' and: 'what kind of hardware is appropriate?'. Depending on which of these two aspects is of most concern, we can classify parallelism based on architectures or based on distribution.

4.1.1. Architecture based classification. In computer graphics systems, we can distinguish the following types of parallel architectures: pipeline, MIMD, systolic array and SIMD.

- *Pipeline.* In this class of architectures, data is passed from one processor to the next, each processor performing a specific task. For this type of systems load balancing and capability of handling large data streams are essential. Data dependent processing reduces the efficiency of these systems.
- *MIMD.* A multiple instruction, multiple data system is a set of asynchronous operating processors. Synchronisation is handled by message passing.

Job scheduling and communication are the most critical aspects. Scalability and configurability made this type of system popular. On the other hand, this architecture is costly and can make debugging tedious.

- *Systolic array.* A systolic array processor is closely related to the pipeline architecture. In general however it is a multi-dimensional structure handling multiple data streams. The processing elements handle instruction level tasks only and are interconnected in a problem specific way. An array as a whole should be considered as one dedicated processor rather than a flexible programmable system.
- *SIMD.* A single instruction, multiple data system (also known as vector-processor) is a set of synchronous operating processors: all execute the same instruction in lock-step. Its centralized control makes it well suited for massive parallelism. The efficiency may reduce dramatically when conditional instructions are involved.

Apart from the classification types mentioned here we can distinguish several levels of granularity. Especially for MIMD architectures, we encounter systems involving mainframes, workstations, loosely coupled multi-processors, tightly coupled multi-processors, down to instruction set level multi-processors.

- *Mainframes.* Processing can be off-loaded to a mainframe, a vector-processor, a supercomputer or even a cluster of supercomputers. These types of number crunching facilities can usually be accessed via a wide area network. This type of approach is typical for non-interactive computer intensive preprocessing tasks and has little relevance for this discussion.
- *Workstations.* A processing task can be distributed among a cluster of workstations connected via a local area network. Normal operating system facilities can be used to start remote procedures. The cluster may be very heterogeneous.
- *Loosely coupled multi-processors.* This term refers to message passing systems. Each processor has its private local memory. Data exchange and synchronisation is handled by sending messages via multiple links. Communication overhead can be considerable.
- *Tightly coupled multi-processors.* If all processors of a system share a piece of memory, they are said to be tightly coupled. Data can be shared and synchronisation can be handled by centrally administered status information. Memory contention puts an upper limit on the number of processors that can efficiently be coupled.
- *Instruction set level multi-processors.* In this category we find dataflow processors, neural networks and the like. These architectures execute machine instructions in parallel. The programming of these type of architectures differs much more from programming a sequential machine than any of the other granularity levels above.

4.1.2. Distribution based classification. From the algorithmic point of view the following task subdivision strategies can be recognized: image space, object space, and functional subdivision.

- *Image space.* In this rather popular strategy each processor operates on one or more pixels of the image. The maximum level of parallelism is determined by the image resolution, a constant factor. Efficiency of such systems may be low because image complexity is usually not homogeneous. Furthermore, geometric data must be distributed to all processors.
- *Object space.* Processors are allocated to process one or more graphical objects (e.g. lines, areas, characters). The number of objects in a scene which should relate to the level of parallelism is a dynamic quantity. Pixels generated by each processor must be collected and compared with results from other processors to determine their visibility. The number of pixels each processor will generate is data dependent.
- *Functional.* Processors are allocated for one of the tasks found in the image generation pipeline (geometric transformation, clipping, shading etc.). The tasks vary in complexity and some of the tasks are data dependent (e.g. clipping) which complicates load balancing.

For parallel algorithms we also find the concept of granularity and recognize the classes coarse, medium and fine grain parallelism. This classification is based on the number of instructions executed between two points of synchronisation. There is no real concensus on the exact quantification.

