Towards a Taxonomy for Display Processors

Bengt-Olaf Schneider

Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Graphisch-Interaktive Systeme
Auf der Morgenstelle 10 / C9
D-7400 Tübingen, FRG
e-mail: igbengt@dtupev5a.bitnet

Abstract

Image generation for raster displays proceeds in two main steps: geometry processing and pixel processing. The subsystem performing the pixel processing is called *display processor*.

In the paper a model for the display processor is developed that takes into account both function and timing properties. The model identifies scan conversion, hidden surface removal, shading and anti-aliasing as the key functions of the display processor. The timing model is expressed in an inequation being fundamental for all display processor architectures.

On the basis of that model a taxonomy is presented which classifies display processors according to four main criteria: function, partitioning, architecture and performance.

The taxonomy is applied to five real display processors: Pixel-planes, SLAM, PROOF, the Ray-Casting Machine and the Structured Frame Store System.

Investigation of existing display processor architectures on the basis of the developed taxonomy revealed a potential new architecture. This architecture partitions the image generation process in image space and employs a tree topology.

CR Categories and Subject Descriptors:
B.4.2[Input/Output and Data Communications]: Input/Output Devices — Image Display
C.1[Processor Architectures]
I.3.1[Computer Graphics]: Hardware Architecture — Raster display devices

General Terms: Algorithms, Design, Performance

Additional Keywords and Phrases:
Raster Graphics, Display Processors, Taxonomy, Classification, Model, Partitioning, Architecture

*The author is now with the IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, e-mail: bosch@ibm.com.

1 Introduction

During the last decade many computer graphics systems have emerged. A lot of them achieved their often impressing performance by using specially designed hardware. Especially with the move to raster graphics the quest for powerful hardware accelerators became more and more urgent. This is because the rendering of a million pixels and more requires a lot of computing power.

The different requirements for cost, performance and functionality gave rise to a great variety of computer graphics systems. All these systems try to optimise the compromise between the often contradicting design constraints. That optimisation is a rather difficult task because the design space, i.e. the number of possible alternatives, is huge. With the advent of VLSI technology the design space grew even larger. On the other hand VLSI technology introduced new constraints that required new tradeoffs. For instance, in contrast to conventional technology, in VLSI it is favourable to design systems with many identical and simple elements instead of a few, very complex parts. The paper takes these aspects into account.

We will concentrate on a special piece of hardware for computer graphics systems: the *display processor*. The display processor is performing the low-level tasks in the image generation process, namely pixel generation. We will present a classification scheme for these processors. We hope that such a taxonomy will help in the design of display processors. By providing means to classify different architectures and their properties it will be easier to compare and judge different designs. Furthermore, the taxonomy can serve as a starting point for the specification of a new display processor. It should be noted here that a taxonomy cannot (and is not intended to) provide figures of merit that identify one architecture to be *better* than another one. It can only supply criteria along which an evaluation could take place. In other words: the taxonomy is only a means to describe the available design space.

The paper starts with a short review of earlier attempts to classify display processors. Afterwards we present our own approach. First, we develop a model for the display processor that takes into account both, functionality and timing properties of the display processor. This model serves as an implicit definition of the term *display processor* and as a guideline for the taxonomy which is introduced in the next section. The employed criteria for the classification are presented and discussed. The classification scheme is then applied to some published architectures. The last part of the paper proposes a new display processor architecture that is the result of a search for "white spots" in the taxonomy.

2 Related Work

In [**Abram et al. 1986**] [**Fuchs 1988**] [**Kilgour 1985**] [**Dew et al. 1985**] overviews of existing graphics architectures are given. They only make the distinction whether a system partitions the image generation in image space or in object space. In [**Kilgour 1985**] this is called object serial or pixel serial respectively.

In [**Reghbati et al. 1988**] this classification is refined further. It is determined in how many sets the objects (pixels) are divided and what is the maximum cardinality of these sets.

3 Modelling the Display Processor

3.1 General Description

Producing computer generated images involves two main steps. The first one, *geometric processing*, prepares an image described on a high level of abstraction for the display on a physical output device. This involves the transformation and projection of a scene into a normalised coordinate system which can be easily mapped onto the output device. These coordinates are called normalised device coordinates (NDC). Preparing an image description for display also requires to break down the objects in the scene into simple geometric shapes, that can be easily handled by the low-level hardware. These shapes are referred to as atomic objects or primitives. The geometric processing also comprises calculations for the illumination of the scene, e.g. vertex colours or shadow polygons. Furthermore, in some implementations also a part of the hidden surface removal takes place in this step, e.g. assignment of priorities to objects in the scene and backface culling.

In the second step the scene description in NDC is processed further to be displayed on the output device. Since we restrict ourselves to raster devices this step is called *pixel generation*, because the final pixel colour is determined during this step. To this end, some or all of the following tasks have to be fulfilled:

Scan Conversion actually maps the primitives from NDC to PDC (Physical Device Coordinates), i.e. those pixels are identified which are covered by the considered primitives. This step is always part of the pixel generation process.

Hidden Surface Removal identifies those portions of a primitive that are visible to the viewer, i.e. that are not obscured by other objects. Usually, this problem is also solved during the pixel generation.

Shading. The colour of the different objects at the single pixels is a function of the optical properties of the objects and the spatial arrangement of objects and light sources in the scene. Shading computations approximate this complex relationship. The complexity of the shading calculations depends strongly on the actual implementation. Some systems perform only flat shading, whereas others implement computationally more complex shading models. There are several ways to partition the steps involved in the shading calculations between the geometric and the pixel processing.

Anti-aliasing. If no special care is taken raster images show artifacts that stem from the limited spatial sampling frequency. Anti-aliasing tries to alleviate these effects. Although it is commonly agreed that proper anti-aliasing is necessary, not all graphics systems are able to compensate aliasing effects.

We call that part of an image generation system performing the pixel generation the *display processor*. Different names for the display processor are pixel generator, display controller, or rasteriser. We prefer the term display processor because it reflects the fact that nowadays these pieces of hardware are fairly complex and also show a limited programmability.

It should be noted here that it is of course possible to implement most of the tasks for pixel generation in software. However, this would result in a system with low performance. We will not further investigate that alternative here.

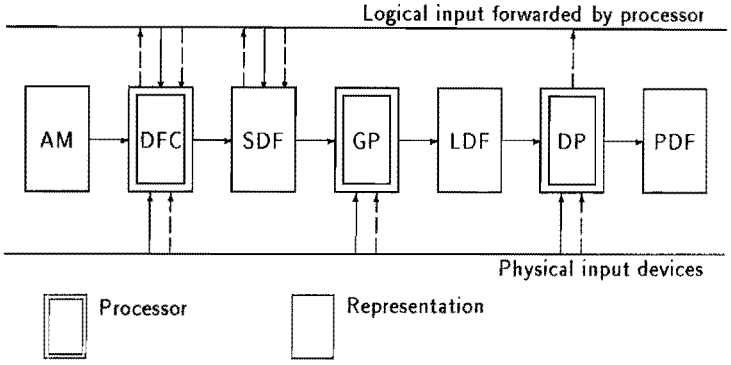

Figure 1: **Functional model of an image generation system.** AM – application model, SDF – structured display file, LDF – linear display file, PDF – physical display file, DFC – display file controller, GP – geometry processor, DP – display processor. The solid arrows symbolise input used to modify output. The dashed arrows denote input forwarded to other processors.

3.2 A Functional Model of the Display Processor

In order to get a clearer understanding of the tasks and functions of the display processor, we want to look closer at the underlying display processor model. We will try to give a somewhat formal description of its functions.

3.2.1 Output

Figure 1 shows a simplified functional model of an image generation system [**Carlbom 1980**]. A similar model has been proposed in [**Kilgour 1981**]. Both models partition the image generation process according to the properties and the levels of abstraction of the intermediate image descriptions.

Starting from the application model (AM) the scene is translated into a structured and often hierarchical description (SDF). This description is then transformed into a linear, non-hierarchical display file (LDF). The LDF is a representation of the scene that is based on the atomic graphical objects. They are described in the NDC system. Finally, the LDF is translated to the physical display file format (PDF).

As already pointed out in [**Carlbom 1980**] this model does not imply that there exist actual files that contain the LDF and PDF. The data contained in the LDF (PDF) description can either be transient. This means that they are generated and processed on the fly without intermediate storage. However, many graphics system have an explicit memory for the PDF: the frame buffer. Another possibility to implement the LDF or the PDF is to distribute them across several processing elements. This solution will be addressed later in the context of partitioning the display processor.

According to figure 1 the function of the display processor can be described by the following mapping:

$$LDF \xrightarrow{DP} PDF$$

The LDF is a low level description of the scene and consists of a sequence of object descriptions in the NDC system with associated shading information:

$$\begin{aligned} LDF \ ::= \ & \{(p_i, s_i)\} \text{ where} \\ & type(p_i) \in P \\ & model(s_i) \in S \end{aligned}$$

p_i is the geometry description of the object i. The type of that object must be one out of the set P, that contains all primitives that can be handled by the display processor. Correspondingly, s_i is the shading information for object i. The shading model must be in the set S of all shading models that can be computed by the display processor.

The PDF is a pixel oriented description of the image. It can be modelled in the following way:

$$PDF ::= \{(x, y, C)\}$$

x and y denote the coordinates of pixels in the final image (PDC system). Without loss of generality we can restrict x and y to lie inside certain intervals: $x \in [0, X_{max}]$, $y \in [0, Y_{max}]$, where $X_{max} \times Y_{max}$ is the number of pixels on the screen. C represents the colour at the given pixel. The colour is usually represented in the RGB colour system. However, it may be possible to use other colour system, e.g. HLS or CMY [**Robertson 1988**].

Using these representations of the LDF and the PDF the function of the display processor is the following mapping:

$$(p_i, S_i) \xmapsto{DP} (x, y, C)$$

This mapping together with the operations identified earlier lead to a conceptual algorithmic model for the display processor:

```
for (all objects and all pixels)
{ /* scan conversion */
  determine covered pixels ;
  /* hidden surface removal */
  determine visible pixels ;
  /* shading */
  determine colour of covered pixels ;
  /* anti-aliasing */
  perform filtering ;
}
```

The actual sequence of loops and operations is dependent on the implementation of the display processor.

