Architectures for Human-Computer Communication

A.A.M. Kuijk

1. Introduction

Human-Computer Communication deals with efficient transfer of information between humans and computers and with information structures that tie in with human conceptual abilities. The human visual system is a most powerful image cognition machine that is able to perceive, analyze, classify and evaluate a great deal of information in real-time. Therefore, interactive computer graphics can significantly enhance our ability to understand data, to perceive trends, and to visualize real or imaginary objects.

The design and development of commercially available graphics systems has primarily been driven by the availability of cost-effective hardware technology. A more proper strategy would also take into account user requirements: i.e., interaction mechanisms provided by graphics user interfaces influence the design of the underlying graphics system architecture. The design of a graphics system that strictly adheres to this concept leads to several challenging research issues.

We shall briefly describe the basics and research issues of interactive computer graphics. Next we will outline the research activities carried out at CWI.

2. Graphics and interaction

A graphics system provides facilities to

- specify (or model) a scene in terms of a set of logical graphical elements;

- specify parameters according to which the scene is transformed into an image;
- handle graphical input.

The scene is mapped onto the display- or image-space. This mapping (also known as rendering) includes both geometric and attribute transformations. The result, a set of display primitives which describe an image, is stored in a refresh buffer. The type of display primitives in which the image is described depends on the type of display technology in play (e.g., a set of pixels for a raster display or a set of line drawing instructions for a vector display).

A logical model of the graphics system envisions the mapping of a scene onto the display space as a stepwise process in which primitives travel through a pipeline of functional modules (see figure 1). This model distinguishes several logical representation levels of the scene that exist in the image synthesis pipeline and the operations applicable on these representations. Each module of the pipeline performs an elementary graphical operation on all passing primitives. Graphical input originates from data that comes in via physical input devices. The raw, device dependent input data, are converted into a set of input primitives. This conversion is handled completely within a logical input model so that all types of physical input devices and associated user actions are encapsulated within this logical model.

Actions involved in generating graphical feedback upon user input include:

- handling of the logical input data;
- handling of the input primitives;
- possible updates of some application data;
- traversing (part of) the graphics pipeline;
- update of the refresh buffer.

Response on user input is a key determining factor of the quality of the user interface. A system's response time depends on several factors. Obvious factors are the amount of processing involved in the feedback-loop and the raw performance of the computing resources. Other factors are operating system related dependencies such as interrupt handling, context switching and the like.

3. Research issues

Effective interactive computer graphics applications require considerable computing resources to guarantee a sufficiently fast response. In the last decade, we have witnessed a remarkable improvement of computing power due to improved processor technologies. This evolution can be expected

