I.M.O.G.E.N.E.—A Solution to the Real Time Animation Problem

Christophe Chaillou, Michel Meriaux and Sylvain Karpf

ABSTRACT Current graphics processors are very slow for displaying shaded 3D objects. A lot of work is being done in order to define faster display processors by using massive parallelism and VLSI components. Our proposal goes along this line with the supplementary aim of displaying images in real time, i.e., 25 or 30 times per second. We choose to design a graphics module without any working memory and thus without frame buffer. A massive parallelism over objects, and thus a pixel pipe-line, are used. Each Object Processor handles one 3D object; all the processors work in a synchronous way, processing the same pixel simultaneously at pixel rate. These processors are built from very simple Elementary Processors (2 adders, 2 registers and 6 memory words) computing linear or quadratic expressions V(x,y), where (x,y) are the coordinates of a pixel. A pipelined tree made of basic operators (min, max, or, and, ...) gathers the results given by the Object Processors and makes inter-objects operations, i.e., at least hidden part elimination. Such a choice of course involves a high hardware complexity when displaying rather simple scenes. However, we feel that it is the price to pay for building graphics processors allowing real-time interactive animation (e.g., the graphics unit of a driving simulator).

1 Introduction

Our concern in this paper is to propose an original architecture for displaying images in real-time, thus obtaining a graphics processor for interactive animation. We use geometric rendering methods rather than ray-tracing or radiosity which are currently quite incompatible with the real-time constraint. Our choices are to associate graphics objects to processors and to use massive parallelism.

In a first part we shortly describe the display process of an image and the generally proposed architectural solutions. Then we present new solutions being studied and using a massive parallelism. At last we introduce a solution allowing for a real-time display processor.

The second part is a complete description of the I.M.O.G.E.N.E. machine. A special emphasis is put on the design and realization of Elementary Processors.

2 State of the Art

2.1 Rasterization [1]

The process generating pictures elements values from a geometric description of the image is called rasterization, and may be divided into four steps.

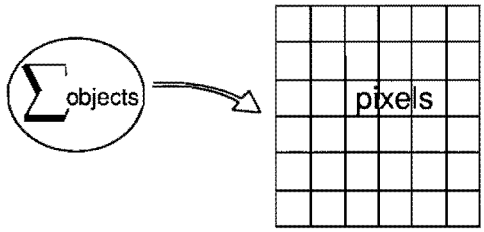


Fig. 1. Scan Conversion

- Traversal
  - Traverse the data structure, delivering graphics objects and light sources to the appropriate processor(s).
- Transformation
  - Objects and light sources positioning.
  - Coordinate system transformation.
  - Clipping.
- Scan Conversion
  - Transform each object into a set of pixels.
  - Hidden part elimination.
- Display

This description clearly shows that rasterization may be split into two main steps, one dealing with objects, and the other with pixels. As a matter of fact converting graphical objects into pixels requires many computations.

When real time animation is required, rasterization must be done for all the objects at image display rate. Moreover, rasterization may deal with more complex inter-objects computations (such as shadowing or transparency) that have not been presented here.

2.2 The Classical Architecture [2]

Figure 2 shows the architecture of currently available display machines.

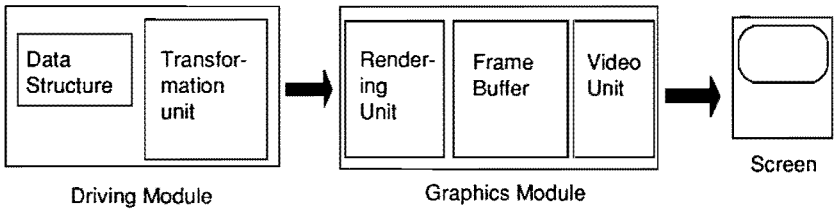


Fig. 2. Classical Rasterization Systems

All the current commercial graphics machines use this architecture. The most powerful ones are the Silicon Graphics 4D/240 GTX [3] and the STELLAR GS1000 [4]. They are able to display 100,000 small Gouraud-shaded three-dimensional triangles per second.

Such an architecture has the following limits:

- Limited real-time performances.
- Poor image quality due to Gouraud shading.
- Strong bottleneck between the rendering unit and the frame buffer.
- Triangle tesselation is not always an accurate solution.

