A Real-Time Raster Scan Display for 3-D Graphics

D. Jackèl
TU Berlin, Institut für Technische Informatik
Franklinstrasse 28–29, D-1000 Berlin 10

and

H. Günther, B. Herwig, H. Rüsseler
GMD-FIRST an der TU Berlin
Hardenbergplatz 2, D-1000 Berlin 12, FRG

Abstract

This paper describes the architecture of a raster scan display for real-time visualisation of *shaded* polygons. A performance of $15 \cdot 10^6$ Phong shaded pixels per second is a primary goal of a pipelined rendering processor. The performance of the geometry processor, which is responsible for the geometrical transformations, the 3-d clipping and the perspective projection, will exceed 100,000 triangle shaped polygons.

Following a survey of the entire 3-d real-time system, we will describe architectural details of the rendering processor. Finally, the main features enabled by the architecture are highlighted.

1. Introduction

The hardware of conventional raster scan displays supports only the generation of 2-d graphical primitives (*e.g.* points, vectors, circles, etc.) and their mapping from object-space to screen-space. The 3-d visualisation processes, which are extremly time consuming, are executed by the general purpose processor(s) of a host-system.

To fulfill the requirements for 3-d real-time visualisation, high performance graphic architectures have changed significantly. The characteristic of such architectures is the hardware supported viewing pipeline, for which a variety of architectural solutions have been introduced in the past decade.

In order to classify these architectures, we have to divide the visualisation process sequence into its geometry processes and into its *rendering* processes. All vertex oriented processes (*e.g.* geometrical transformations, 3-d clipping and perspective projection of a vertex) belong to the class of geometry processes. All pixel oriented processes (e.g. determination of coordinates and rgb-values of a pixel) belong to the class of rendering processes. These classifications help us to differentiate four basic architectural structures:

delivered by
EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

- Both the geometry processes and the rendering processes are supported by pipeline architectures. The architectural concept of the Silicon Graphics IRIS-Systems [1] as well as the Hewlett Packard SRX graphics engines [2] are representative for this type of graphics system.

- The geometry processes are supported by a pipeline architecture and the rendering processes are supported by a parallel architecture. An example of this are the Pixel Machines of AT & T described by Potmesil and Hoffert [3].

- The geometry processes are supported by a parallel architecture and the rendering processes are supported by pipeline structured architecture. The TITAN of ARDENT and the STELLAR of STELLAR-Systems described by Diede et al [4] and [5] are representative for this type of architectural structure.

- The support for the geometry processes, and the rendering processes are supplied by a parallel structured architecture, which can be found in the PIXAR-System described by Levinthal and Porter [6].

Our system introduced in the following belongs to the third group. The objectives for the capability and performance of the system are:

More than 100,000 triangular polygons should be processed per second. The geometry processes include: rotation, translation, scaling, backfacing, 3-d clipping and perspective projection.

$15 \cdot 10^6$ *pixels* should be rendered per second. The rendering process includes: calculating pixel coordinates, computation of the pixel normals, Phong-shading, HSL to RGB-conversion and hidden surface removal.

At first, a survey of the graphic system and its basic function will be presented, and some implementation details of the rendering processor will be discussed. This is followed by a description of the display file structure and the synchronisation of the geometry processors. Finally, we will summerize the main features and present the state of implementation, including the planned extension and improvements of our system.

2. System Survey and Implementation

The main subsystems of the real-time display, as shown in Figure 1, are :

a geometry subsystem
a rendering subsystem
a frame-buffer subsystem
a 2d-subsystem.

In the following, we will describe the tasks, the basic principles, and the architecture of these subsystems.