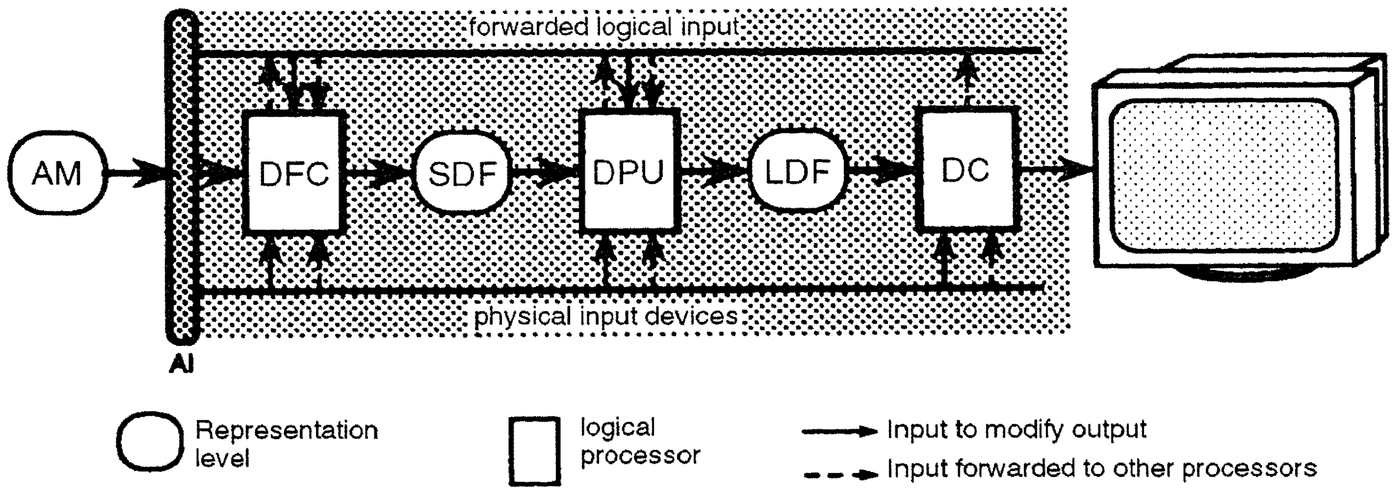

Figure 1. Logical model of image generation.

to continue. However, to satisfy the need for higher image quality, scene complexity and interactivity, experts in the field estimate that four to five orders of magnitude more processing power than available in present day processors is necessary. Such a gap can only be bridged by making use of highly parallel multiprocessor systems.

The inherent nature of graphics algorithms makes that they are well suited to be implemented on multiprocessor systems. In increasing levels of complexity, 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 graphics algorithms can be mapped onto numerous multiprocessor configuration alternatives.

Many solutions have been proposed to distribute the work involved in fast generation of high quality images across multiprocessor systems [1]. The image generation process itself is a pipeline of several sequential processes which automatically suggests functional subdivision (see figure 2 on the left). Subdivision of the image generation pipeline in smaller tasks can be done up to a limited number of steps only, so that the maximum degree of multiprocessing by means of pipelining only is limited. Also data dependent operations make it hard to balance the load for a purely pipelined architecture. Image-space partitioning, i.e., subdivision of the image in small parts that are handled by separate subsystems (see figure 2 in the middle), implies that all primitives have to be processed by all processors (i.e., the system is object-serial). Hence the throughput is limited by the processor speed. Object-space partitioning, i.e., each object is handled by one of the processors of the system (see figure 2 on the right), results in multiple pixel