2.3 New Architectures

In the recent years have been proposed many new architectures in order to build very powerful graphics modules. The most powerful ones use massive parallelism and VLSI components. When using such components, new constraints arise with regard to the machine design: the processors we intend to use must be simple enough to be realised in VLSI, and identical in order to limit costs. It is then interesting to choose a modular architecture in order to be able to increase machine capabilities only by adding new components.

With these assumptions, two approaches may be considered for the graphics module architecture:

- Image space partitioning, i.e., to associate processors to pixels. One processor per pixel is used, with massive pixel parallelism and thus object pipe-line. This solution can be viewed as an intelligent frame buffer.
- Object space partitioning, i.e., to associate processors to graphics objects. Thus object parallelism is used. Different objects are processed in parallel at pixel rate, pixels being processed in a pipe-lined way. With this solution the frame buffer appears to be unnecessary if inter-objects processings are done at pixel rate. Objects could be processed at a different rate, but this would involve the two following constraints:
  - There are access conflicts between the Objects Processors and the (necessary) frame buffer.
  - Inter-object operations can only be done in the frame buffer.

  Thus, object massive parallelism is interesting only when Object Processors do work at pixel rate.

2.4 Pixel-oriented Systems [5]

Such a machine has one processor per pixel. Two solutions with regular architecture can be considered for sending the objects to the processors:

- Broadcasting: all the objects are sent separately to all the pixels, all the processors being independent. The Pixel-Planes machine [6, 7] belongs to this class, if considering that all the objects are coded with linear expressions and that the adder trees are a broadcasting mechanism.
- Pipe-line network: the objects go from left to right through the network. The inputs are on the left edge; every processor receives the objects from its left neighbour and transmits them to its right one. The objects must have been split into horizontal spans. It is likely that such an intelligent frame buffer could be built by using one thousands of SAGE chips [8].

The main features of intelligent frame buffers are:

- Rigidness: as processors are designed for specific data types, any change would imply their full redesign.
- Uniformity, which allows an efficient VLSI implementation.

With regard to our aim (a graphics module for real-time animation), this solution appears to have some drawbacks:

- The execution time on different processors depends on the number of objects the processor has to deal with. So it is hard to guarantee that real-time will be obtained.
- There are access conflicts to the frame buffer between processors and video stream [9].
- It is not obvious that the whole scene could be loaded during the time of one image.

These are the reasons why we choose the other solution: associate the processors to the objects to be displayed.

2.5 Object-oriented Systems

Figure 3 shows the architecture of a general object-oriented machine. Every Object Processor (O.P.) outputs for every pixel at pixel rate the features of the corresponding object. The decision unit makes inter-objects operations, i.e., at least hidden part elimination.

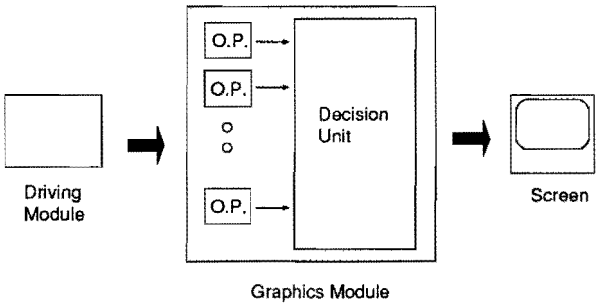


Fig. 3. Object-oriented rasterization system

Three open questions remain: i) how to make the decision unit, ii) with what strategy to allocate objects to the Object Processors, iii) what graphical primitives to choose?

The GSP-NVS machine [10] is a recently proposed massively parallel object-oriented machine. It uses the pipeline structure of the Weinberg machine [11], which was the first object machine proposed in 1981. GSP-NVS is made of a pipeline of VLSI processors. Every processor handles a 3D triangle. The hidden surface elimination is made by a pipelined Z-buffer algorithm. Every processor receives from the previous one in the pipeline the value to be displayed for every pixel. If, for a given pixel, the corresponding triangle has a smaller depth, the processor replaces the value by one of its triangles, else the 'old' value follows. Indeed the decision unit is distributed over every line. A pipelined post-processor computes the shading by a Phong's method for 5 light sources. PROOF [12] is another project using the same approach.