- *Coarse grain.* The tasks are executed with hardly any information exchange. Typically the number of instructions is over a thousand.
- *Medium grain.* Tasks executed are in the order of tens to hundreds of instructions. Communication is substantial.
- *Fine grain.* Tasks are up to several instructions long, the communication rate is very high.

4.2. Parallel graphics

Graphics algorithms are inherently parallel. In increasing order of granularity, the computing tasks can be organized based on pixels, vertices, polygons, patches, objects or frames. Furthermore, the image generation pipeline consists of a number of clearly separable tasks. As a result, numerous solutions can be considered to map graphics algorithms on a hardware configuration. For some specific problems, such as geometric transformation, a satisfactorily mapping on parallel hardware has been found [8], but a configuration optimal for the entire graphics pipeline has not been found yet. Each of the solutions proposed so far has its specific characteristics and is well suited for a particular type of images and a particular type of rendering only.

In computer graphics, however, we must deal with more than just one type of images and more than just one type of rendering. We find a great variety in the internal characteristics of images, such as shading techniques [9,10], hidden surface removal algorithms [11,12] and graphics primitives [13-15]. Also, the characteristics of applications vary considerably, and in an interactive environment, even momentarily.

For this reason, research in the department of interactive systems has been concentrated on a part of raster graphics hardware that is common to all systems, i.e., the rasterization step, and on adaptable graphics environments. These topics will be discussed in the next two sections.

4.3. A domain level display controller

Technically the most demanding element of the interaction based architecture presented, is the display controller. As mentioned above, in this architecture the input of the refresh process is a domain-level rather than a pixel-level representation of the image. As a result, *all* pixel related operations must be handled by the display controller at a rate of about 60 Hz. These pixel related operations include scan-conversion, shading and anti-aliasing of the graphical primitives (see plate number I). We believe that, due to this, a well partitioned multi-processor system is a prerequisite. The system should be flexible enough to be able to support different kinds of shading techniques.

The way in which images are scanned (a sequence of horizontal scanlines) already leads to a subdivision of the refresh process in vertical (y) and in horizontal (x) directions. This phenomenon reflects in the two distinct types of processors (x- and y-processors) that can be found in the display controller architecture shown in Figure 7 [16].