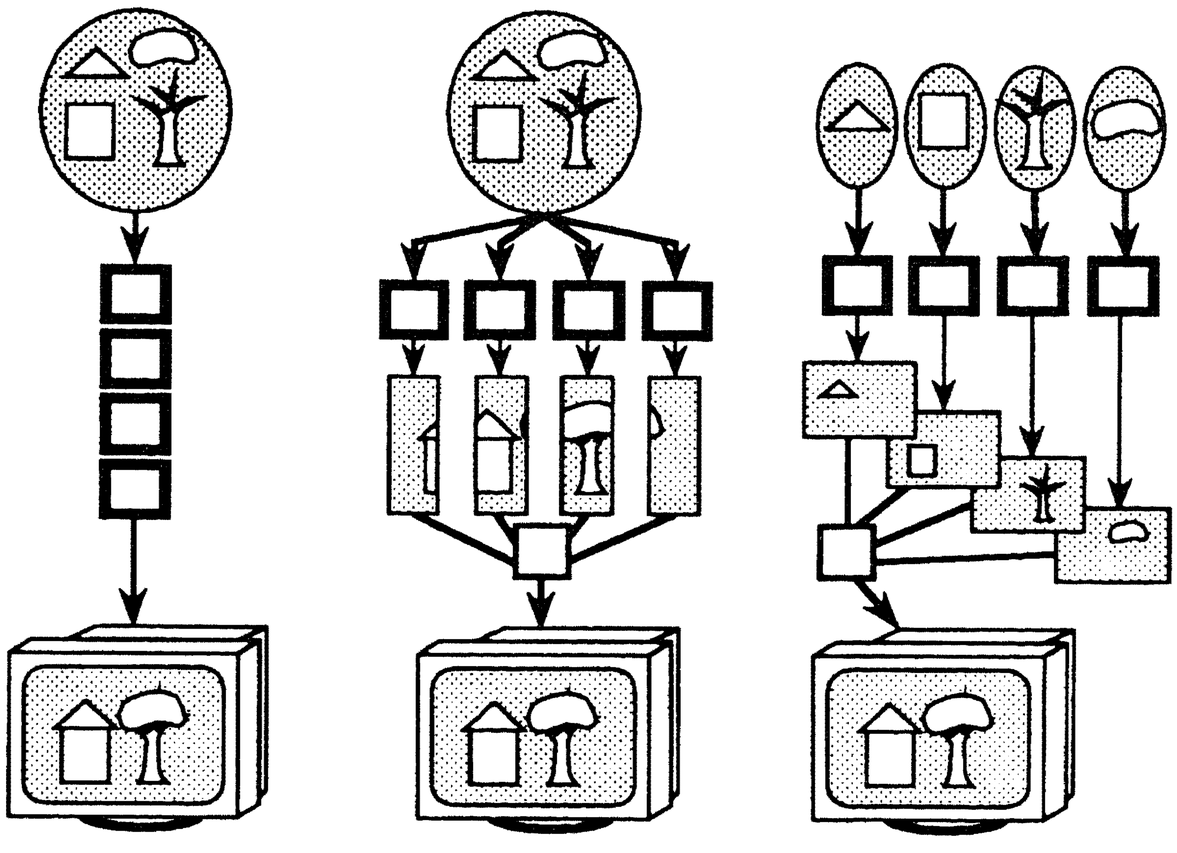

Figure 2. Subdivision strategies.

streams that have to be combined (i.e., the system is pixel-serial). A simple solution implies synchronization which reduces efficiency. An asynchronous solution requires extra memory and composition hardware. Practical solutions in both commercial and academic systems are often mixtures of several of these three 'pure' subdivision strategies.

Presently there exist two mainstream approaches to make raster graphics

systems more viable for interaction:

- Perform each step involved in the graphical feedback loop as quickly as possible by pushing the hardware limits to the maximum, viz., running many processors per given task in the image synthesis pipeline.
- Restructure the functional model to reduce the effort needed to complete the graphical feedback loop, viz., looking at image synthesis from a fresher perspective.

A common characteristic of the first approach is to identify and isolate a simple (subset of) operation(s) and map it, frequently in a conceptually simplistic manner, to hardware.

The second approach is no different from the first one in terms of its goal, i.e., produce a responsive raster graphics system. Yet, the methodology is

quite different. In this case, one analyzes interaction tasks and then tries to develop original data structures and devise new architectural organizations which guarantee that for all interaction tasks representations of the proper level are at hand. Only then one maps expedient tasks into hardware as much as this is justified.

4. RESEARCH AT CWI

The computational complexity community has long ago come to know that the laws of parallel computation are qualitatively different from that of the sequential computation; algorithms do not always smoothly translate from uniprocessor to multiprocessor architectures. We believe that without clarifying the algorithmic improvements, brute-force mappings of existing graphical algorithms into hardware will introduce only temporary speed-ups and these improvements will be nullified in time by growing user demands. The real solution to the hard problems of computer graphics will come, in our view, from a direction which considers the intrinsic difficulty of user driven problems from a computational standpoint. Therefore, research at CWI adhered to the above mentioned second approach: first examine the structure of the image synthesis pipeline in relation with interaction requirements, and only then try to push the hardware limits to the maximum where this is needed [4].

We observed that in an interactive computer graphics application a user interacts with a three-dimensional model (or object) at several levels of abstraction. For an efficient support of interactive editing and incremental updates a raster independent representation of the three-dimensional model should be immediately available at each of these levels. This forms the basis of a lean, yet flexible, computation model.

As a consequence of this raster independent object-space paradigm, the research at CWI has focussed on explicit identification of all visible surfaces, shading methods, rasterization hardware and adaptive rendering.

4.1. Visible surfaces

A fundamental step in generating images of 3D scenes is clipping and hidden surface removal: the resulting image exists of (parts of) surfaces that are visible from a certain position in space. Several types of algorithms exist to tackle this classical computer graphics problem [3]. In most graphics systems hidden surface removal is supported by hardware that checks the visibility on pixel level (the so-called z-buffer algorithm). However, one of the levels of abstraction with which the user interacts most frequently is the level which contains only all visible surfaces. Such a level is not available in a pixel-based z-buffer architecture.