With regard to real-time display, such a machine reveals some drawbacks:

- It requires for every image a sort of the triangles depending on the vertex with the maximum ordinate, which is unthinkable in real-time.
- It may happen that on a given line there are more triangles than Object Processors. Thus all the triangles cannot be processed in one pass and real-time is lost.

The only solution to ensure real-time is to assign objects to processors before the display phase. Moreover, each Object Processor should handle only one object, because assigning several objects to the same processor would require preprocessing incompatible with the real-time display.

The idea of dealing with shading in a post-processor appears to be very fruitful, because the driving unit does not have to take care of these (heavy) computations. This post-processor must receive for every pixel the visible object with its depth and normal, and computes at pixel rate the RGB values. The driving unit must send at the beginning of every frame the positions and features of the various light sources. Another advantage of such a solution is that shading is computed for visible pixels only.

The previous analysis gives us the general architecture of a real-time object-oriented machine (see Figure 4).

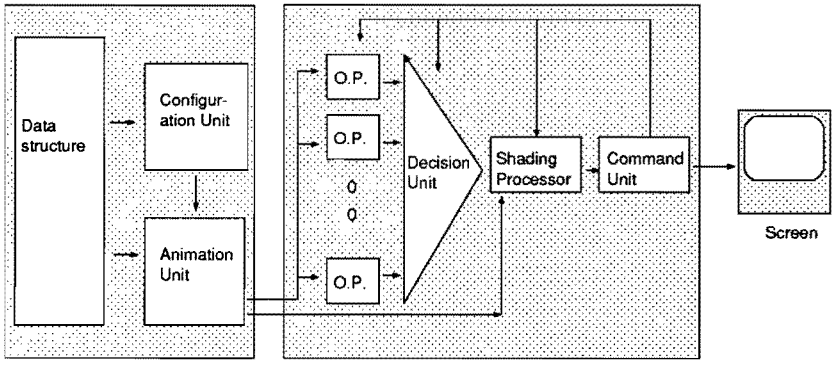


Fig. 4. Real-time object-oriented rasterization system

The configuration unit assigns the objects to the processors and defines the structure of the decision unit. The animation unit gives every processor at image rate the features of the corresponding object. The whole graphics module is synchronised by the command unit.

3 Description of I.M.O.G.E.N.E. [13]

I.M.O.G.E.N.E. (Image by Means of Objects GEnerated by Numerical Expressions) is a special implementation of the above architecture.

We choose to have synchronous Object Processors, i.e., they process the same pixel at the same time. Then the decision unit is a binary tree with $\log_2 N$ levels, N being the number of objects. The decision unit makes the hidden part elimination by a distributed Z-buffer algorithm [13].

We will now describe in detail each unit of I.M.O.G.E.N.E.

3.1 Object Processors

Introduction

It is quite difficult to choose the object processors, since this choice on the one hand determines the kind of scenes that can be displayed, and on the other hand the VLSI feasibility. Indeed, if we choose a too simple basic object, the number of processors necessary for a good modelling will be very high; but if it is too complex, VLSI integration could be impossible.

Whatever the Object Processor may be, it has to indicate the decision unit whether the corresponding object is present or not at the current pixel. If it is the case, it outputs to the decision unit the depth of the object and the normal components at this point.

Moreover, it looks interesting to choose objects as close as possible to the forms they have in the data structure, in order to limit the host work. We have chosen rather complex basic objects, and thus rather complex Object Processors, but composed of several Elementary Processors working synchronously in parallel.

The Elementary Processor

Definition In the first part of this paper we have clearly shown that, with object partitioning, the Object Processors have to output values all over the scene. The Elementary Processors must have the same features. Moreover, these values must be produced with the correct sequence (i.e., after the value in (x,y), they have to produce the value in (x+1,y) then (x+2,y)...). A simple solution consists of defining the Elementary Processor as an entity computing an expression $V(x,y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$ (or $F(x,y) = Ax + By + C$), with $A, B...F$ integer constants and (x,y) the pixel coordinates; this value can be incrementally computed at pixel rate for every pixel at (x,y).

Let us remark that this choice is not at all arbitrary, since Pixel-Planes 4 [6][7] and Pixel-Planes 5 [14] have clearly shown that such expressions could model rather realistic scenes.