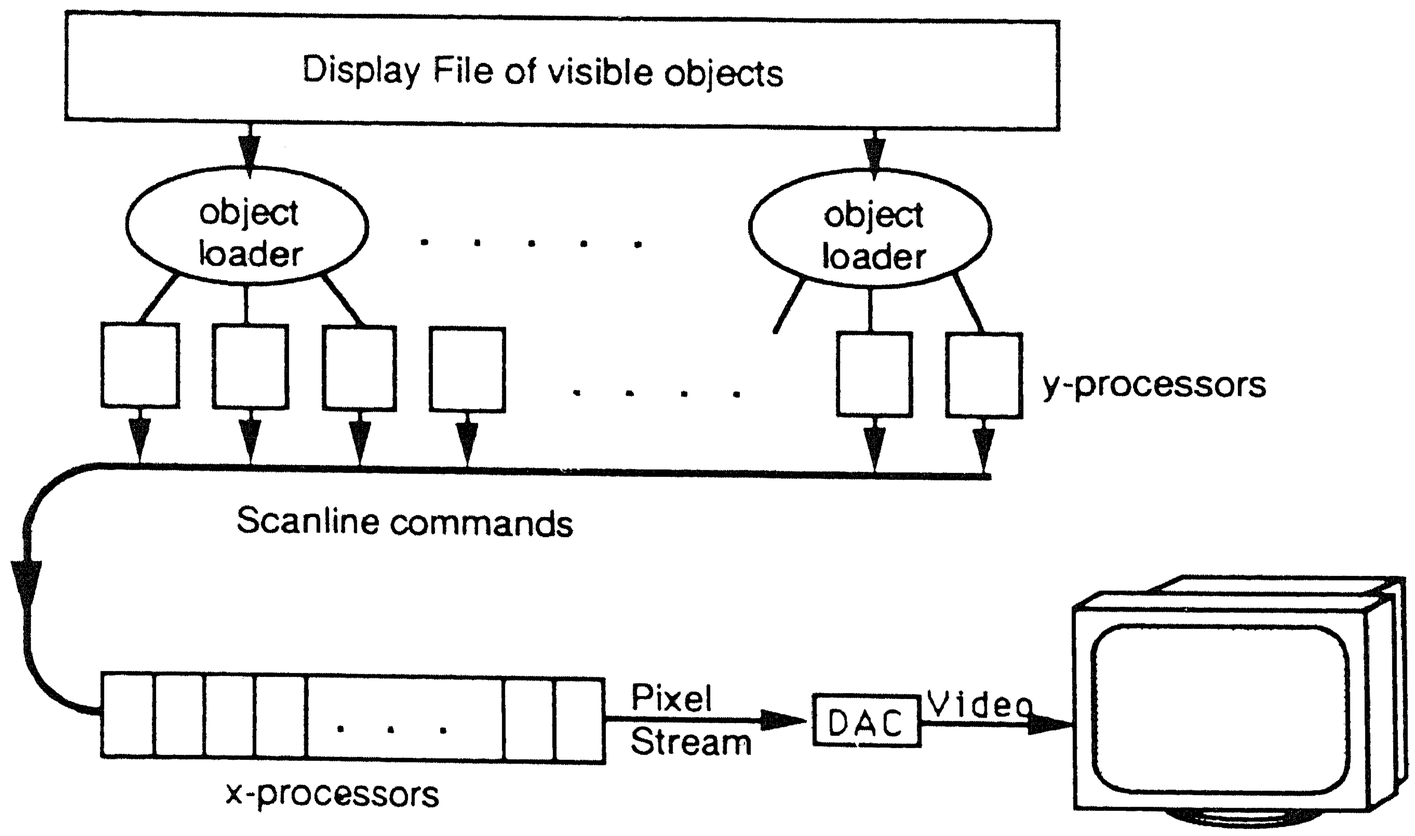

FIGURE 7. Schematics of the display controller.

Most primitives contribute to a limited number of horizontal scanlines only, i.e. they become 'active' during a short period of the refresh process. The display file from which the display controller operates is organized such that primitives that become active can be easily found. These active primitives are

presented to the *y*-processors by a so-called object loader. The *y*-processors scan these active primitives, thereby generating scanline commands containing shading information of the intersection of the current scanline with the primitive. These commands in turn are sorted and converted to a pixel stream (of about 90 MHz, i.e. 11ns/pixel) by a systolic array of *x*-processors. Finally, this stream of digital pixel intensities is converted into an analogue video signal.

The result is a hybrid architecture in that it exploits image space subdivision (each vertical pixel column is assigned to one *x*-processor from the array), as well as object space subdivision (each *y*-processor operates on a different active primitive).

The display controller can be scaled according to the needs. The number of *x*-processors can be adapted to obtain the desired resolution of the system, whereas the number of *y*-processors can be adapted to the desired maximum complexity of images to be displayed. The instruction repertoire of the processor elements provides the flexibility desired for different types of shading (see also plates II, III, IV, V and E.H. Blake's contribution on 'Shading' in the Appendix).

For implementation of the display controller, full-custom VLSI chips were needed. A photograph of the *x*-processor array chip, designed by our colleagues at the University of Twente, can be found in this issue (plate number VI).

4.4. Adaptable parallelism by means of BONSAI

Parallelism can also be applied to higher levels of the image synthesis pipeline. We have designed and implemented a prototype system, called BONSAI, some key principles of which we will discuss here.

Two major shortcomings of current day graphics systems are related to (1) the static nature of the application interface and (2) the static nature of the graphics pipeline semantics. We illustrate each issue by an example.