Explicit identification of all visible surfaces implies that the visibility calculation takes place in object-space. Interactive applications involve incre-

mental picture changes. The research at CWI resulted in a hidden surface removal algorithm which includes a set of logical operations on 3D objects. These operations can be used to add and delete individual objects so that incremental changes affect only those objects of which the visibility is changed. Thus a firm basis for interaction and animation is established. Our algorithm operates on a pre-sorted representation of objects and a geometry-based data structure to store these objects. This specific representation of objects reduces the complexity of both the hidden surface removal and the scan-conversion process. The data structure reduces the search space for geometry-based object identification and facilitates data distribution for a multiprocessor implementation.

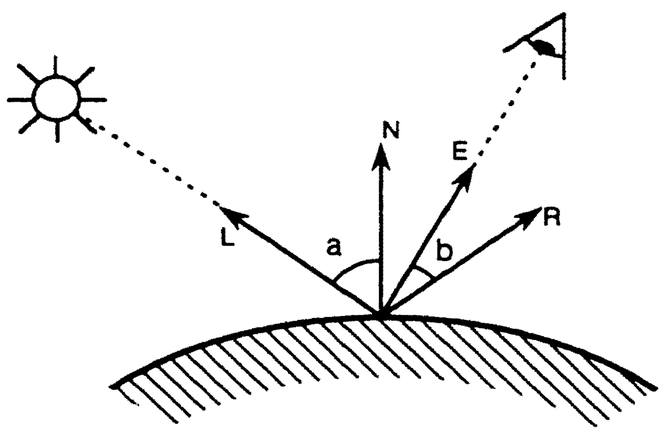

Figure 3. Vectors involved in Phong's illumination model.

4.2. Shading methods

By looking around us in the real world we observe the result of rather complicated physics: the interaction of photons with the inhomogeneous entities that make up the physical environment. This reality is far too complicated to simulate accurately. Therefore computer generated images are produced using a simplified illumination model that describes the interaction between light and the elements of the simulated 3D environment.

Most popular shading methods are based on the illumination model developed by B.T. Phong [2] that has the potential to produce remarkably realistic results, in spite of its simplicity. The model incorporates ambient, diffuse and specular components. The intensity vector I is calculated using the expression

$$I = I_{\text{amb}} + \sum_{\text{sources}} I_{\text{light}} \cdot ((N \cdot L) + (E \cdot R)^n).$$

In this expression I_{amb} represents the amount of energy of the indirect light cast upon the surface area by the environment, I_{light} is the intensity of the light source, **N** is the surface normal, **L** is the direction of the light source, **E** is the direction of the viewpoint, **R** is the direction of reflection and n is a coefficient which relates to the reflectivity of the surface. The vectors **N**, **L**, **E** and **R** are normalized (see figure 3).

The Phong shading method, known since 1975, based on this illumination model involves calculation of the intensity across polygons (a graphics area primitive) based on interpolated vectors on a per-pixel basis. Due to the costs involved (which includes renormalization of interpolated vectors and calculation of the above expression for each individual pixel), this method