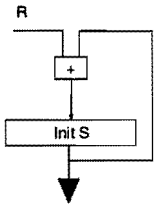
Realization We will now detail the incremental method used for $V(x,y)$ evaluation. As the method for F follows immediately, we will not present it.

N.B.: in the following, EP1 will be the Elementary Processor F and EP2 the one for V.

Given $P(y) = Cy + D$ and $Q(y) = By^2 + Ey + F$, it follows that $V(x,y) = V1(x) = Ax^2 + Px + Q$.

The computation method consists of computing $P(y)$ and $Q(y)$ at the beginning of every line, and then for every pixel computing $V1(x)$.

Let us note that the opposite structure can be used to incrementally compute the first order expression Rx+S. Moreover, $V1(x+1) = V1(x) + (V1(x+1) - V1(x)), V1(x+1) - V1(x)$ being a first order expression, thus computable with this structure. $V1(x)$ can thus be incrementally computed by means of two serially connected structures. This new element may also be used for $P(y)$ and $Q(y)$ computations if backup registers are available. Figure 5 shows the Elementary Processor scheme and its working algorithm (assuming a non interlaced monitor).



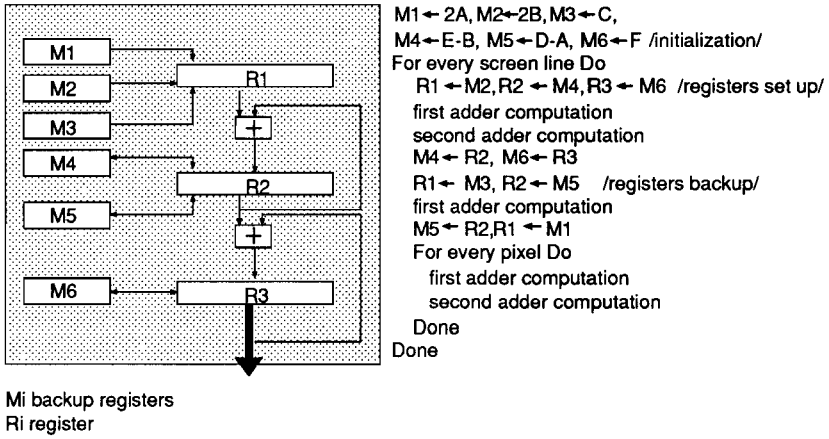


Fig. 5. Elementary Processor structure

Let us note that such a structure involves a high connection complexity and is not well suited for interlaced monitors. These are the reasons why we chose a less complex solution, using a unique backup RAM instead of backup registers (see Figure 6). The backup RAM must contain 6 memory words for a non interlaced monitor, and 12 for an interlaced one (since original coefficients must be available at the beginning of the second half frame).

A pipe-line effect appears in the Elementary Processor structure (the two additions may be computed simultaneously). Thus pixel generation time is set by the slowest element of the structure.

The working algorithm of the Elementary Processor is a bit different from that given in Figure 5: registers backups must be done through the whole structure, requiring at least three clock cycles (but during horizontal retrace). This algorithm is implemented on a microprogrammed controller.

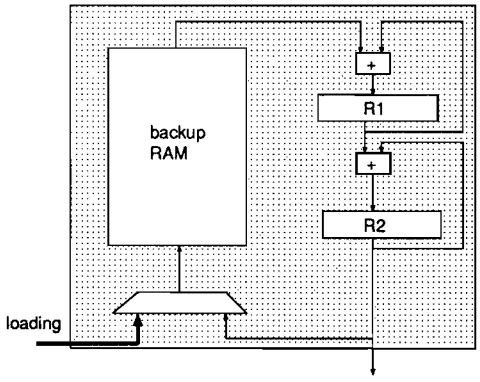


Fig. 6. Elementary Processor architecture

Object Construction

The outputs of the Elementary Processors may be interpreted as:

- A binary value (negative = absent; positive = present) indicating whether the object is present at the current pixel; combining such values allows to define the border of the object.
- The depth of the object in the viewer's reference.
- One of the components of the normal to the object at the current pixel.