- *Static application interfaces.* Current graphics systems provide application program with only a fixed collection of primitives, attributes, and storage schemes. This, in general, leads to tedious specifications for those applications that use primitives which do not fall in a system's fixed collection. A lengthy decomposition process must be done by the application in order to use the primitives available in a graphics package. A more promising approach seems to be to allow the application to augment the collection of primitives and attributes or even define its own storage scheme. By importing new primitive definitions into a graphics package, the application can be kept simple.
- *Static pipeline semantics.* All graphics systems define a pipeline through which a primitive passes when it is rendered. Attribute binding and storage of intermediate representations are fixed by the semantics of the pipeline. This approach is awkward for those applications that need to use other storage schemes to store graphics data. Allowing the application to define its own storage scheme will greatly enhance interaction since only the application can have the knowledge of how data retrieval can be done

efficiently. A second advantage of a generic pipeline definition is that it allows the application to provide the graphics system with a specialized hardware configuration.

BONSAI addresses these two shortcomings by providing support for what we believe will have a fundamental impact on the definition of new graphics software architectures. We refer to these issues as (1) providing support for extensible application interfaces and (2) providing support for configurable pipelines.

1) Extensible application interfaces will allow an application to define and import new primitives, attributes and storage schemes into BONSAI. Libraries of predefined primitives and attributes will be provided for general use. An application can specialize on or augment to a library primitive.
2) Configurable pipelines will allow the application to exploit novel hardware architectures by placing those components in the pipeline where they are needed the most [17]. For example, BONSAI should be able to exploit all possibilities of parallel components by allowing the application to define a pipeline that can exploit this parallelism. A second example is to allow the application to configure the pipeline to exploit remote computing services.

Extensible interfaces can be realized in two steps. First, BONSAI must provide suitable abstractions for basic concepts such as primitives, attributes, and storage. The second step allows the application, by means of object-oriented techniques, to specialize on these abstractions. In this way applications can define new abstractions in terms of the more general ones known to BONSAI. Object-oriented techniques can be used to specialize in various storage schemes as well. We find all the necessary object-oriented techniques in the programming language C++, so no special purpose specification language is necessary.

4.4.1. The internals of BONSAI. The definition of the BONSAI system is based on a new approach to the development of systems, called the Component/Framework Process. Inherent in this process is a model of graphics systems which sees a graphics system as constructed from a collection of components set in a framework. Components include datatypes and operations. A framework is the glue which joins components together to form a system and manages the display and control. This idea is illustrated in Figure 8. Systems A, B and C each have their own frameworks. Some components are used by more than one system, others by only one. The relationship between the PHIGS standard [18] and the PHIGS+ proposal [15] illustrates this idea in that PHIGS and PHIGS+ share a common framework, but differ in their choice of output primitive components and attribute components. PHIGS+ uses a richer set of components which take illumination into account. In contrast however, PHIGS and PHIGS+ use the same input components.