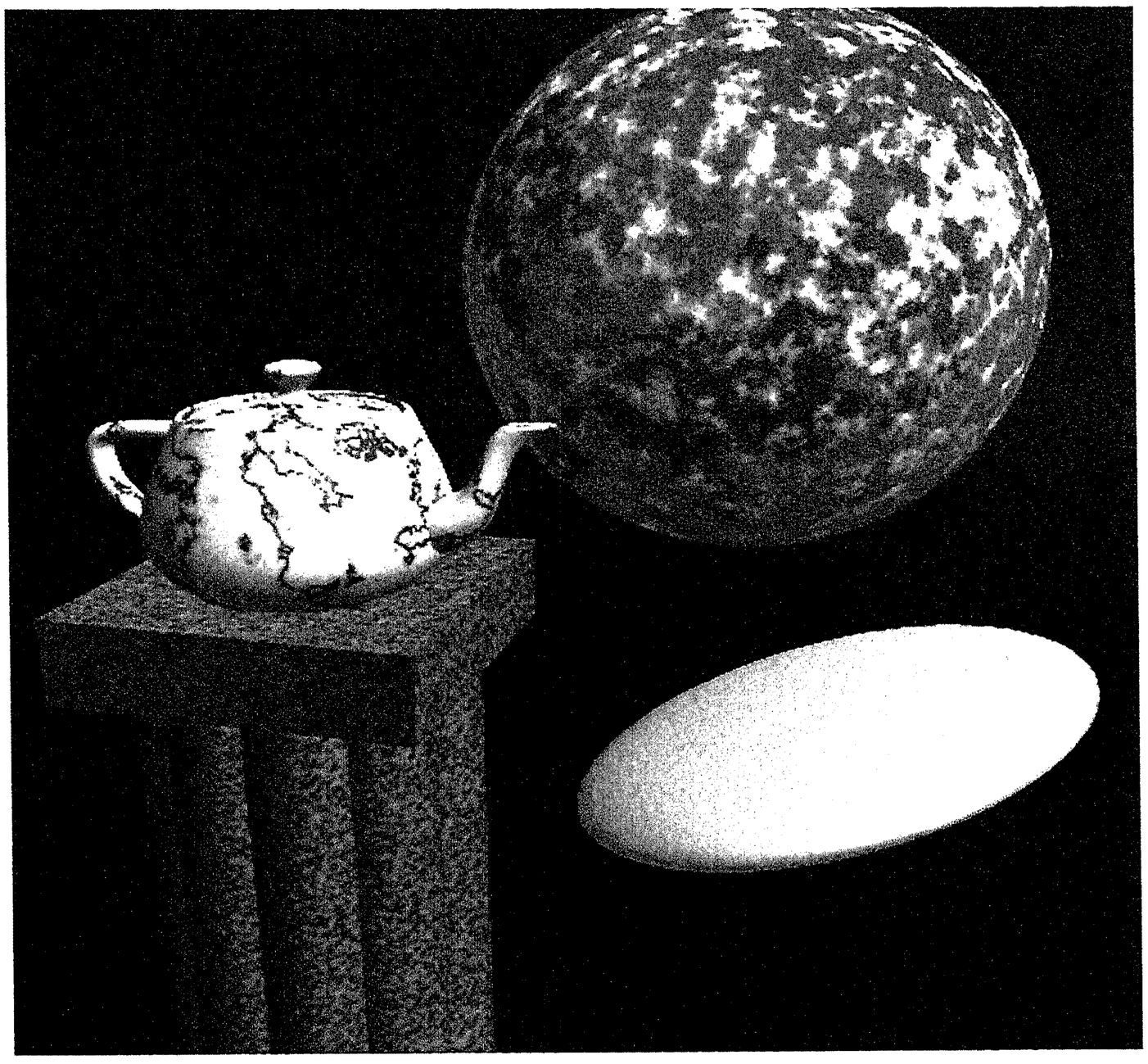

Figure 4. Examples of the most advanced shading methods presently used in interactive computer graphics.

is not suited for high speed rasterization. We developed a similar shading method in which interpolation of vectors involved in the calculation is interpreted as rotations. Spherical trigonometry then leads to a linear expression of the angle between the vectors along a scanline. The outcome is a parameterized piecewise quadratic expression for each intensity term. These terms can be handled directly by forward differencing. The image quality obtained is virtually the same as the quality of Phong shaded images.

4.3. Rasterization hardware

A display controller handles the refresh process of a graphics display device. We developed a display controller which reads 'area drawing instructions' and which in real-time produces scanline based video signals to control the electron beam which scans the display area. The path of the electron beam prescribes the organization of the scan-conversion process (see figure 5). The vertical phase of the scan-conversion process involves calculation of the

intersection of objects with a horizontal scanline and determination of the colour function along that scanline. The actual rasterization takes place in the horizontal phase of the scan-conversion process. As a result of our research on illumination models, this can be reduced to relatively simple operations repeated numerous times. For tasks like this we opted for a custom designed high speed 36-bit forward differencing engine, implemented as a highly pipelined systolic array.