3.2.2 Input

It is more difficult to model the behavior of a graphics system in reaction to external inputs than to model its output behavior. Although this is also true for display processors, the problem is less severe in this case. The only necessary input action to a display processor is to trigger a pick operation of an object at a certain pixel location. This means that the display processor must identify which primitive from the LDF is displayed at the specified pixel. We can therefore assume that the display processor accepts as an external input

the inquiry for an object identifier at a certain pixel location. The reaction to this inquiry will be that after a certain delay the object identifier will appear at the display processor's output.

3.3 A Timing Model for the Display Processor

The functional model states that the display processor has to transform all primitives from the LDF to the PDF. This means that the display processor must

- execute *every* operation
- for *every* primitive in the LDF
- in order to give a colour to *every* pixel.

In order to meet the requirements set for the entire graphics system the display processor has to complete its traversal of the LDF in a certain time. Basically, the display processor has to produce pixels. Therefore, we define the time the display processor can spend for the generation of one pixel as a measure of its speed. We call this time the pixel write time t_{pw}.

As follows from the functional model this time depends on several parameters. First the length of the LDF is important, i.e. the number of primitives in the scene. We refer to this quantity as scene complexity C_{sc}. t_{pw} also depends on the properties of the primitives. We summarise these properties in the coefficient α_{sc} which comprises e.g. the average size of the primitives (relative to the total number of pixels on the output device) and their position relative to each other [1]. α_{sc} is strongly dependent on the employed scan conversion algorithm. If the actual algorithm is insensitive to the size of objects, α_{sc} is set to 1.

The display processor must handle all pixels. The more pixels there are the less time is available for each pixel. We call the number of pixels the image complexity C_I.

Obviously, the permitted pixel write time t_{pw} can be defined only if there is a limit for the image rendering time. This time, the frame time t_f, is an upper limit for the traversal of the LDF.

The pixel write time t_{pw} is then bounded by the following inequation:

$$t_{pw} \leq \frac{t_f}{\alpha_{sc} \cdot C_{sc} \cdot C_I} \tag{1}$$

Apart from the characteristics of the scene (C_{sc} and α_{sc}) and the image (C_I) the necessary speed of the display processor also depends on the kind and number of operations to be applied to the primitives. We call this parameter the functional complexity C_F. It represents the number of steps necessary to perform the operations. The maximum time available for each of these steps, t_{step}, is an indicator for the necessary clock frequency. It is given by the following inequation:

$$t_{step} \leq \frac{t_{pw}}{C_F} \leq \frac{t_f}{\alpha_{sc} \cdot C_{sc} \cdot C_F \cdot C_I} \tag{2}$$

We consider this to be the fundamental inequation for raster display processors. The actual values of its parameters give a good estimate of the performance of a particular display processor.

[1] The relative position influences how many objects are located on one pixel. This number is sometimes called depth complexity. The performance of some HSR algorithms or anti-aliasing algorithms depends on the depth complexity.

4 A Classification for Display Processors

Based on the model developed above we will now present a way to classify different display processors. We will extract as much information as possible from the model. However, some elements of the taxonomy cannot be deduced from the model. These topics, e.g. the topology of the architecture or its regularity, are empirical and stem directly from the investigation of existing display processors.

Moreover, we will point out some implications of the choices for one or another alternative.

4.1 A Functional Classification

The functional classification takes into account only the output behavior. The input behavior is not considered because it is of minor impact on the display processor architecture.

4.1.1 Primitive Types and Scan Conversion Algorithms

These two attributes of the display processor are closely coupled because the possible scan conversion algorithm has to comply with the available types of primitives. Therefore, one of the basic decisions in the design of a display processor is to define the set P of available primitive types. Subsequent to this decision suitable scan conversion algorithms can be chosen.

The choice of primitives also influences strongly the design of the geometry processor because it must break down the high-level description of the scene into primitives.

We will now look at some possible primitives.

Pixels are the simplest drawing primitives. Providing pixels as primitives offers the rest of the graphics system a well defined access to the output device. However, breaking down a scene into pixels is a computationally intensive task. (It is actually the scan conversion process.) Therefore this primitive is never the only primitive type provided.

Vectors are frequently used drawing primitives. This is for two reasons. First, historically the first graphics systems were pure vector drawers. This gave rise to a lot of graphics algorithms that worked best with vectors. Second, vectors are the most natural geometric shape for a lot of applications, e.g. technical drawings. Furthermore, vectors can often be generated faster than other drawing primitives.

Triangles are the simplest geometric shapes that cover an area on the screen. They exhibit some very pleasant properties. Triangles are always planar and convex. Linear interpolation of colours across triangles (Gouraud shading) is invariant with respect to rotations of the triangle.

Spans are trapezoids that are bounded by the upper and lower border of a scanline. The vertical edges can have an arbitrary angle. Usually, spans are supposed to be planar. Spans are often used to render general polygons by assembling these polygons from several spans. (Spans are also called scanline segments.)

General polygons form a generalisation of the triangle primitive. In contrast to triangles, general polygons can be concave, non-planar and self-intersecting. The result

of Gouraud shading general polygons depends on the orientation of the polygon. Therefore it is advisable to use other shading algorithms with general polygons, e.g. biquadradic interpolation.

Sometimes polygons are restricted to be convex, planar and not self-intersecting. This simplifies considerably the scan conversion but it does not cure the shading problems.

Freeform surfaces are a very powerful means for object description. They offer a high degree of flexibility for controlling the shape of the surface. There are several kinds of surface representations around, interpolating and approximating ones. The more popular ones are Bézier surfaces, B-spline surfaces and NURBS surfaces.

The advantage of providing freeform surfaces as a display processor primitive is that the transformation of the high-level description of the scene into the LDF is rather simple. Furthermore, the LDF can be kept small compared to a triangle-based LDF.

On the other hand, scan conversion of freeform surfaces is difficult. Although there exist some proposals how to implement such a scan converter in hardware [**Pulleyblank et al. 1987**] [**Schneider 1988b**] this is not yet state of the art.

Halfspaces are the basic building blocks of constructive solid geometry (CSG). The boolean combination of linear and/or quadratic halfspaces can be used to generate many objects in a convenient way. The feasibility of display processors for halfspaces has been already demonstrated.

Usually, display processors can handle more than one primitive type. It is up to the designer to select those primitives that are most appropriate for the intended applications. It should be stressed again that the choice of the set of primitives is crucial because it has a strong impact on other parts of the display processor and the rest of the graphics system.

The choice of the scan conversion algorithm is a consequence of the selection of primitives. Scan conversion algorithms can be coarsely divided into two classes:

Contour tracking algorithms first identify the pixels on the borders of the primitive. Afterwards they fill all pixels in between the border pixels with the appropriate pixel values. Typical representatives for this class of algorithms are scan line algorithms that use DDA or Bresenham for computing the edges of triangles or polygons.

This method of scan conversion is applicable only to vectors, triangles, trapezoids (spans) and general polygons.

Inside testing algorithms examine (a subset of) all pixels whether they are inside the primitive under consideration. These algorithms are often very elegant from an algorithmic point of view. Unfortunately, they tend to test many pixels in vain because these pixels are outside of the primitive. For some primitives there exist algorithms that reduce the tests of empty pixels on the expenses of more complicated control structures in the algorithm [**Pineda 1988**].

Algorithms that employ inside tests are available for all primitives. They are mandatory for scan conversion and hidden surface removal using ray tracing.

One important difference between these two classes of scan conversion methods is in which order they generate pixels. The sequence of pixels produced by contour tracking

algorithms is defined by the geometry of the object to be displayed. It is therefore an unpredictable sequence of coordinates which requires the display processor to be granted random access to the output device. In particular, this means that there has to be a frame buffer.

In contrast, for inside testing algorithms the sequence of pixels to be generated can be defined. It is therefore possible to produce the pixels in the scan order of the output device. Hence, pixels may be written directly to the output device without any intermediate buffering. That becomes important when the display processor is integrated into a real-time graphics system.

4.1.2 Hidden Surface Removal (HSR)

The decision about which parts of an object are visible to the viewer is an essential step for the creation of realistic images. Plenty of algorithms have been presented to solve the visibility problem, for both vector and raster displays [**Sutherland et al. 1974**] [**Latham 1985**] [**Lakshminarasimhan et al. 1989**].

The choice of the HSR algorithm determines which capabilities the display processor will have. Not all HSR algorithms are equally well suited to handle e.g. penetrating objects or transparency. Moreover, the memory requirements vary significantly for the different algorithms.

As pointed out in [**Sutherland et al. 1974**] HSR is essentially a sorting process. Therefore, many HSR algorithms require an extensive sorting step prior to the actual HSR. The order of sorting in the directions of x, y and z is unimportant. However, the various algorithms may be of different efficiency with regard to the amount of coherence exploited during the sorting.

A major distinction between different HSR algorithms is whether they use priorities or real depth for the visibility decision. The priority is assigned to an object once. In contrast, the depth is changing across the object and has to be computed for every pixel.

Since priorities define a partial order over the objects it is impossible to handle cases of penetrating objects. Such constellations must be eliminated by subdividing the objects along the intersections.

According to [**Lakshminarasimhan et al. 1989**] HSR algorithms fall into three major categories.

Object space algorithms. [2] HSR in object space is usually done in the geometry processor. The result is a number of primitives that are non-overlapping in object space. They are forwarded to the display processor. Hence, the display processor is not concerned with the HSR.

List-priority based algorithms in fact solve the visibility problem outside the display processor. The primitives are presorted according to their priority (either in priority order or in reverse priority order). The display processor then writes the primitives in that order to the output device. Hence, the HSR in the display processor is quite easy.