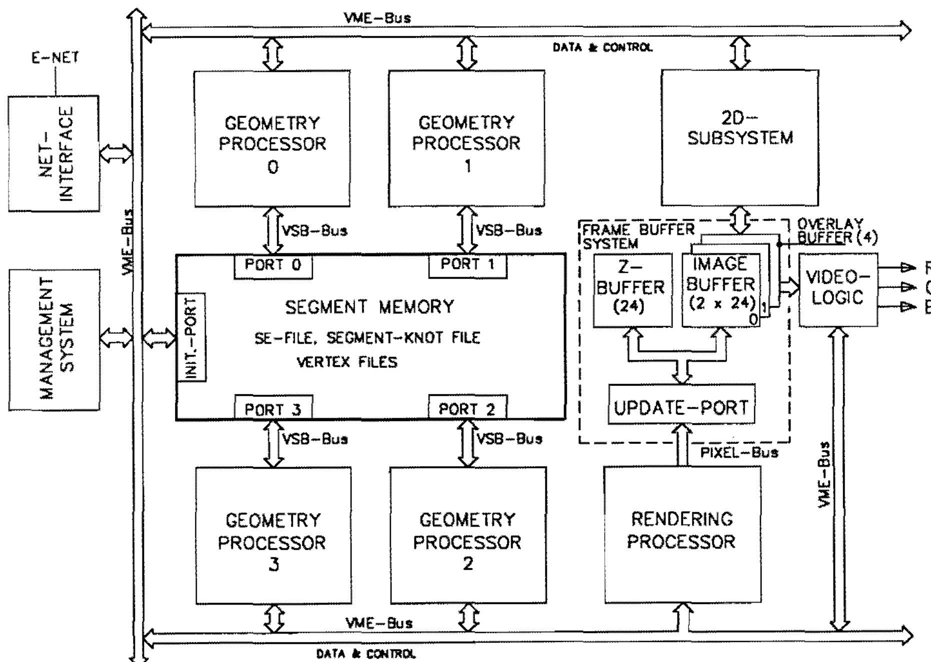

Figure 1: Architecture of the Real-Time Raster Scan Display

2.1 Geometry Subsystem

The main objective of the geometry processing unit is the efficient handling of geometrical rotation, translation and scaling of 3-d objects. Moreover, this subsystem executes the *backfacing, 3-d clipping* and the *perspective-projection* processes. The pipeline of the geometry processes terminates after computing the *parameter set* needed for the initialization

of the *rendering processor*. All geometry processes mentioned above are executed in real time for more than 10,000 triangular polygons every 0.1 second by using four processing units working in parallel.

For the initialization of the rendering processor, two methods are applied to generate the parameter set mentioned above.

Using the first method, the processing pipeline will be started via function calls. Therefore, a set of graphical routines are implemented.

The second method for generating the parameter set implements a display file, referred to in this paper as *object-descriptive data structure* (*ODDS*). Although this method is not as flexible as the first one, it is more efficient. It will be used mainly if a model of a 3-d scene has only fixed vertex coordinates. The *ODDS*-architecture and how this display file is processed by the geometry processor array are discussed later.

2.2 Rendering Subsystem

The execution of the *scan converting* process, which includes *shading* and *z-buffering*, is the main task of the *rendering processor*. The components of the rendering subsystem are shown in Figure 2.

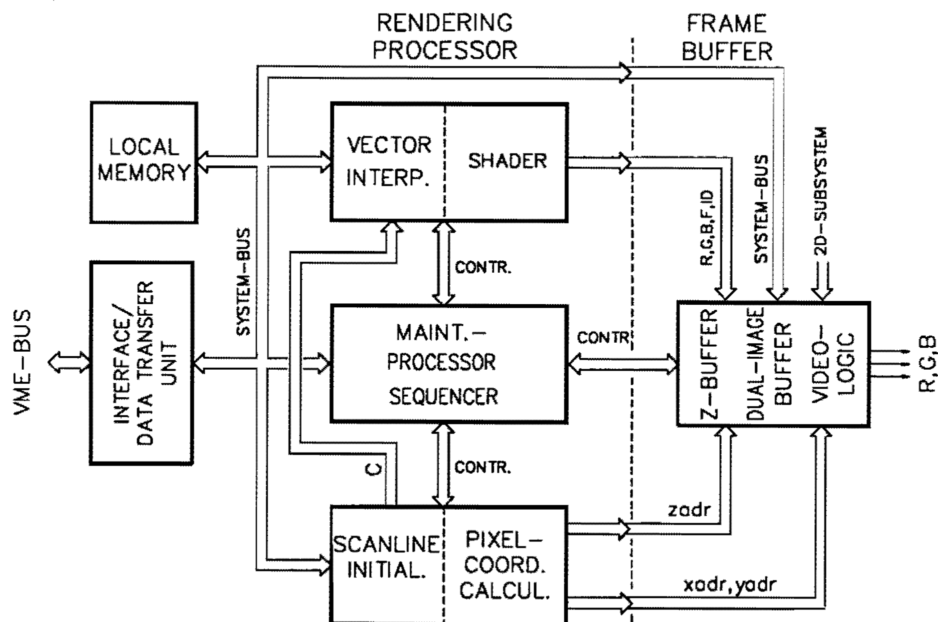

Figure 2: Block Diagram of the Rendering Subsystem

At the core of the rendering subsystem are two pipelines. The task of the first pipeline, consisting of a vector interpolator and a shading unit, is to calculate the rgb-values of the pixels using the Phong illumination model. The second pipeline, with the functional units *scanline initialiser* and *pixel coordinate calculator*, is responsible for the determination of the z-coordinate values as a function of the xy-coordinates.