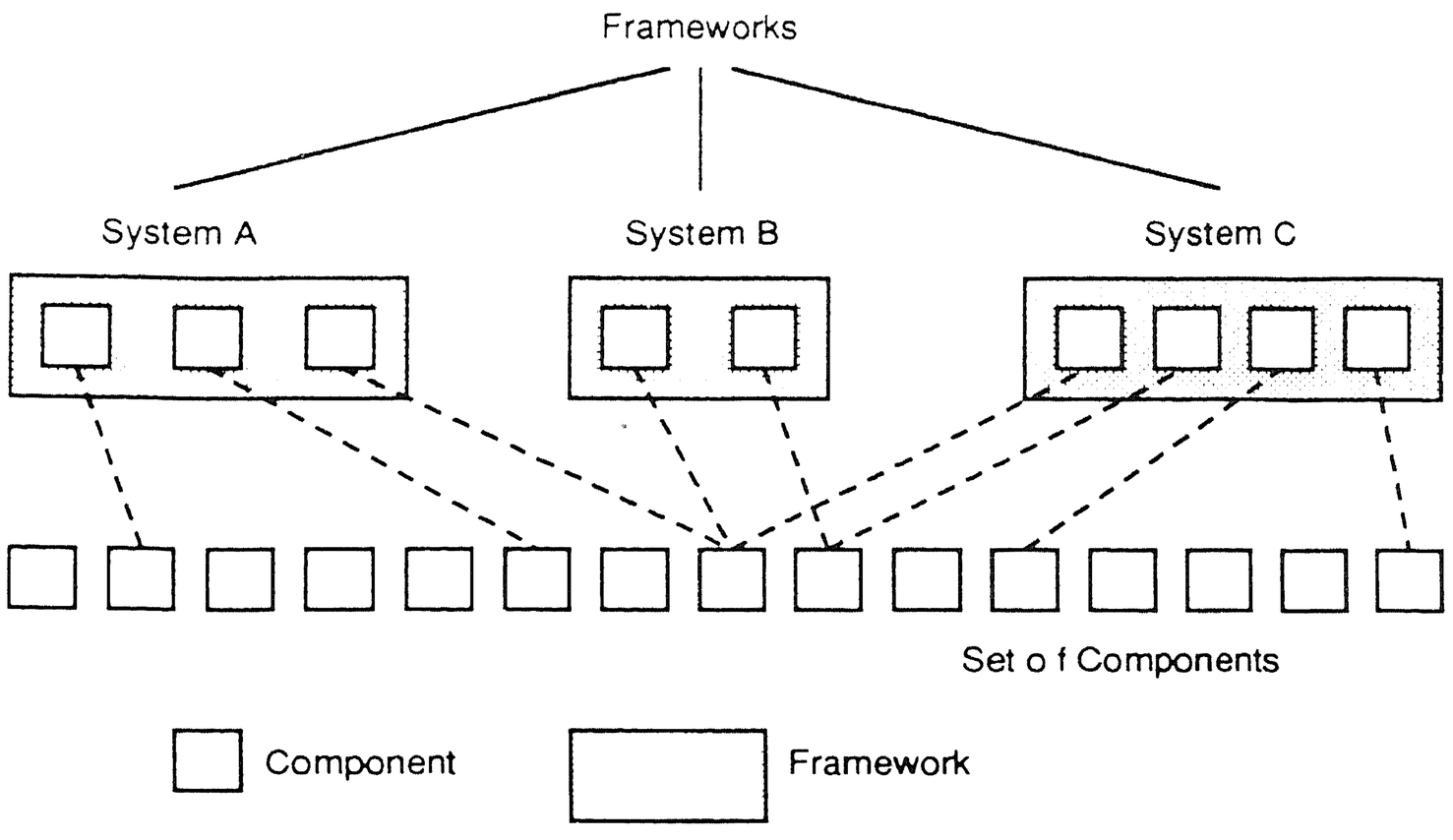

FIGURE 8. Component/Framework Process: Graphics systems can be constructed from a collection of components set in a framework.

The realization of a graphics pipeline can conceptually be partitioned into three tiers: tier one is called the shell; tier two is BONSAI; tier three is called the virtual workstation.

- *Tier-One: Shell.* The shell implements the high level functional interface as defined by some standard. Typical examples of shells are GKS, PHIGS, PHIGS+ and RenderMan [15]. The shell will use all functionality that is available from the BONSAI system.
- *Tier-Two: BONSAI.* BONSAI is the intermediate level that is used by high level graphics packages as a veneer to the virtual workstation. It is the shell's own responsibility to map high-level representations onto representations that are suitable for BONSAI.
- *Tier-Three: virtual workstation.* The virtual workstation level is an abstraction of a particular hardware configuration. Both the shell and BONSAI can parameterize the virtual workstation in order to take advantage of the underlying hardware capabilities. A typical example of a device dependent parameterization is how an area primitive is decomposed. On some hardware configurations an area primitive must be decomposed into a mesh, whereas on others an area primitive will be decomposed into triangular strips. The virtual workstation level can be parameterized to be either a mesh machine or a triangular strip machine.