The *painter's algorithm* is based on a reverse priority ordering of the objects. This guarantees that the object with highest priority is written last and therefore lies in

[2] In the context of HSR *object space* means that the accuracy of the visibility calculations depends on the precision of the object representation in the SDF. This meaning of *object space* must not be confused with the notion of *object space partitioning* defined in this paper.

front. Translucency can be incorporated into the painter's algorithm easily by mixing the object's colour with the previous pixel colour. Anti-aliasing is not straightforward; problems exist to handle the inner edges of polygon meshes.

When using a *sorting in priority order* the frontmost object is displayed first. The pixels covered by that object are tagged. Objects with a lower priority can only write to untagged pixels. Thus, each pixel is written only once. This reduces the number of accesses to the display significantly compared to the painter's algorithm. Both, anti-aliasing and translucency are non-trivial tasks for this method.

Image space algorithms. [3] solve the visibility problem with a precision that depends on the screen resolution. According to [**Lakshminarasimhan et al. 1989**] there are four basic types of such HSR algorithms:

- **Scanline algorithms** have been developed mainly to handle polygons together with contour tracking scan conversion algorithms. They first identify for every scanline which objects are contributing to this scanline. The edges of these objects are kept in an active-edge-table. For every scanline, these edges are sorted in x (scan order). Every time an edge is crossed while scanning a scanline it is decided which of the active polygons is nearest to the viewer.

  If the algorithm is implemented as sketched out, it is unable to process penetrating objects; it is then mainly priority based. However, it is possible to extend the algorithm to handle such cases. Translucency can be incorporated in scanline algorithms.

- **Depth-buffer** is the most popular HSR technique in today's high-performance graphics systems. For every pixel, depth and colour are stored. Every object writes its colour only to those pixel whose depth is greater than the object's depth. Thus, the visibility problem is solved pixel by pixel. Hence, the depth buffer algorithm is able to handle intersecting objects without extra effort.

  Transparent objects can be treated correctly only if they are written to the buffer last. This requires an extra sorting step. It is very difficult to incorporate good anti-aliasing into the depth buffer-algorithm. However, there exist extensions to the depth-buffer that address this problem, e.g. [**Carpenter 1984**] [**Porter et al. 1984**].

- **Area sorting** subdivides the screen until a specified level of subdivision has been reached or the whole subscreen is covered by only one primitive. The quadtree representation of images uses this algorithm.

  Incorporation of translucency into this concept is a non-trivial task. There are difficulties to produce anti-aliased pictures with the area sorting approach.

- **(Primary) ray tracing** is a technique which shoots through every pixel a ray of light into the scene. The object hit first by the ray is visible to the viewer. This technique requires a scan conversion algorithm of the inside test class. Note that ray tracing has been developed to model effects like multiple reflections and refractions of light at objects. The computational complexity only pays off if this is desired. Ray tracing can easily handle transparency and penetrating objects. Also anti-aliasing can be integrated readily into this method.

[3] The term *image space* used in the context of HSR differs from that introduced later on in this paper. Here *image space* only means that the precision of the HSR is a consequence of the output device.

4.1.3 Shading

In order to give computer generated images a realistic appearance the illumination of the scene must be modelled. Since the circumstances of illumination in the real world are very complex shading of scenes in computer graphics always simplifies the physical world to some extent.

In general, shading in computer graphics means to compute the colour of an object at certain points. The method, that computes this colour is called *shading model*. The *shading algorithm* specifies at which points of the object the shading model is applied. (The distinction between models and algorithms is somewhat arbitrary. Models and algorithm are often related to one another.) Since the very beginning of computer graphics shading models and algorithms have been proposed. A good survey of models and algorithms is given in [**Claussen 1988**]. Here, we will not discuss the various shading models. Only the most important shading algorithms will be enumerated.

Wash shading is the simplest algorithm. It assigns a fixed colour to every primitive. This colour does *not* depend on any light sources in the scene. This algorithm is very simple but does not give realistic images.

Flat shading assigns one colour to the whole primitive. In contrast to the wash shading algorithm this colour has been calculated in dependence of the illumination of the scene.

Gouraud shading interpolates colours from one point of the primitive to another point. The colours at the two points are computed in the geometry processing step.

Usually Gouraud shading employs linear or bilinear interpolation of colours from one vertex of a polygonal primitive to another vertex. The shading model used to compute the vertex colours can be chosen freely.

Phong shading specifies that the normal vector of a primitive has to be interpolated from one point on the primitive to another point. For every step the interpolated normal vector together with other attributes of the object (reflectance, glossiness, etc.) is used to compute the object's colour.

As with Gouraud shading, Phong shading was originally intended for polygonal primitives. Then, the normal vector is (bi)linearly interpolated from vertex to vertex. Although, there is a Phong shading model this model is not necessarily tied to the Phong algorithm.

Ray tracing is (apart from determining visible objects) used to find out where shading has to take place. Since ray tracing can provide much information about the scene's illumination very demanding shading models can be employed.

4.1.4 Anti-aliasing

Aliasing is a consequence of determining the colour of the entire pixel by sampling the pixel only at its center or too few sampling points in the pixel. In static pictures these anomalies are the well known jagged edges and thin lines broken into parts. More deficiencies appear in animated pictures: small blinking objects, objects jumping from scanline to scanline and "running ants" on jagged edges.

Although, in practice, a perfect anti-aliasing is impossible [**Blinn 1989a**] [**Blinn 1989b**] there are ways to cure the worst aliasing effects. This can be done either by increasing the sampling frequency or by taking into account that a pixel covers a non-zero area. The latter methods try to compute how much of a pixel is covered by the contributing objects.

The various methods differ in the number of passes they take to produce the anti-aliased image. There are one-pass and two-pass techniques. Two-pass techniques first render the entire image and generate additional information about the image, e.g. contribution of an object to a pixel. In the second pass this information is used to enhance the picture. In contrast, one-pass techniques try to approximate this additional information. They generate the final pixel colour immediately. One-pass techniques exhibit serious drawbacks if the LDF is not preprocessed properly, e.g. sorting the objects from front to back.

A further distinction between different anti-aliasing techniques is whether the anti-aliasing is adaptive or not. Anti-aliasing is necessary only at the contours (edges) of objects. However, many implementations do not explicitly search for edges but process the whole picture in a uniform manner, e.g. supersampling.

Processing only the contours of objects is correct only if there are no intersections of objects. Some anti-aliasing techniques are not able to handle penetrating objects.

We will now describe some common anti-aliasing techniques in display processors.

The gz-buffer is a one-pass technique [**Ghazanfarpour et al. 1987**]. It is basically a depth buffer with another parallel geometry-buffer, the g-buffer. The g-buffer contains information about the current coverage of a pixel. This information is used if the edge of another object is running through that pixel. Then, the colour of one of the four neighbor pixels is mixed with the colour of the object in order to obtain the new pixel colour. Also the new contents of the g-buffer is computed.

The gz-buffer is able to handle penetrating objects. It is restricted to at most one edge, i.e. two objects, per pixel.

Compositing is a one-pass technique, too [**Porter et al. 1984**]. For every pixel in addition to the colour an alpha channel is sustained. The value of alpha represents the fraction of the pixel that is covered by a particular object. Also the depth of the object at the four corners of the (square shaped) pixel is evaluated.

Correspondingly, for every pixel the depth values at its four corners are stored. Using the depth values at the pixel corners it is determined how the new and the old pixel overlap each other. The value of alpha and the pattern of overlap are combined to the new pixel colour and the new alpha value.

Supersampling means that one pixel is sampled at several points. For each of these subpixels depth and colour are computed. That high resolution picture is then compressed to the resolution of the output device by averaging all subpixels contributing to a pixel. Although supersampling is still a point sampling technique it produces very good results since the aliasing effects are diminished by the averaging step. Supersampling can be easily combined with all scan conversion, shading and HSR algorithms because the averaging step is completely independent from the image generation step. Unfortunately, supersampling is very expensive with respect to both time and memory.

The A-buffer is an extension of the depth buffer [**Carpenter 1984**]. Instead of storing only depth and colour of the frontmost object the A-buffer stores depth and colour of all objects that are potentially contributing to a particular pixel. (The depth of the objects has been determined at the pixel center.) These objects are stored in a depth sorted list. For every object in the list a subpixel mask is computed. The subpixel mask specifies which subpixels are covered by an object.

In a second step, for every subpixel the visible object and its colour are computed. These subpixel colours are then averaged in order to obtain the final pixel colour.

The A-buffer technique has the drawback that its requirements for time and memory are not constant. They depend on the statistical properties of the scene to be rendered.

Filtering is not a real anti-aliasing technique. It combines certain pixels in the neighborhood of the pixel under consideration to determine the pixel colour. The effect of a low-pass filter is only a blurring of the picture.

4.2 Classification by Partitioning

The fundamental inequation for raster display processors (2) indicates that display processors have to parallelise the image generation process in order to achieve reasonable performance.

A short example will explain this: For a screen with 1000×1000 pixels, a scene with 10,000 primitives covering 1000 pixels each ($\alpha_{sc} = 0.001$) and a frame time of 0.1 s we obtain ($C_F = 1$):

$$t_{step} \leq 10ns$$

This time is just within the limits of today's technology. More complex scenes, screens with a higher resolution or shorter frame times will demand shorter step times. Therefore it is natural to search for ways to introduce parallelism into the display processor operation.

The number of steps necessary to generate an image is fixed. A partitioning describes how these steps can be distributed among a number of processor elements that are working in parallel on different steps. The result of a partitioning will be a rule how to construct such processor elements. The display processor is then constructed from *display sub-processors*. t_{step} for each of these sub-processors is greater than the step time for the display processor itself.

We will now discuss the different possibilities to partition the display processor. The inequation (2) indicates how the necessary partitioning can be accomplished. Every parameter stands for another domain of partitioning.

4.2.1 Partitioning in the Time Domain

In the inequation (2) the time domain is represented by the frame time t_f. Enlarging t_f for the display sub-processors means that each of these sub-processors has more time available to perform its operations. In order to sustain the performance of the display processor several sub-processors have to work in parallel on different, subsequent tasks. While one sub-processor is working on the task T_i the next sub-processor is processing task T_{i+1}.