In addition to the pipeline units, a local memory, a maintenance processor, a VME-bus interface, and a data-transfer unit make up the rendering subsystem. The tasks of these units are discussed in short at the end of section 2.2.

2.2.1 Rendering Pipelines

parameter set. As mentioned in section 2.1, the geometry subsystem generates parameter sets for the initialization of the rendering pipelines. Each parameter set describes a triangle-shaped polygon, which decomposes into a unified form as shown in Figure 3.

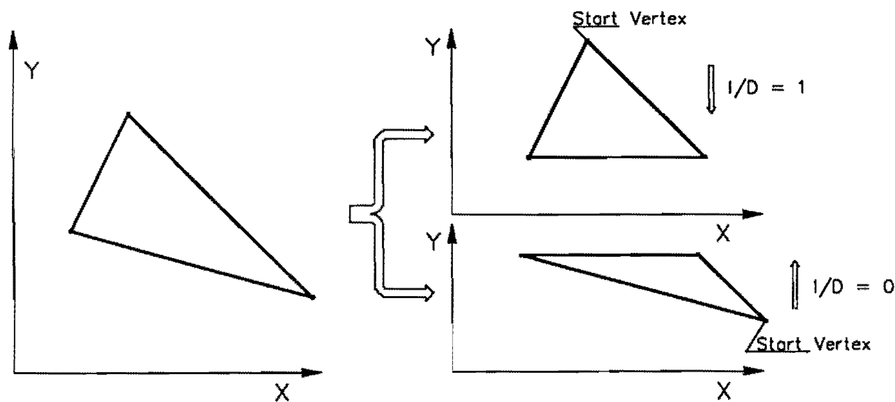

Figure 3: Decomposition of the Triangles into Unified Forms

The decomposition process is necessary to achieve a simple parametrical description of the polygons. By this measure, we obtain a more efficient hardware solution for the rendering pipelines.

Figure 4 shows the organisation of the parameter sets for *smooth-shaded* and *constant-shaded* polygons.

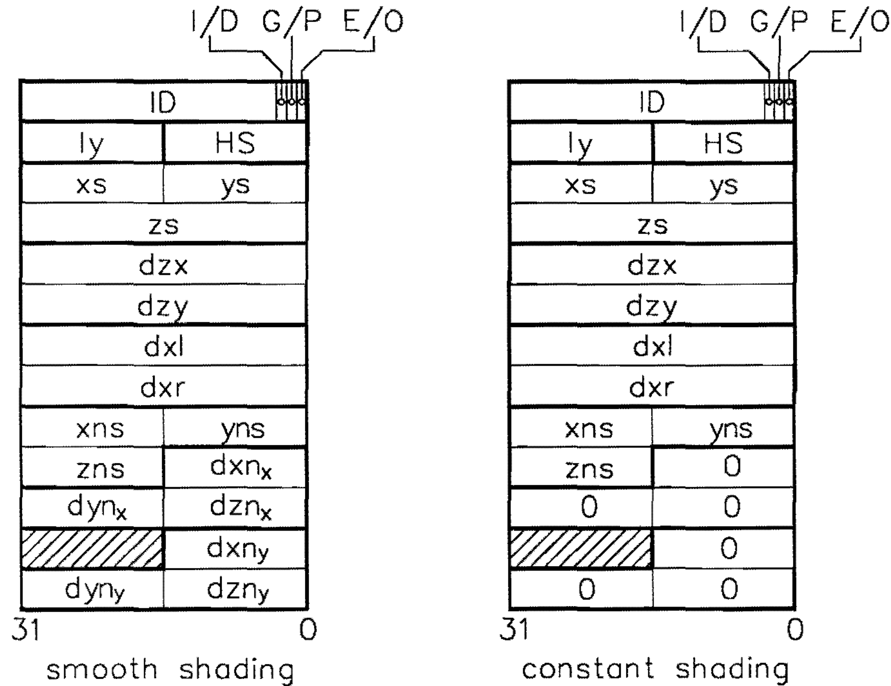

Figure 4: Parameter Sets for Initialising the Rendering Pipelines

The functions of the parameters are discussed in the following:

ID: The ID is a concatenation of the segment identifier SID and the surface element identifier SEID and is used for man-machine-interactions.