5. Conclusion

In this paper we presented a novel architecture for interactive raster graphics. The architecture reflects the balance between carefully designed hardware and well-structured high level logical modules which make effective use of the hardware. We explained the importance of parallel systems and how they can be applied to implement the described architecture.

References

1. J. Foley, A. van Dam (1982). *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Mass.
2. W.M. Newman, R.F. Sproull (1979). *Principles of Interactive Computer Graphics*, McGraw-Hill, New York.
3. K. Guttag, T. van Aken, M. Asal (1986). Requirements for a VLSI graphics processor. *IEEE Computer Graphics and Applications 6 (1)*, 32-47.
4. I. Carlbom (1980). *System Architecture for High-Performance Vector Graphics*, Ph.D. Thesis, Dept. of Computer Science, Brown University, Providence.
5. D. Rosenthal, J. C. Michener, G. Pfaff, R. Kessener, M. Sabin (1982). The detailed semantics of graphics input devices. *Computer Graphics 16 (3)*, 33-38.
6. P.J.W. ten Hagen, A.A.M. Kuijk, C.G. Trienekens (1987). Display architecture for VLSI-based graphics workstations. W. Strasser (ed.). *Advances in Graphics Hardware I*, Eurographic Seminars, Springer-Verlag, Berlin.
7. V. Akman, P.J.W. ten Hagen, A.A.M. Kuijk (1988). A vector-like architecture for raster graphics. A.A.M. Kuijk, W. Strasser (ed.). *Advances in Graphics Hardware II*, Eurographic Seminars, Springer-Verlag, Berlin.
8. J.H. Clark (1982). A VLSI geometry processor for graphics. *Computer Graphics 16 (3)*, 127-133.
9. B.T. Phong (1975). Illumination for computer generated images. *Communications of the ACM 18 (6)*, 311-317.
10. R. Hall (1988). *Illumination and Color in Computer Generated Imagery*, Monographs in Visual Communication, Springer-Verlag, Berlin.
11. A.A.M. Kuijk, P.J.W. ten Hagen, V. Akman (1988). An exact incremental hidden surface algorithm. A.A.M. Kuijk, W. Strasser (ed.). *Advances in Graphics Hardware II*, Eurographic Seminars, Springer-Verlag, Berlin.
12. E. Fiume, A. Fournier (1988). The visible surface problem under abstract graphic models. R.A. Earnshaw (ed.). *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI Series, Springer-Verlag, Heidelberg.
13. F.R.A. Hopgood, D.A. Bruce, J.R. Gallop, D.C. Sutcliffe (1983). *Introduction to the Graphical Kernel System (GKS)*, Academic Press.

14. ISO (1985). Information processing - graphical kernel system - functional description. *International Standard DIS 7942.*
15. A. v. DAM (1988). PHIGS+ functional description, revision 3.0. *ACM Computer Graphics 22 (3).*
16. J.A.K.S. JAYASINGHE, A.A.M. KUIJK, L. SPAANENBURG (1991). A display controller for an object-level frame store system. A.A.M. KUIJK (ed.). *Advances in Graphics Hardware III,* Eurographic Seminars, Springer-Verlag, Berlin.
17. D.B. ARNOLD, D.A. DUCE, G.J. REYNOLDS (1987). An approach to the formal specification of configurable methods of graphics systems. G. MARECHAL (ed.). *European Computer Graphics Conference and Exhibition,* North-Holland, Amsterdam, 439-463.
18. ISO (1989). The programmer's hierarchical interactive graphics system (PHIGS), *ISO 9592 (3 parts).*