4.2.2 Partitioning in Image Space

The image complexity C_I stands for the number of pixels. The time t_{step} can be enlarged for the sub-processors by distributing subsets of all pixels to the sub-processors, i.e. C_i is reduced.

There are several ways to achieve this. A single display sub-processor can either work on single pixels, on complete scanlines or on contiguous parts of the screen like stripes or windows. Another possibility to partition the screen is to interlace the pixel sets that the different sub-processor are treating [**Parke 1980**] [**Hu et al. 1985**]. The association of sub-processors and certain pixels can either be static or dynamic.

This partitioning concept implies that either all objects are broadcast to all sub-processors or that the objects are sorted according to that part of the screen they are contributing to.

The image space partitioning approach has potential drawbacks for images where the objects are not distributed evenly across the image. Then the workload may be unbalanced among the sub-processors.

Image space partitioning is often reflected in the organisation of the image memory, i.e. the frame buffer.

4.2.3 Partitioning in Object Space

Reducing the scene complexity C_{sc} means to off-load the sub-processors from a part of the objects. Every sub-processor, a so-called object processor, is handling a subset of the LDF. That subset can contain either a single primitive or several primitives. These primitives may be independent of each other. Alternatively, the primitives may all belong to a higher-order object that has been defined by the application (AM).

A potential problem with object space partitioning is that the processing time for different primitives may vary. (For instance, processing bigger primitives could take more time than small objects.) Since, in practice, these differences can be only estimated, some object processors may run idle while others are overloaded.

A special kind of object space partitioning is the partitioning in the parameter space. This strategy is frequently used when parametric surfaces must be displayed by a display processor that can only handle e.g. triangles.

4.2.4 Partitioning in the Functional Domain

The individual display sub-processors are less burdened by their task if they have to perform less operations in a certain time period. This means that the parameter C_F is made smaller. (C_F is the functional complexity and represents the number of steps to be performed for each pixel). This can be achieved by distributing the various operations amongst the display sub-processors.

The different sub-processors are not performing the same function and are therefore probably not of the same type [4]. This can be a drawback because the effort to design such a display processor is bigger than for the other partitioning strategies. The fact that there are different types of sub-processors leads to a lower regularity of the display processor (see below).

[4] It is possible to implement different functions on identical processor elements. This can be achieved e.g. by means of (micro)programming. A good example for such a solution is the system described in [**Clark 1982**].

4.2.5 Multi-level Partitioning

The partitioning schemes presented so far are pure solutions in the sense that they partition the display processor's task only in a single domain. In contrast, real systems often employ a combination of different problem partitioning approaches. We can distinguish here such solutions which divide the problem several times in the same domain and solutions that partition in different domains.

Looking at systems that employ a multi-level partitioning leads to the following observation: Each system partitions the image generation either in image space or in object space. We call these partitioning schemes *primary partitioning*. Partitioning the time domain or the functional domain are auxiliary or *secondary partitionings*. The secondary partitionings are used to efficiently implement one of the primary problem subdivision strategies.

4.2.6 Notation

We will now introduce a notation that expresses how a certain display processor partitions its task. This notation is an extension of the scheme proposed in [**Reghbati et al. 1988**].

We represent the domain in which the partitioning takes place by the first letter of its name (F, I, O, T). In parenthesis we put two arguments: The first tells in how many parts the domain is divided, i.e. how many sub-processors are working in parallel in that domain. The second parameter states the maximum number of elements in each partition, i.e. how many elements each sub-processor handles. Elements can be e.g. pixels, scanlines, or primitives.

For instance, the expression

$$I(4096, 256)$$

means that the display processor partitions the image space in 4096 subimages each containing 256 e.g. pixels.

Multi-level partitioning schemes are represented by catenating the different partitioning levels with a 'o', e.g.

$$I(16, 64) \circ I(4, 64 \times 256)$$

can be the description of a system that divides the screen into 16 groups of 64 scanlines each. Each group is further partitioned into 4 parts that are handling 256 pixels of each of the group's 64 scanlines.

If there are different kinds of domain partitioning on the same level they are separated by a '||'.

4.3 Classification by Architecture

A major issue for classifying display processor (and computers in general) is how the algorithms have been mapped onto a hardware structure. To categorise general computer architectures is a difficult task on its own. Several proposal have been made in the past [**Giloi 1983**]. However, they do not concentrate on what we consider to be important and descriptive for display processor architectures. We will therefore highlight only some useful features that are also generally accepted as descriptive for computer architectures. (All features assume that display processor is a multiprocessor.)

Topology of the network. The topology gives an idea about the physical construction of the processor. Possible topologies are pipelines, two-dimensional arrays, arrays in more than two dimensions (hypercubes), tree structures, etc.

Different architectures can be further distinguished by their capability to be configured dynamically, i.e.the topology can be adapted to different applications.

Control of the network. A multiprocessor network can be controlled by a central master. Alternatively, a decentralised control is distributed across a network of processors having equal rights. Furthermore, the control can be classified according to the manner how commands are distributed inside the network. The commands can be either broadcast simultaneously to all processor or propagated from one processor to the next.

The communication scheme is a property that is strongly related to the control structure. In the context of a VLSI design it is preferable to employ local communication structures instead of global ones.

Data types. Architectures can be distinguished with respect to the type of data propagated through the network. The classification of the data types falls into two categories. The first category is the kind of data, e.g. pixels, objects, colours, etc. The second category states how the data is represented. Possible representations are integers, floats, lists etc.

Type of parallelism. In a multiprocessor there can be parallelism in the execution of different commands and/or the processing of different data. Such processors are called single/multiple instruction machines with single/multiple data processing (SISD, MISD, SIMD, MIMD).

Parallelism can be further classified as coarse grain parallelism or fine grain parallelism.

Regularity. By regularity we understand the number of identical elements in relation to all elements. The more regular an architecture is the better it is suited for a VLSI implementation. We will now give a somewhat intuitive definition of regularity:

Let be N the number of all processing elements in the architecture and T the number of types of processing elements, e.g. the number of different chips. By n_i we denote the number of processing elements of type i.

We define the regularity R as the product of two factors r_1 and r_2:

$$\begin{aligned} r_1 &= 1 - \frac{T-1}{N} \\ r_2 &= \begin{cases} 1 - \sqrt{\frac{1}{T-1}\sum_{i=1}^{T}\left(\frac{1}{T} - \frac{n_i}{N}\right)^2} & \text{für } T > 1 \\ 1 & \text{für } T = 1 \end{cases} \\ R &= r_1 \cdot r_2 \end{aligned}$$

r_1 takes into account only the size of the architecture and the number of different chips. It exhibits the following properties.

- $T = 1 \Rightarrow r_1 = 1$
  The architecture is most regular if there is only one type of element.
- $N = T \Rightarrow \lim_{T\to\infty} r_1 = 0$
  If all elements are different the regularity is the smaller the more elements there are.
- $0 < r_1 \leq 1$

- The more types there are in the architecture the lower the regularity.
- The regularity is higher if there are more elements in the architecture.

r_2 reflects how many instances of each chip type there are in the architecture. We chose r_2 as the inverted standard deviation of the normalised population (n_i/N) of the type classes i. Hence it has the following properties:

- $n_i = n_j = \frac{N}{T} \; \forall \; 1 \leq i, j \leq T, \; i \neq j$
  $\Rightarrow \max(r_2) \;\; , \;\; r_2 = 1$
  If the number of elements in the different classes is the same, the regularity has a maximum.
- $n_1 = N - T + 1, n_i = 1 \; \forall \; 1 < i \leq T$
  $\Rightarrow \; \min(r_2)$
  If the elements of the different types are distributed totally unequally, r_2 has a local minimum.
- $0 < r_2 \leq 1$

The first two properties of r_2 reflect the idea that structures look more regular if all elements appear with the same quantity.

4.4 Classification by Performance

Performance can be classified along different guidelines.

Characteristic parameters. The fundamental inequation (2) for raster display processors gives the maximum time for each operation of the display processor (t_{step}). This time can serve as a common basis for comparing and classifying the performance of different display processors. The bigger the time t_{step} may be (for equal functions) the easier such a display processor can be implemented with available technology.

Primitives per time period. It is very frequent to specify how many primitives can be processed by a particular display processor in certain period of time. Usually, this time period is either 1 second or one video frame. The video frame time is often set to 1/30 second or 1/60 second.

Delay time. The time between issuing a command to the display processor and obtaining visible results is called delay time. This time, although often not specified, is important to judge the ability of a graphics system to conveniently interact with the user.

5 Classification of some Existing Display Processors

This section applies the developed taxonomy to some display processors that have been published in the last years. The intention of this section is not to give a survey of existing display processors.

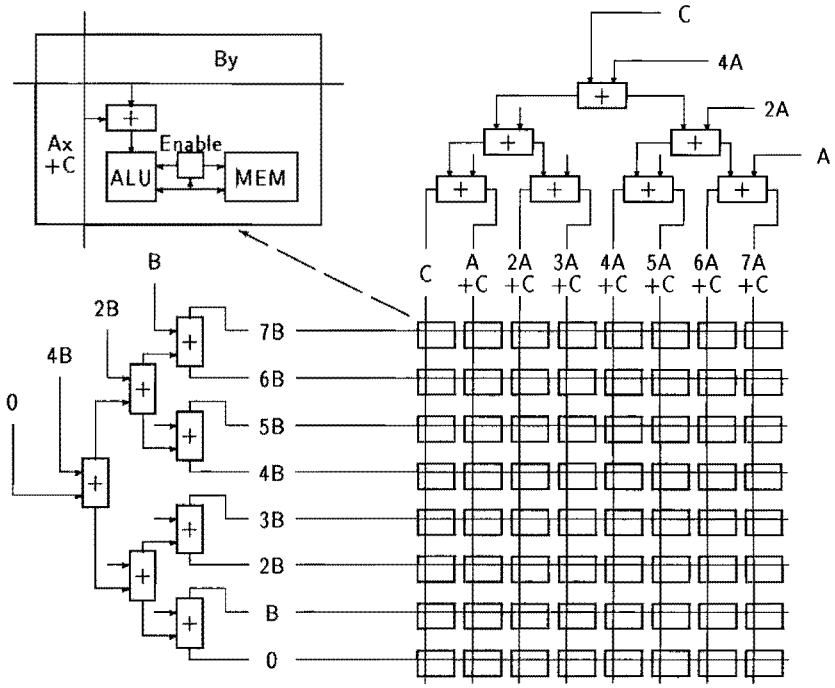

Figure 2: **Conceptual block diagram of the Pixel-planes display processor.**

5.1 Pixel-planes

The Pixel-planes system is one of the best documented display processors in the literature [**Fuchs et al. 1982**] [**Poulton et al. 1985a**].