I/D: The I/D-bit indicates a decrementation (I/D = 1) or an incrementation (I/D = 0) of the y-coordinate by the scanline generation process.

G/P: The G/P-bit initializes the shading pipeline for executing either the Gouraud or the Phong shading process. For G/P = 0 the shader is switched into the Gouraud shading mode. Otherwise (G/P = 1), the Phong shading mode will be selected.

E/O: The E/O-bit controls the dual image buffer. E/O = 1 selects the odd and E/O = 0 selects the even image buffer unit.

ly: *ly* is the y-address of the last scanline which is rendered by the pipeline units (see Fig. 5a).

xs,ys,zs: *[xs ys zs]* is the vertex position from which the rendering process starts (see Fig. 5a).

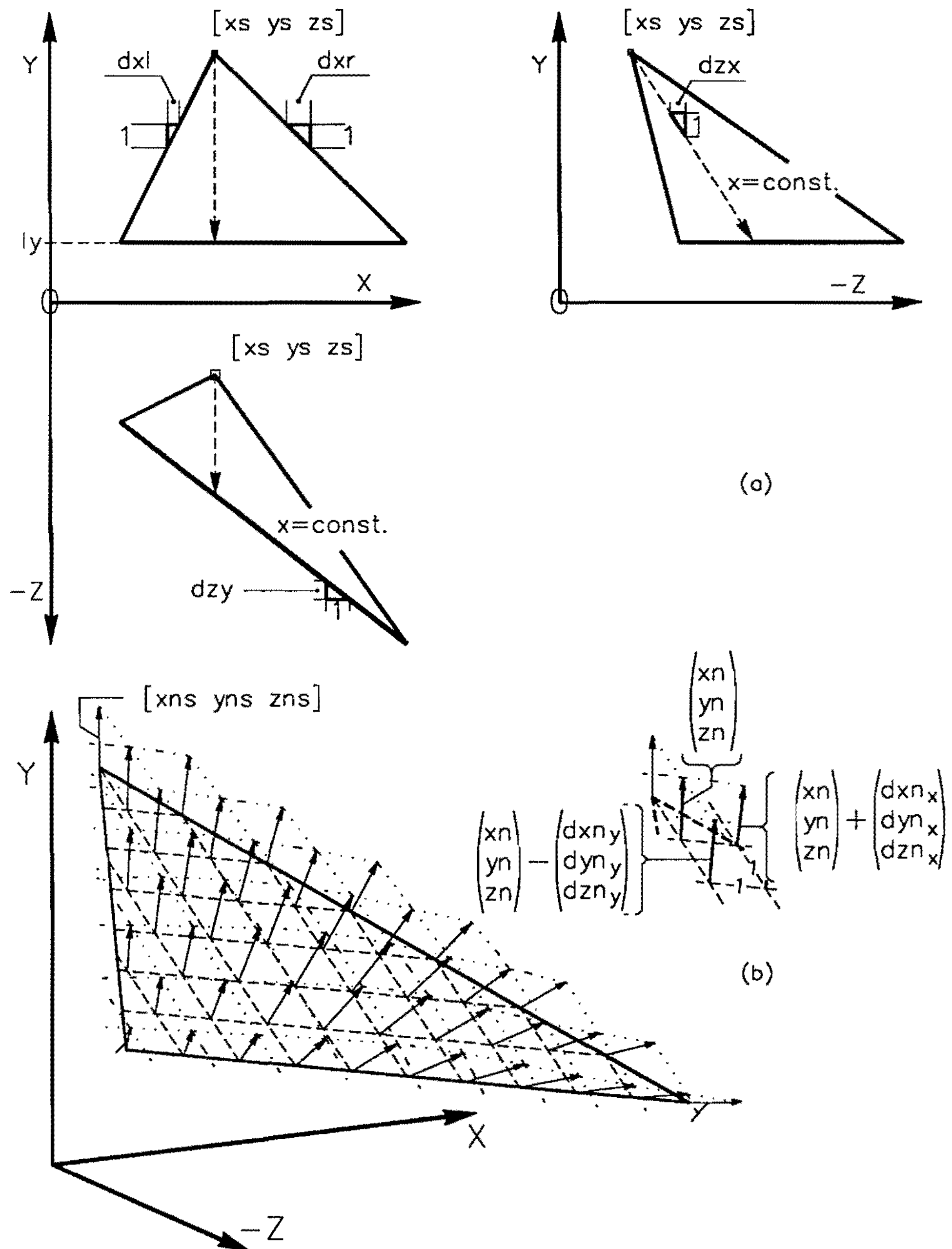

Figure 5a,b: Function of the Parameter: (a) ly, xs, ys, zs, dzx, dzy, dxl, dxr (b) xns, yns, zns, dxn_x, dyn_x, dzn_x, dxn_y, dyn_y, dzny

dzx,dzy: dzx and *dzy* represent the z-slope increment of the SE-plane. *dzx* is a function of an x-incremental step; *dzy* is a function of an y-incremental step (see Figure 5a).