These different values are combined in a pipelined tree called the Object Maker in order to define a given object. Thus an Object Processor has the following structure:

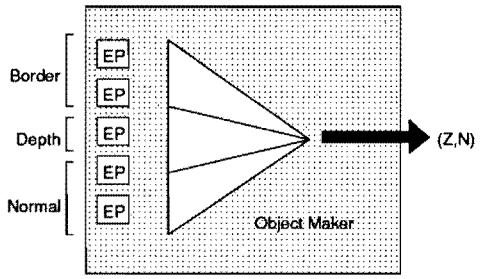


Fig. 7. An Object Processor Structure

Objects must be convex so that their screen border can be computed by combination of linear or quadratic expressions in a fixed structure whatever the perspective we use. Thus, the basic objects of I.M.O.G.E.N.E. are convex polyhedra, spheres, cylinders...

Let us remark that the explicit construction of convex polyhedra is unnecessary, since their faces can be considered as independent objects, the decision unit implicitly making the polyhedron construction by eliminating hidden parts with the Z-buffer algorithm.

We describe in the following Object Processors associated with a 3D triangle and a sphere.

Triangular Face

It may be defined by:

- Three Elementary Processors EP1 defining the border of the triangle by intersection of three half-planes.
- One Elementary Processor EP1 giving the depth for every pixel (i.e.,the equation of the 3D plane containing the triangle).
- Three Elementary Processors EP1 computing the three components (Nx, Ny, Nz) of the normal to the face by a bilinear Phong interpolation if the face approximates a surface, three constants if not.

Figure 8 shows a triangular face processor.

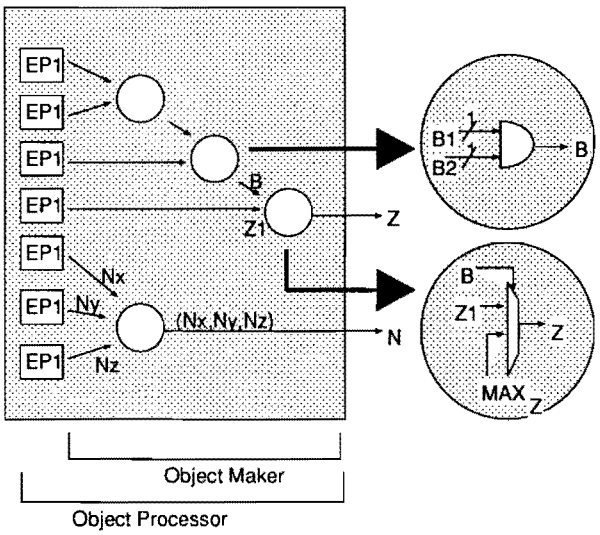


Fig. 8. A triangular face processor

Sphere

A sphere is defined by:

- One Elementary Processor EP2 to define the border of the sphere.
- One Elementary Processor EP2 to approximate the depth with a parabola.
- Three Elementary Processors (one EP2 and two Epls) to define the normal to the sphere.

The front half sphere is approximated with a parabola (the best parabola has been found using integral square error approximation). Simulations have shown that such an approximation is quite accurate. The Nz component is also an approximation of the same kind.

Figure 9 shows a sphere processor.

3.2 The Decision Unit

It is a pipelined binary tree implementing a distributed Z-buffer algorithm. All the nodes of the tree are identical; they keep the pair (Z,N) with the lower depth (see Figure 10). The decision unit may also deal with more complex inter-objects computations (e.g., shadowing and transparency), but this would involve a much higher hardware complexity.

3.3 The Shading Processor

It also works at pixel rate. It receives for every pixel the depth, the normal and the intrinsic features (basic colour, reflection coefficient) of the visible object, and computes the RGB values to be displayed according to light sources characteristics. In order to improve the quality of the images, the shading processor will handle diffuse and specular reflections.

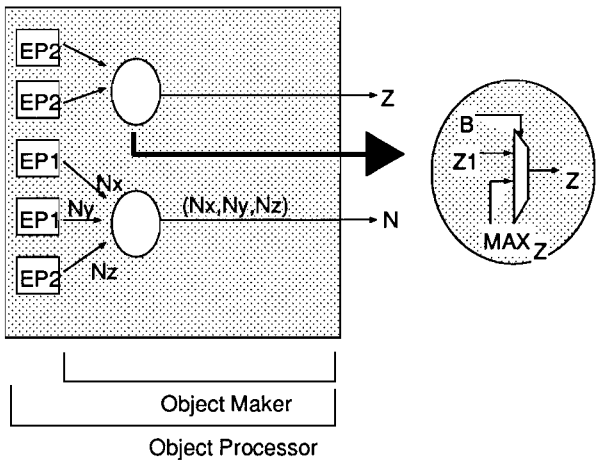


Fig. 9. A sphere processor

3.4 The Command Unit

It handles the system clock, micro-controllers and screen synchronization signals, and the Digital to Analogical conversions between the shading processor and the screen. It can be built with any standard graphics controller.