Figure 2 shows the principal construction of the Pixel-planes architecture ([**Fuchs et al. 1985**]). It is essentially a frame buffer whose pixel cells have been enhanced by some extra logic. This logic enables Pixel-planes to compute the expression $Ax + By + C$ in parallel for all pixels (x, y). Evaluation of this linear expression with different sets of parameters A, B and C enables the Pixel-planes system to render z-buffered Gouraud-shaded polygons.

5.1.1 Functional Classification

Primitive Type(s). Convex, planar polygons [5].

Scan Conversion. Inside test based on a polygon representation by the bounding lines of the polygon.

[5] In [**Fuchs et al. 1985**] it is shown how the Pixel-planes system can be used to render circles and spheres.

Hidden Surface Removal Depth-buffer.

Shading. Gouraud shading (interpolation of *precomputed* vertex colours).

Anti-aliasing. None [6].

5.1.2 Partitioning

Pixel-planes parallelises the image generation process by one-level image space partitioning: $I(512 \times 512, 1)$.

5.1.3 Architecture

Topology. The two main building blocks of the Pixel-planes system are two binary trees of bit-serial adders and an array of pixel processors. The outputs (leaves) of the binary trees provide partial sums of the expression $Ax + By + C$ to the pixel cells.

Control. The whole Pixel-planes system is controlled by a central control unit on each Pixel-planes chip [**Poulton et al. 1985b**]. The controllers receive the commands from the outside world – either from the geometry processor that issues drawing commands or from the scan-out circuitry that reads out the image from Pixel-planes.

Data types. The data fed into Pixel-planes are always the parameters A, B and C of a linear expression, describing either edges, or depth or colours. A, B, C are integers and are supplied in bit-serial format to the Pixel-planes system.

Parallelism. The commands and parameters to Pixel-planes are the same for all pixel processors. The values of the linear expressions vary from pixel to pixel. Thus, all pixel processors are operating with different data. Therefore, Pixel-planes is a SIMD display processor.

Regularity. The Pixel-planes employs only one type of chip. It is therefore highly regular ($R = 1$). The Pixel-planes IV system is built from 2048 identical chips each of them storing 128 pixels [**Fuchs et al. 1988**].

5.1.4 Performance

Characteristic parameters. The following numbers are based on the assumption that Pixel-planes is rendering triangles. Then seven steps are required to render one triangle (3 edges + 1 depth + 3 colours). Some of the numbers are taken from the literature. It should be noted that the speed of Pixel-planes is independent of the actual size and location of the triangles ($\alpha_{sc} = 1$).

$$\begin{aligned} t_f &= 1\,s \\ \alpha_{sc} &= 1 \\ C_{sc} &= 35,000 \\ C_P &= 7 \\ C_I &= 1 \end{aligned}$$

Using the inequation (2) we obtain $t_{step} \leq 4\ \mu s$

[6] [**Fuchs et al. 1985**] explains how Pixel-planes can perform anti-aliasing by either supersampling or employing subpixel masks.

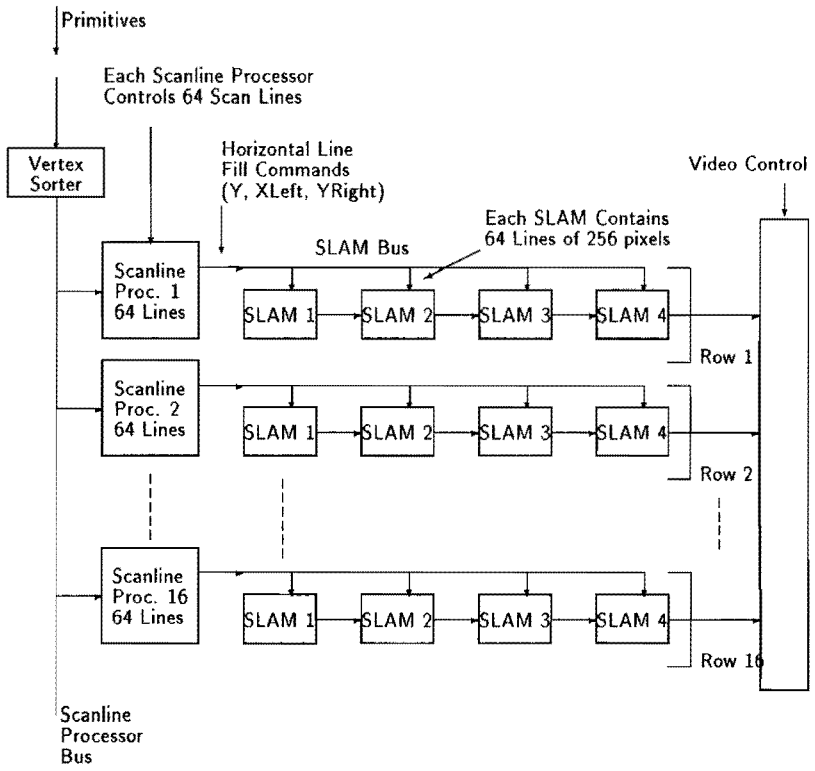

Figure 3: **Block diagram of the SLAM display processor.**

Primitives per second. Pixel-planes IV can process about 35,000 Gouraud-shaded, z-buffered triangles per second. Assuming that there are 30 frames per second, these are nearly 1,200 triangles per frame.

Delay time. The delay time of Pixel-planes is not specified in the literature. However, the working principle of Pixel-planes suggests that the delay is exactly the time necessary to process one polygon ($\approx 30\ \mu s$).

5.2 SLAM

In [**Demetrescu 1985**] a display processor has been presented that uses two different kinds of subsystems (figure 3): *Scanline processors* break down the incoming polygon descriptions into spans.

The spans are described by the scanline they are covering and the coordinates of the first and the last pixel in that scanline. Each scanline processor is responsible for 64 scanlines.

These spans are dispatched to the *scanline access memories* (SLAMs). Each SLAM fills the specified span with a downloaded pixel pattern. By specifying pixel patterns

characters can be displayed. Each SLAM chip contains the memory of 64 scanlines with 256 pixels each. All 256 pixels in one scanline can be accessed, processed and stored in one step. SLAMs can be cascaded to construct longer scanlines.

5.2.1 Functional Classification

Primitive Type(s). Y-monotone polygons [7] and characters. (Lines are drawn as thin polygons.)

Scan Conversion. Contour tracking. The scanline processor computes the start and end coordinates of the polygon for every scanline. The span is filled by the SLAMs.

Hidden Surface Removal. HSR is not addressed in the published material. It is supposed that the system is implementing some kind of priority based scan conversion algorithm, e.g. painter's algorithm, i.e. the HSR took place already in the geometry processor.

Shading. Halftone patterns. Since this display processor uses only one bit per pixel no colour shading is provided.

Anti-aliasing. None.

5.2.2 Partitioning

This display processor employs a three-level image space partitioning. The main partitioning is in image space; it occurs two times: First, the image is subdivided into groups of 64 scanlines each. In a second step these groups are then split into sets each of which contains 64×256 pixels.

Image generation is also partitioned in the functional domain. Vertex sort, producing scanline commands and actual pixel generation are allocated to distinct processors.

The partitioning is therefore characterised as:

$$F(3,1) \circ I(16,64) \circ I(4,64 \times 256)$$

5.2.3 Architecture

Topology. The topology of this display processor is a tree. The root of the tree is formed by the vertex sorter that is connected to the input of the display processor. The scanline processors constitute the first level of the tree. The leaves of the tree are formed by the SLAMs.

Control. This display processor employs a distributed control scheme. Both, scanline processors and SLAMs have their own control unit. The communication between them is not detailed in the literature.

Data types. The scanline processors receive the description of polygons, lines or characters. The primitives are transformed into spans that represented by the x-coordinates of the right and left edge of the span and its y-coordinate. A halftone pattern is provided that is used for filling the span.

The literature does not give details about the representation of the various data.

[7] Y-monotone polygons are characterised by the fact that horizontal lines intersect the boundary of the polyon at most twice. Convex polygons are special cases of monotone polygons. They are monotone in x and y.

Parallelism. All scanline processors receive different primitives from the geometry processor. The commands and data that are generated for each SLAM are different. This characterises a MIMD machine.

Regularity. The display processor is built from two main building blocks ($T = 2$): the scanline processors and the SLAMs. In the following it is assumed that one scanline processor can be integrated on a single chip [**Demetrescu 1985**]. The standard configuration of scanline processors and SLAMs uses 16 scanline processors each of which controls 4 SLAMs. This results in a total chip count of $N = 80$. The regularity is therefore $R = 0.99 \cdot 0.58 = 0.57$.

5.2.4 Performance

The display processor is composed of two different types of chips, scanline processors and SLAMs. These two should be evaluated separately. However, since the scanline processors have been described not very detailed we concentrate on the SLAMs.

The following numbers have been derived from figures given in the literature. (The original paper specified the performance in terms of memory accesses.)