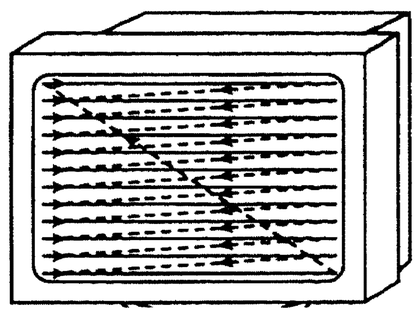

Figure 5. Scanpath of raster display device.

A working prototype system (figure 6) has been built to provide further insight into the operation of the technologically challenging part of the system: the custom VLSI Difference Engine. This prototype produces pictures on a CRT display directly from instructions and without buffering images in a frame buffer.

The Difference Engine, which was developed as a very specialized pixel generator is really very general; it can handle any order forward differences with 36-bit numerical accuracy and an 11ns cycle time. The spline interpolation goes with constant cost independent of span length. Since the Difference Engine can interpolate any spline (polynomial) curve, any signal that is expressed in terms of a spline basis can be reconstructed. Not only that, the architecture with its accumulator allows one to sum over incrementally generated output so that the splines can be summed over different scales to produce the final image to any required accuracy. The reconstruction time depends not on the spacing of the knots in the splines (the lengths of the interpolation spans) but only on the number of knots. An image can be decompressed even at video rates provided that the number of knots is less than the number of pixels to be generated (by some fixed overhead per scanline).

This has opened the way for using this hardware also to reconstruct images that have been coded with a wavelet transform [5]. The wavelet transform is a multiresolution description of the image that can be decoded to yield more and more accurate reconstructions of an image. The transform also precisely locates high-frequency features in space and low-frequency signals in the frequency domain. In fact it is often argued that wavelet transforms perform better than the discrete cosine transform advocated by the JPEG standard, it fits in better with human perceptual aptitudes and is a more compact coding.