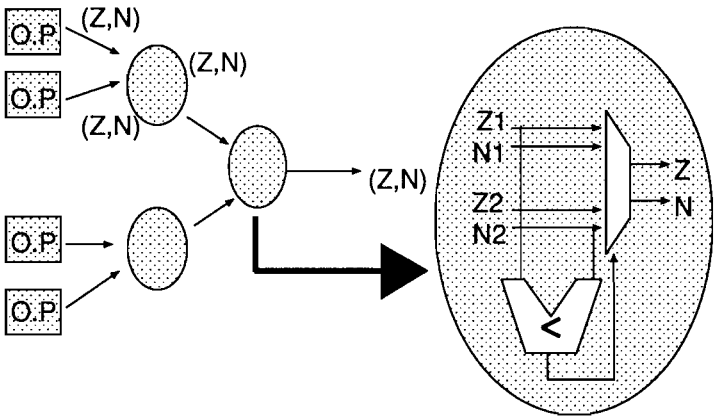


Fig. 10. The decision tree

3.5 The Driving Module

This unit is implemented on a general purpose computer, and is divided into two units.

The Animation Unit

The animation system computes the expressions coefficients according to objects and viewer positions. A real-time animation will be obtained only if the system is able to calculate them at screen rate; moreover, coefficients loading from the driving unit to the

Elementary Processors, and light sources information transfer to the shading unit must be achieved in the same time.

It is highly likely to use several processors for such computations in the (general) case when objects are independent. Thus coefficients loading is distributed over several links and the risk of bottleneck decreases.

The Configuration Unit

The Configuration Unit has to initiate the Display Unit, i.e., define the object processors structure and the way coefficients will be loaded into the Elementary Processors, before the real-time activity. The data structure must also describe for every object the elementary processor structure needed for its display.

Link Between the Driving Module and the Graphics Module

In order to guarantee real-time animation, the Elementary Processors backup RAMs must be refreshed at the beginning of every frame. In fact, time available for the host computer to compute and load all the Elementary Processors coefficients is $t1 + t2$, where $t1$ is display time and $t2$ vertical retrace period.

Our solution is to use a intermediate RAM where coefficients of the next frame to be displayed are stored. This RAM is controlled during period $t1$ by the host processor, and during period $t2$ by a wired automata that loads new coefficients into the Elementary Processors RAMs. Furthermore, this memory should be considered as an extension board of the host processor memory: this greatly reduces the host work, since coefficients allocation may be considered as a dynamic memory allocation.

Figure 11 illustrates the loading principle.

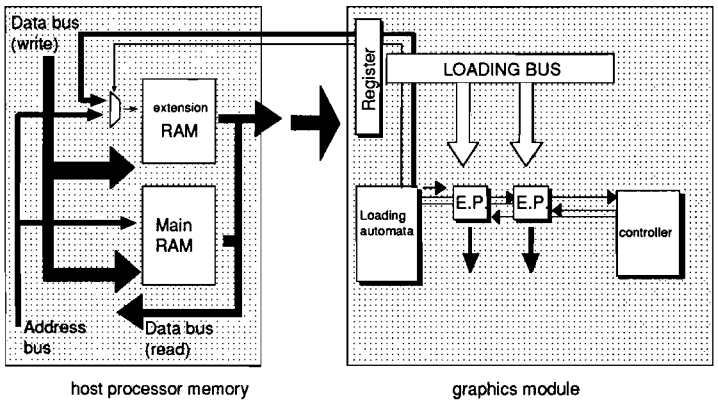


Fig. 11. Link between the driving module and the graphics module

4 Conclusion

4.1 Simulation

We are developing a simulation of the whole machine (Object Processors, Decision Tree and Shading Processor) on a Transputers-based workstation. This simulation has already given interesting results. Direct display of spheres or cylinders generated by means of few

first and second order expressions looks better than the classical triangle tesselation. We are not studying how to build more complex objects with such expressions.