Characteristic parameters. We assume that triangles are processed by the display processor. Since the number of pixels covered by a primitive influence the number of steps to scan convert that polygon the performance of that display processor depends on the properties of the image. Therefore no exact value for α_{sc} can be given. It is estimated for an average size of a primitive of 5,000 pixels and a screen size of 1024 × 1024 pixels.

$$\begin{aligned} t_f &= 1\ s \\ \alpha_{sc} &\approx 0.005 \\ C_{sc} &\approx 50,000 \\ C_F &= 2 \\ C_I &= 64 \end{aligned}$$

This values give a maximum step time of

$$t_{step} \leq 32\ \mu s$$

Primitives per second. [**Demetrescu 1985**] claims that a SLAM can perform approximate 5 million scanline accesses per second. If we make the (conservative) assumption that in average 20 percent of a scanline are covered by a primitive. If we further assume that an object covers 5,000 pixels in average we can estimate the number of primitives processable in one second: 50,000 primitives per second.

Delay time. There is no delay time specified in the literature.

5.3 PROOF

The PROOF system, proposed in [**Schneider 1988a**], is built from three main blocks (figure 4 [**Schneider et al. 1988**]).

The first one, the object processor pipeline (OPP), is made up of a number of *object processors (OP)* storing one object each. Each OP scan converts its object and performs

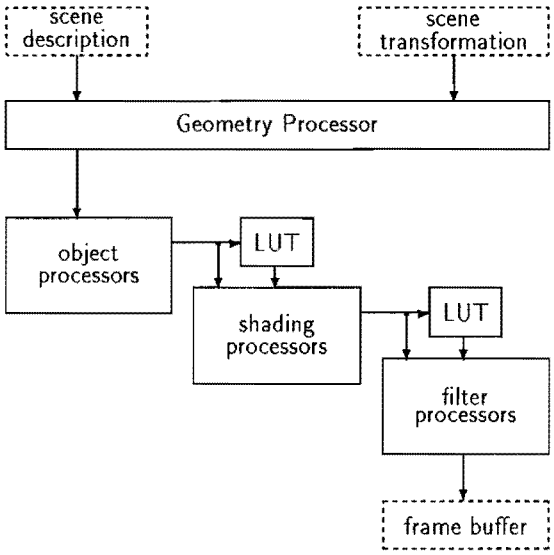

Figure 4: **Block diagram of the PROOF display processor.**

the HSR on a pixel basis. The OPs also interpolate, depending on the configuration of PROOF, colours in RGB or normal vectors.

The next stage in PROOF is a *shading stage* that performs Phong shading. The shading stage is a pipeline of shading processors (SP). Each two SPs are handling one lightsource.

The last stage in PROOF is the *filter stage* that performs the anti-aliasing executing an A-buffer algorithm. There is one processor for each subpixel.

5.3.1 Functional Classification

Primitive Type(s). Triangles and vectors.

Scan Conversion. Inside test. The triangles are described by their bounding lines.

Hidden Surface Removal Depth-buffer (A-buffer).

Shading. Flat and Gouraud shading (OPP only). Phong shading (with shading stage).

Anti-aliasing. A-buffer on a 8×8 subpixel grid.

5.3.2 Partitioning

The PROOF architecture partitions the image generation process in several domains. The first is the functional domain. Each stage carries out another set of functions. The OPP performs scan conversion, HSR and (limited) shading. The shading stage and the filter stage perform illumination and anti-aliasing respectively.

Each stage partitions its task again. The OPP and the shading stage allocate objects (primitives or light sources) to the different processors (object space partitioning). The filter stage provides one processor for each of the 64 subpixels (image space partitioning). This partitioning scheme is represented by

$$F(3,3) \circ [O(50000,1) \| O(16,0.5) \| I(64,1)]$$

5.3.3 Architecture

Topology. The organisation of PROOF is a pipeline. The OPP and the shading stage are pipelines themselves. The filter stage is constructed from two trees of adders that compute coverage information for every subpixel [**Romanova 1988**]. The output of these trees is fed into an array of subpixel processors that compute the colour of the subpixels.

Control. There is no central control in PROOF. All processing elements, OPs, SPs and FPs, have their own controllers. The different processors are communicating via an asynchronous protocol.

Data types. Information for pixels is propagated through the various pipelines. In the OPP and the shading stage a list of objects is associated with every pixels. The objects in the list are described by an object identifier, depth and normal vector (or colour value) at the current pixel and geometry of the coverage for the current pixel. The filter stage produces a RGB triple. All data items are describe in fixed point format.

Parallelism. In total PROOF is a MIMD machine. The single stages are SIMD processors. All processors in one stage are executing the same instruction (may be delayed due to the pipeline structure) on different data.

Regularity. OPP and shading stage consist only of one type of processing elements, namely the OPs and SPs. Five OPs and two SPs are expected to reside on the same chip. For a system that handles 50,000 objects and 16 light sources 10,000 OP chips and 16 SP chips are needed.

The filter stage uses two types of processing elements; processor slices for the adder trees and subpixel processors. Each of the x-, y- and z-trees will be distributed on three chips. Four subpixel processors will be integrated on one chip. For a resolution of 8×8 subpixels 16 subpixel processor chips are needed.

PROOF is built from four types of chips ($T = 4$) and approximately $N = 10,000$ chips. The population of the chip classes is $n_1 = 10,000$, $n_2 = 16$, $n_3 = 9$ and $n_4 = 16$. From this we obtain $R = 1.0 \cdot 0.50 = 0.50$.

5.3.4 Performance

We assume a screen resolution of 1024×1024 pixels. The average depth complexity may be 2. (The depth complexity directly influences the length of the lists constructed for the A-buffer.)

Characteristic parameters. 1. Object Processors

$$\begin{aligned} t_f &= 33\,ms \\ \alpha_{sc} &= 2 \end{aligned}$$

$$\begin{aligned} C_{sc} &= 1 \\ C_F &= 3 \\ C_I &= 1024 \times 1024 \end{aligned}$$

This gives a step time of $t_{step} = 15.73\ ns$ which is equivalent to a clocking frequency of 63.5 MHz.

2. Shading Processors

$$\begin{aligned} t_f &= 33\ ms \\ \alpha_{sc} &= 2 \\ C_{sc} &= 1 \\ C_F &= 5 \\ C_I &= 1024 \times 1024 \end{aligned}$$

This results in a step time of $t_{step} \leq 3.15\ ns$. To enlarge the step time for the individual shading processors, time domain partitioning with up to 16 shading processors in parallel will be employed. This will give a step time of $t_{step} \leq 50\ ns$.

3. Subpixel Processors

$$\begin{aligned} t_f &= 33\ ms/(1024 \times 1024) \\ \alpha_{sc} &= 2 \\ C_{sc} &= 1 \\ C_F &= 7 \\ C_I &= 1024 \times 1024 \end{aligned}$$

Since the step time is about 2 ns up to 10 parallel processors will be working in parallel. Thus the step time is enlarged to about 20 ns.

Primitives per second. In [**Schneider 1988a**] a throughput of 50,000 triangles every 33 ms has been announced.

Delay time. The literature gives a worst case delay time of 10.5 ms for a complete image update.

5.4 The Ray-Casting Machine of Kedem & Ellis

The Ray-Casting Machine [**Kedem et al. 1984a**] [**Kedem et al. 1984b**] aims at the scan conversion of objects defined by constructive solid geometry (CSG). In CSG objects are described by set operations, e.g. union, intersection or difference, applied to halfspaces.

The Ray-Casting Machine is built from two types of processors (figure 5). The *primitive classifiers (PC)* calculate the intersection of a ray (a line) with a primitive, i.e. classifies the primitive with respect to the line. Each PC holds one primitive and accepts rays for classification to the stored primitive. The classification returns segments of the ray that are inside or outside of the primitive. The *classification combiners (CC)* combine two classified rays using a certain set operation.

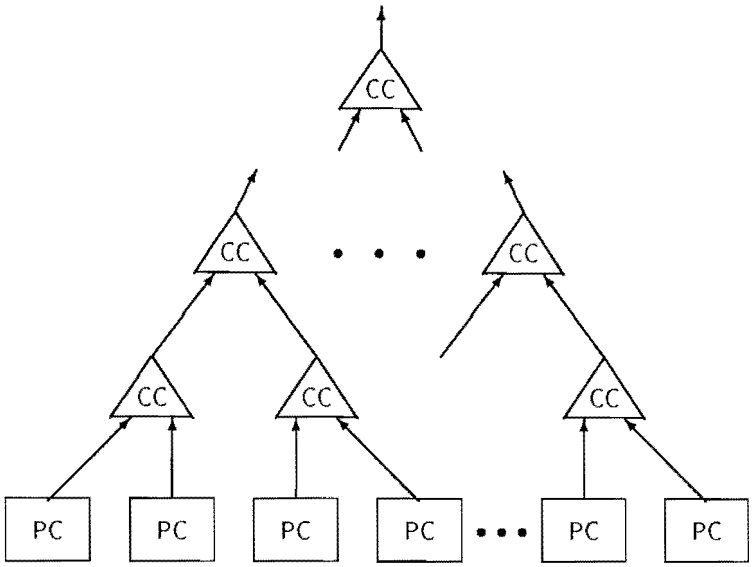

Figure 5: **Block diagram of the Ray-Casting Machine.** PC – Primitive Classifier, CC – Classification Combiners.

5.4.1 Functional Classification

Primitive Type(s). Linear and quadratic halfspaces.

Scan Conversion CSG with ray tracing. The rays into the scene are restricted to be all parallel.

Hidden Surface Removal. Ray tracing.

Shading. None.

Anti-aliasing. None.

Shading and anti-aliasing could be added as a postprocessing step that uses the results of the Ray-Casting Machine.

5.4.2 Partitioning

The Ray-Casting Machine employs an object space and a functional domain partitioning. The object information is distributed to the PCs and CCs. These two types of processors also constitute the functional subdivision: One performs primitive classification the others combine classifications.

The partitioning is characteried as

$$F(2,1) \circ (O(n,1) \| O(n \log_2 n, 1))$$

if a maximum of n primitives have to be handled.