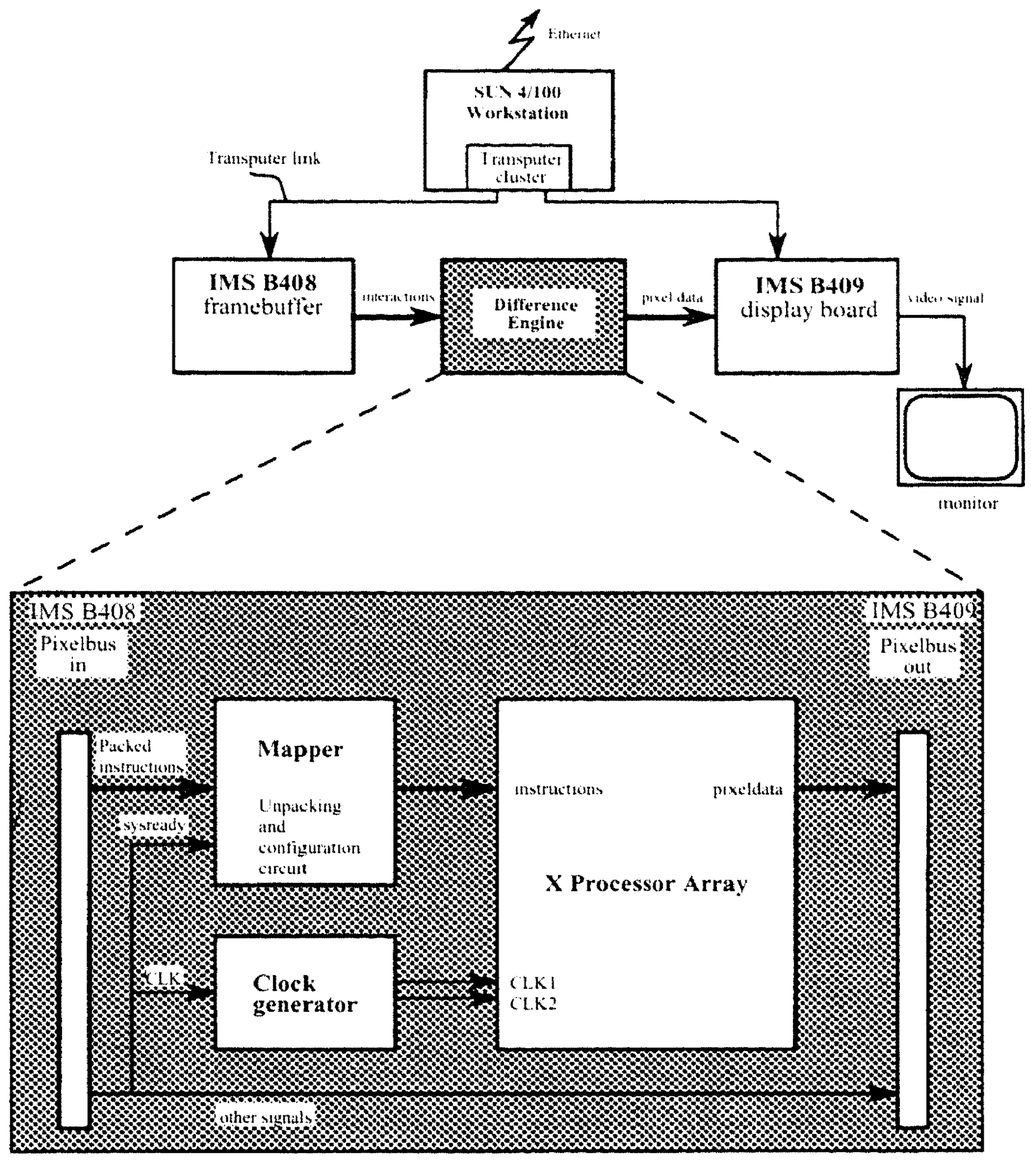

Figure 6. Difference Engine embedded in prototype.

4.4. Adaptive rendering

Generating a physically perfect image would by far exceed the processing power of any state of the art supercomputer. Due to this, a whole scale of rendering models emerged, each with a different level of approximation of the 'physical correct' image. Based on requirements of a specific application, one of these rendering models can be selected. If the system is tuned for worst case situations the efficiency will be far from optimal. On the other hand if the system is optimized for 'normal' situations its worst case behaviour may be unacceptable.

Adaptive image generation is a means to adjust the image generating process to the possibilities of a particular moment. The quality of the image, and thus the cost of rendering, is related to the time available between successive updates. Ideally, this results in the best possible image at any time. There are two distinct aspects that determine the cost of the image generation process: the quality of the rendering process, and the quality of

the object representation. These can be varied according to the need.

For a study on adaptive rendering we implemented an environment to test a rule-based system (ADMIRE). This rule-based system serves to optimize the performance of interactive graphics applications by dynamically selecting rendering algorithms, data structures and level of detail on a per-object basis.

5. Conclusion

A thorough rethink of interactive graphical workstations from a user point of view has uncovered a range of novel approaches to system architectures. CWI has built an integrated set of solutions based on these insights that span conceptual, software and hardware solutions. Interesting spin-offs in image compression are also being exploited.

References

1. S. Molnar, H. Fuchs (1990). Advanced raster graphics architecture. *Computer Graphics: Principles and Practice*, The Systems Programming Series, Addison-Wesley Publishing Company, Reading, Massachusetts, 855-920.
2. B.T. Phong (1975). Illumination for computer generated images. *Communications of the ACM 18(6)*, 311-317.
3. I.E. Sutherland, R.F. Sproull, R.A. Schumacker (1974). A characterization of ten hidden-surface algorithms. *Computing Surveys 6(1)*, 1-55.
4. A.A.M. Kuijk, E.H. Blake, P.J.W. ten Hagen (1992). *An Architecture for Interactive Raster Graphics*, CWI Report CS-R9229.
5. P.C. Marais, E.H. Blake, A.A.M. Kuijk (1993). A spline-wavelet image decomposition for a difference engine. *CWI Quarterly 6(4)*, 335-362.