4.2 Realization

We have just finished the design of one second order Elementary Proccessor on a VLSI CAD software. We now intend to use these chips for designing an Object PCB, which would allow us to build a complete prototype (with of course only a small number of objects). The next step would be the design of a complete Object Processor, leading to a more powerful prototype. In the same time, we are developing the Shading processor.

4.3 Expected Performances

Performances directly depend on the number of Objects Processors. Moreover, classical performances tables (in number of small triangles per second) are inadequate for our machine. For example, 1000 spheres Processors would allow to display 50,000 spheres per second with a 50 Hz monitor (but only 1000 simultaneously on the screen), corresponding to at least 1,000,000 triangles per second for tesselation-based machines.

The main drawback of our machine is the limited number of simultaneously displayed objects according to the number of available object processors. This limitation can be easily overcome by connecting a classical frame buffer to the shading unit, using several frames to build a complete image (but of course losing true real-time display). With 1000 spheres processors, we could then display up to 50,000 spheres simultaneously, but at only one frame per second.

4.4 Future Work

As said in this paper, the objects our machine can handle are not true quadric volumes, since the second order Elementary Processor allows us to define only approximated objects (for the moment only spheres and cylinders). In order to overcome this limitation, and thus to build a dramatically more powerful machine, we are now working on a true Quadric Elementary Processor (called Q. E. P.), able to solve in real-time the general quadric equation $Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + K = 0$.

Such a processor will allow us to display in real-time any quadric (ellipsoid, cylinder, paraboloid, hyperboloid...). Moreover, these quadrics will be defined in the host computer directly by their equations in the screen reference.

References

[1] Meriaux, M.: Contribution à l'imagerie informatique : aspects algorithmiques et architecturaux. *Thèse d'état* USTLFA Lille 1984.

[2] Gharachorloo, N. and Gupta, S.: A Characterization of Ten Rasterization Techniques. *ACM Computer Graphics* vol. 23 no. 3, 1989, pp. 355-368.

[3] Akeley, K.: The Silicon Graphics 4D/240GTX Superworkstation. *IEEE Computer Graphics and Applications*, vol. 9 no. 4, 1989, pp. 71-83.

[4] Apgar, B. and Bersack, B. and Mammen, A.: A Display System for the Stellar Graphics Supercomputer Model GS1000. *ACM Computer Graphics* vol. 22 no. 4, 1988 pp. 255-262.

[5] Lepretre E.: Algorithmique parallèle et architectures spécialisées pour la synthèse d'images. *Thèse de Doctorat* USTLFA Lille 1989.

[6] Fuchs, H. and Poulton, J.: Pixel-Planes:A VLSI oriented design for raster graphics engine. *VLSI Design* no.36, 1981, pp. 20-28.

[7] Eyles, J. Fuchs, H.: Pixel-Planes: a summary, in *Advances in Computer Graphics Hardware II*, Springer-Verlag, 1988, pp. 183-207.

[8] Gharachorloo, N.: Subnanosecond pixel rendering with million transistor chips. *ACM Computer Graphics* vol. 22, no. 4, 1988, pp.41-49.

[9] Cordonnier, V. and Meriaux, M.: Image display from multiprocessors. *SID International Symposium*, Digest of technical papers, vol. 21, 1990, pp. 49-52.

[10] Deering, M.: The triangle processor and normal vector shader: a VLSI system for high performance graphics. *ACM Computer Graphics* vol. 22, no. 4, 1988, pp. 21-30.

[11] Weinberg R.: Parallel Processing Image Synthesis and Antialiasing. *ACM Computer Graphics* vol. 15, no. 3, 1981, pp. 55-62.

[12] Schneider, B. and Claussen, U.: PROOF: An Architecture for Rendering in Object Space. *Advances in Computer Graphics Hardware III*, Springer-Verlag, 1989.

[13] Chaillou, C. et al.: A real-time Image Generator. *Proc. of the Eighth IASTED International Symposium Applied Informatics*, ACTA PRESS, 1990, pp. 140-143.

[14] Goldfeather, J.: Fast constructive solid geometry display in the Pixel-Powers graphics system. *ACM Computer Graphics* vol. 20, no. 4, 1986, pp. 107-116.