5.4.3 Architecture

Topology. The topology is a binary tree. The leaves of the tree are the PCs and the nodes are the CCs. The Ray-Casting Machine can be configured dynamically to the CSG descriptions of different objects. The configuration takes place by loading the CCs such that they implement the required CSG tree.

Control. The control of the Ray-Casting Machine is distributed among all processing elements, PCs and CCs. The processing elements are communicating synchronously using a complete handshake protocol.

Data types. Data moving through the processor tree represent line segments classified as being IN our OUT of the CSG object. These segments are described by integer numbers.

Parallelism. All PCs together form a SIMD computer. The CCs are also working in SIMD mode.

Regularity. The Ray-Casting Machine is built from only one type of chip. Such a chip contains from 4 to 8 slices each of which consists of one PC and 9 CCs. The chip level regularity of the Ray-Casting Machine is therefore $R = 1$.

5.4.4 Performance

The literature does not contain any performance numbers for the Ray-Casting Machine. We can therefore summarise here only those numbers that are a consequence of the architectural features.

Characteristic parameters. 1. Primitive Classifiers

$$\begin{aligned} \alpha_{sc} &= 1 \\ C_{sc} &= 1 \\ C_I &= 1024 \times 1024 \\ C_F &= ? \\ t_f &= ? \end{aligned}$$

2. Classification Combiners

$$\begin{aligned} \alpha_{sc} &= ? \\ C_{sc} &= 2(1) \\ C_I &= 1024 \times 1024 \\ C_F &= ? \\ t_f &= ? \end{aligned}$$

5.5 Structured Frame Store System

In [**Jayasinghe et al. 1988**] a system has been presented that can display the contents of the LDF without a frame buffer directly on the output device. The system is based on methods and algorithms developed in [**Akman et al. 1988**] and [**Kuijk et al. 1988**].

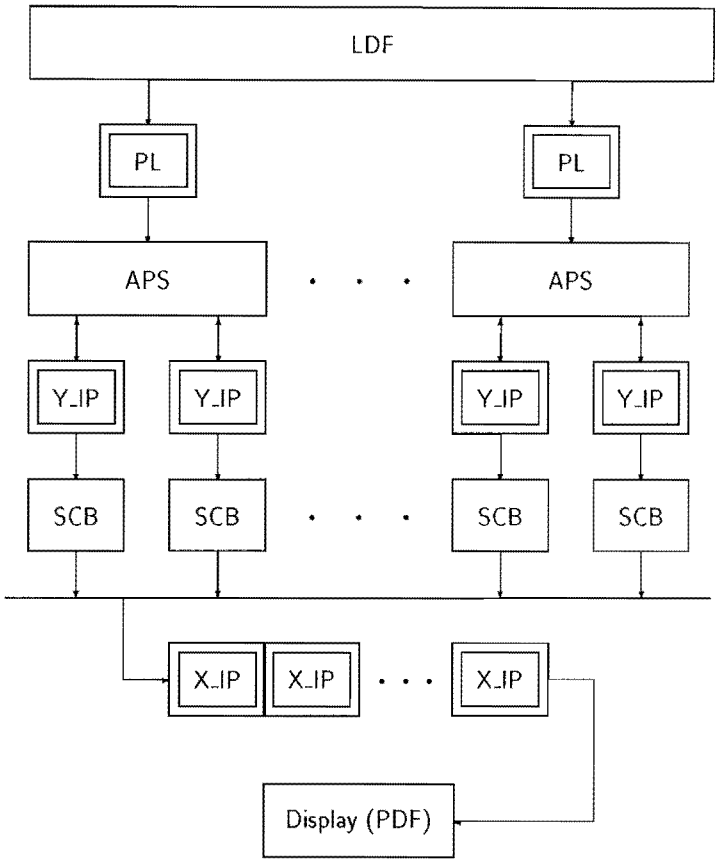

Figure 6: **Display processor of the Structured Frame Store System.** LDF – Linear Display File, APS – Active Pattern Store, SCB – Scanline Command Buffer, X_IP – X Incremental Processor, Y_IP Y – Incremental Processor.

The display controller of that system is constructed from three main building blocks (cf. figure 6 [**Jayasinghe et al. 1988**]). The *Pattern Loader (PL)* receives patterns from the LDF and, if necessary, decomposes them into simpler patterns that can be processed by the rest of the display processor.

These patterns are handed over to *Y Incremental Processors (Y_IP)*. They generate for each pattern a number of scanline commands that are dispatched to pixel processors, called *X Incremental Processors (X_IP)*.

There is one X_IP for every pixel in a scanline. The X_IPs are connected in a pipeline, forming a one-dimensional systolic array. The first X_IP receives a scanline command, executes this command and propagates it to the next X_IP. The task of each X_IP is the incremental computation of pixel intensities.

5.5.1 Functional Classification

Primitive Type(s). The display processor itself accepts patterns [8] as primitives. These general patterns are decomposed into spans because spans are the primitives on which the X_IPs are operating.

Scan Conversion. Contour tracking. The starting and ending pixels of span are incrementally computed by the Y_IPs. The filling is done by the X_IPs.

Hidden Surface Removal. None. The LDF is already preprocessed such that there are only non-overlapping patterns.

Shading. Due to the (restricted) programmability of Y_IPs and X_IPs Wash, flat (constant), Gouraud and Phong shading can be performed. Furthermore, the display processor offers the possibility to generate periodic textures.

Anti-aliasing is also a programmable feature of of the X_IPs. Although not explained in detail, the anti-aliasing technique used seems to be a one-pass technique.

5.5.2 Partitioning

Neglecting the PLs, the display processor performs a two-level partitioning. The first level is a partitioning in the functional domain. The Y_IPs break down the pattern into scanline segments and generate the necessary scanline commands for the X_IPs. Then the X_IPs fill the span.

The second level of partitioning takes place in the time domain for the Y_IPs and in image space for the X_IPs. There are several Y_IPs in order to achieve satisfying performance. Each X_IP handles one pixel in a scanline. This system can therefore be characterised as

$$F(2,1) \circ (T(24 \ldots 245,1) \| I(1024,1))$$

Furthermore, in [**Jayasinghe et al. 1988**] another partitioning in the time domain is proposed for the X_IPs. If the X_IPs are to slow to perform their task in synchronism with the pixel clock of the output device, each X_IP is replicated in order to enlarge the available time for the individual X_IP.

5.5.3 Architecture

Topology. The topology of the Structured Frame Store's display processor consists of different elements. LDF, PLs, and Y_IPs form a tree structure. The leaves of that tree (the Y_IPs) are working on a common bus. That bus is connected to a systolic array of X_IPs.

Control. The control is distributed across the display processor. The different blocks are communicating via intermediate file structures.

Data types. Input to the PL are patterns on a fairly high level. The exact representation of these patterns is not known to the author. These patterns are transformed into polygons described by their geometry and associated colour information. The polygons are decomposed into single spans by the Y_IPs. The X_IPs translate the spans into pixel information, e.g. RGB values.

[8] In [**Kuijk et al. 1988**] patterns are defined as flat regions in continuous 3D. They may be disconnected, concave, with holes or with islands.

Parallelism. There are different kinds of parallelism in this display processor. The PLs are working in parallel on different primitives that may have different types. They are therefore forming a MIMD processor. The Y_IPs are all executing the same command on different primitives which characterises them as a SIMD architecture. This is also the case for the X_IPs that are all executing the same scanline commands.

Regularity. It had not been fixed in the literature how the PLs, Y_IPs and X_IPs will be implemented. It is therefore not possible to determine the regularity of the system.

5.5.4 Performance

We will look only at the X_IP and the Y_IP because the PL is performing functions that we originally associated with the geometry processor. Since these two blocks are differing they are considered separately.

We will assume a screen of 1024×1024 pixels. The primitives should have an average height of 70 pixels. (Some of the values of the following parameters stem from [**Jayasinghe 1989**].)

Characteristic parameters. 1. Y_IP

$$\begin{aligned} t_f &= 20\ ms \\ \alpha_{sc} &= 0.07 \\ C_{sc} &= 244,000 s^{-1} \cdot 20\ ms = 4880 \\ C_F &= 60 \\ C_I &= 1024 \end{aligned}$$

This results in $t_{step} = 0.95\ ns$ for one (!) Y_IP. It is proposed that for a screen with 1024×1024 pixels and Gouraud shading $\approx$ 122 Y_IPs are used which reduces the functional complexity to $C_F = 60/122 \approx 0.5$. For the new step time we obtain $t_{step} = 0.116\ \mu s$ ($\equiv$ 8.6 MHz).

2. X_IP

$$\begin{aligned} t_f &= 20\ ms/1024 = 20\ \mu s \\ \alpha_{sc} &= 1 \\ C_{sc} &= 4880/70 = 70 \\ C_F &= 4 \\ C_I &= 1 \end{aligned}$$

The step time for the X_IP is $t_{step} = 71.4\ ns$. This is equivalent to a clocking frequency of 14 MHz.

Primitives per second. In [**Jayasinghe et al. 1988**] it is claimed that 244,000 Gouraud shaded quadrilaterals of height 70 can be rendered per second.

Delay time. The delay time of this display processor is one frame time (20 ms).

6 Possible Novel Display Processor Architectures

The investigation and classification of several display processors revealed that there are some "white spots" on the map of possible architectures. Two of them that deserve a closer look are the following: Tree structured display processors that partition in image space and display processors on the basis of 2D-arrays of processing elements.

We want to suggest an architecture that closes the first of these gaps: Quadtrees and octrees subdivide the scene or the screen recursively until certain criteria are met. Subdivision of an image into subimages (cells) stops if

- A cell is empty, i.e. contains no object.
- A cell contains exactly one object.
- A predefined level of subdivision has been reached.

This approach offers the possibility of a rendering process that quickly produces an image with a coarse resolution. Afterwards the image is refined further and further until the final accuracy has been reached. Such a behavior is desirable in an environment where response time dominates the desire for image quality.

An architecture for such a display processor could be a tree structure of processors that reflects the structure of the quadtree or octree. A problem in that approach is that it is impossible to provide a tree structure that contains a processor for every possible cell. Therefore, the tree has to be dynamically configurable. It must also be able to detect and reasonably react to overload conditions.

7 Conclusion

The presented taxonomy is mainly founded on a model for the display processor. The model considers two aspects of the display architecture, function and timing. The classifications for *function*, *partitioning* and *performance* have been derived from them. The last distinguishing feature, the display processor's architecture, could not be based on that model, it has been defined mostly intuitively. However, there is a close relation between the topology of an architecture and the employed partitioning scheme. Improvements of the taxonomy can be achieved by refining the model. Especially, an inclusion of architectural aspects into the model seems promising.

The second part of the paper applied the taxonomy to existing display processors. Several times we were unable to completely classify a display processor. That was due to a lack of information in the published material. The taxonomy can serve as a guideline to identify and select the minimum information that is needed to describe a display processor. This would mean to make best use of a taxonomy.

References

[**Abram et al. 1986**] Gregory D. Abram and Henry Fuchs: *VLSI Architectures for Computer Graphics.* In G. Enderle, editor, *Advances in Computer Graphics I*, pages 6–21, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1986.

[**Akman et al. 1988**] Varol Akman et al.: *A Vector-like Architecture for Raster Graphics.* In A.A.M. Kuijk and Wolfgang Straßer, editors, *Advances in Computer Graphics Hardware II*, pages 137–154, Eurographics, Springer-Verlag, 1988.

[**Blinn 1989a**] James F. Blinn: *Return of the Jaggy.* IEEE Computer Graphics & Applications, 82–89, March 1989.

[**Blinn 1989b**] James F. Blinn: *What We Nee Around Here Is More Aliasing.* IEEE Computer Graphics & Applications, 75–79, January 1989.

[**Carlbom 1980**] Ingrid B. Carlbom: *System Architecture for High-Performance Vector Graphics.* PhD thesis, Dept. of Computer Science, Brown University, Providence, R.I., 1980.

[**Carpenter 1984**] Loren Carpenter: *The A-buffer, an Antialiased Hidden Surface Method.* Computer Graphics, 18(3):103–108, July 1984.

[**Clark 1982**] James H. Clark: *The Geometry Engine: A VLSI Geometry System for Graphics.* Computer Graphics, 16(3):127–133, July 1982.

[**Claussen 1988**] Ute Claussen: *Beleuchtungsmodell und Beleuchtungsalgorithmen in der Graphischen Datenverarbeitung.* Forschungsbericht WSI-GRIS 88-3, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, WSI-GRIS, Auf der Morgenstelle 10/C9, D-7400 Tübingen, W-Germany, 1988.

[**Demetrescu 1985**] Stefan Demetrescu: *High Speed Image Rasterization Using Scan Line Access Memories.* In Henry Fuchs, editor, *Proceedings of the Chapel Hill Conference on VLSI*, pages 221–243, Computer Science Press Inc., 1803 Research Blvd., Rockville, Maryland 20850, 1985.

[**Dew et al. 1985**] Peter M. Dew et al.: *Systolic Array Architectures for High Performance CAD/CAM Workstations.* In Rae A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics*, pages 659–694, NATO ASI, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

[**Fuchs 1988**] Henry Fuchs: *An Introduction to Pixel-planes and other VLSI-intensive Graphics Systems.* In R.A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, chapter 6, pages 675–688, Springer-Verlag, 1988.

[**Fuchs et al. 1982**] Henry Fuchs et al.: *Developing Pixel-Planes. A Smart Memory-Based Raster Graphics System.* In *Proceedings of the Conference on Advanced Research in VLSI, M.I.T.*, pages 137–146, January 1982.

[**Fuchs et al. 1985**] Henry Fuchs et al.: *Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes.* Computer Graphics, 19(3):111–120, July 1985.

[**Fuchs et al. 1988**] Henry Fuchs et al.: *Coarse-Grain and Fine-Grain Parallelism in the Next Generation Pixel-planes Graphics System.* In *Proceedings of the International Conference and Exhibition on Parallel Processing for Computer Vision and Display*, University of Leeds, UK, January 1988.

[**Ghazanfarpour et al. 1987**] Djamchid Ghazanfarpour and Bernard Peroche: *A Fast Antialiasing Method with a Z-Buffer.* In G. Maréchel, editor, *Eurographics '87*, pages 503–512, Eurographics, Elsevier Science Publishers B.V. (North Holland), 1987.

[**Giloi 1983**] W.K. Giloi: *Towards a Taxonomy of Computer Architecture Based on the Machine Data Type View.* In *The 10th Annual Symposium on Computer Architecture*, pages 6–15, ACM, 1983.

[**Hu et al. 1985**] Mei-Cheng Hu and James D. Foley: *Parallel Processing Approaches to Hidden-Surface Removal in Image Space.* Computer & Graphics, 9(3):303–317, 1985.

[**Jayasinghe 1989**] J.A.K.S. Jayasinghe: *Personal Communications.* July 1989. —.

[**Jayasinghe et al. 1988**] J.A.K.S. Jayasinghe et al.: *A Display Controller for a Structured Frame Store System.* In A.A.M. Kuijk, editor, *Third Eurographics Workshop on Graphics Hardware*, Eurographics, 1988. The proceedings will be published in the book "Advances in Graphics Hardware III" by Springer-Verlag in 1989.

[**Kedem et al. 1984a**] Gershon Kedem and John L. Ellis: *Computer Structures for Curve-Solid Classification in Geometric Modeling.* Technical Report TR84-137, Rochester Institute of Technology, Microelectronics Center of North Carolina, September 1984.

[**Kedem et al. 1984b**] Gershon Kedem and John L. Ellis: *The Raycastig Machine.* In *The Proceedings of the 1984 International Conference on Computer Design*, pages 533–538, IEEE, 1984.

[**Kilgour 1981**] A.C. Kilgour: *A Hierarchical Model of a Graphics System.* Computer Graphics, 15(1):35–47, April 1981.

[**Kilgour 1985**] Alistair C. Kilgour: *Parallel Architectures for High Performance Graphics Systems.* In Rae A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics*, pages 695–703, NATO ASI, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.

[**Kuijk et al. 1988**] A.A.M. Kuijk et al.: *An Exact Incremental Hidden Surface Removal Algorithm.* In Fons Kuijk and Wolfgang Straßer, editors, *Advances in Computer Graphics Hardware II*, pages 21–38, Eurographics, Springer-Verlag, 1988.

[**Lakshminarasimhan et al. 1989**] A.L. Lakshminarasimhan and Mandayam Srivas: *A Framework for Functional Specification and Transformation of Hidden Surface Elimination Algorithms.* Computer Graphics Forum, 8(2):75–98, June 1989.

[**Latham 1985**] Roy W. Latham: *Image Generator Architectures and Features.* In Roy Latham, editor, *Course notes for high performance image generation systems*, chapter 7, —, Link Flight Simulator Division - The Singer Company – Sunnyvale, CA, July 1985. Reprinted from the 5th Interservice/Industry Training Equipment Conference, 1983.

[**Parke 1980**] Frederic I. Parke: *Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems.* Computer Graphics, 13(3):48–56, July 1980.

[**Pineda 1988**] Juan Pineda: *A Parallel Algorithm for Polygon Rasterization.* Computer Graphics, 22(4):17–20, August 1988.

[**Porter et al. 1984**] Thomas Porter and Tom Duff: *Compositing Digital Images.* Computer Graphics, 18(3):253–259, July 1984.

[**Poulton et al. 1985a**] John Poulton et al.: *Pixel-Planes : Building a VLSI-Based Graphic System.* In Henry Fuchs, editor, *Chapel Hill Conference on Very Large Scale Integration*, pages 35–60, Computer Science Press, Inc., 1803 Research Blvd., Rockville, Maryland, 1985.

[**Poulton et al. 1985b**] John Poulton et al.: *Pixel-planes graphic engine.* In Neil H. E. Weste and Kamran Eshraghian, editors, *Principles of CMOS VLSI Design – A Systems Perspective*, pages 448–480, Addison-Wesley, Reading, Mass., 1985.

[**Pulleyblank et al. 1987**] Ron Pulleyblank and John Kapenga: *The Feasibility of a VLSI Chip for Ray Tracing Bicubic Patches.* IEEE Computer Graphics & Applications, 7(3):33–44, March 1987.

[**Reghbati et al. 1988**] Hassan K. Reghbati and Anson Y.C. Lee: *Computer Graphics Hardware: Image Generation and Display.* IEEE Computer Society, 1988.

[**Robertson 1988**] Philip K. Robertson: *Visualizing Color Gamuts: A User Interface for the Effective Use of Perceptual Color Spaces in Data Displays.* IEEE Computer Graphics & Applications, 50–64, September 1988.

[**Romanova 1988**] Claudia Romanova: *Effizientes Anti-Aliasing für die Bilderzeugung auf Rastersichtgeräten.* In W. Barth, editor, *Visualisierungstechniken und Algorithmen*, pages 109–118, Gesellschaft für Informatik, Österreichische Computer Gesellschaft, Springer Verlag, September 1988.

[**Schneider 1988a**] Bengt-Olaf Schneider: *A Processor for an Object-Oriented Rendering System.* Computer Graphics Forum, 7:301–310, 1988.

[**Schneider 1988b**] Bengt-Olaf Schneider: *Ray Tracing Rational B-Spline Patches in VLSI.* In A.A.M. Kuijk and Wolfgang Straßer, editors, *Advances in Graphics Hardware II*, Eurographics, Springer-Verlag, 1988.

[**Schneider et al. 1988**] Bengt-Olaf Schneider and Ute Claussen: *PROOF: An Architecture for Rendering in Object Space.* In *Third Eurographics Workshop on Graphics Hardware*, Eurographics, 1988. The proceedings will be published in the book "Advances in Graphics Hardware III" by Springer-Verlag in 1989.

[**Sutherland et al. 1974**] Ivan E. Sutherland et al.: *A Characterization of Ten Hidden-Surface Algorithms.* ACM Computing Surveys, 6(1):1–55, March 1974.