dxl,dxr: dxl and *dxr* represent slope increments of the left and right SE-edges respectively. Both are functions of an y-incremental step (see Figure 5a).

xns,yns,zns: [xns,yns,zns] is the vertex vector from which the rendering process starts (see Figure 5b).

dxn_x, dyn_x, dzn_x: The vector $[dxn_x \; dyn_x \; dzn_x]$ is added to the pixel normal vector [xn yn zn] after executing an x-incremental step. For constant shading all elements of this vector are set to zero (see Figure 5b).

dxn_y, dyn_y, dzn_y: The vector $[dxn_y \; dyn_y \; dzn_y]$ is added to the pixel normal vector [xn yn zn] after executing an y-incremental step. For constant shading all elements of this vector are set to zero (see Figure 5b).

rendering algorithm. Before going into the details of the rendering hardware, the algorithmic structure of the process, which is supported by the pipelines, will be outlined.

At the beginning of the process, the initializing register of the rendering pipelines are loaded with the parameter set. Because the first scanline consists of only one pixel, no pixel interpolation is necessary. The steps b), c) and d) of the inner loop are executed directly.

From the second to n-th scanline the rendering hardware functions as follows:

First, all values needed for initializing the rendering process of one scanline, are computed. The outer loop of Algorithm 1 is responsible for the execution of this initializing process. Within this iteration the x-coordinate components x_l and x_r on both scanline vertices are determined. Additionally, the z-value *zl* and the components pixel normal vector *[xnl ynl znl]* must be calculated only on the left scanline vertex.

xl and *xr* are given by:

$$xl := xl' + dxl \; ; \; xr := xr' + dxr \; . \qquad (1)$$

xl' and *xr'* are the coordinate components of the left and right vertices of the preceding scanline, respectively.

The z-value zl and the components of the pixel normal vector are given by:

$$zl := zm + (xl - xs) \cdot dzx \qquad (2)$$

$$xnl := xnm + (xl - xs) \cdot dxnx \qquad (3a)$$

$$ynl := ynm + (xl - xs) \cdot dynx \qquad (3b)$$

$$znl := znm + (xl - xs) \cdot dznx \qquad (3c)$$

zm, xnm, ynm and *znm* which are initalized by

zm := zs; xnm := xns; ynm := yns; znm := zns

are incrementally decreased or increased by each y-step:

zm := zm + dzy; xnm := xnm + dxny
ynm := ynm + dyny; znm := znm + dzny

Initializing the rendering processor by means of the precalculated parameter set:

if I/D = 0 **then** yinc := -1 **else** yinc := 1

Render the pixel at start vertex [xs ys zs] by the execution of steps b), c) and d) within the inner loop:

for y := ys + yinc **to** ly **step** yinc

Computation of the x-coordinate values x_l and x_r at the left and right vertex of the scanline:
Computation of the pixel normal [xnl ynl znl] at the left vertex: of the scanline

for x = x_l **to** x_r **step** 1

a) computation of the z-value
b) computation of the pixel vector [xn yn zn]
c) computation of the pixel lightness 'l'
d) Z-buffer operation and data transfer of rgb-values to the image buffer

next x
next y

Algorithm 1: Structure of the Hardware Supported Rendering Process

a and b) The scanline rendering is executed within the inner loop. For each x, within the interval *[xl xr]*, the z-component and the pixel normal vector *[xn yn zn]* is calculated (y =const.).

After allocating the initializing values of the scanline

z := zl, xn := xnl, yn := ynl and zn := znl

all further z-values and normal vector components are given by:

$$z := z' + dzx \qquad (4)$$

$$xn := xn' + dxnx \qquad (5a)$$

$$yn := yn' + dynx \qquad (5b)$$

$$zn := zn' + dznx \qquad (5c)$$

z' and the vector *[xn' yn' zn']* are preceding values.

c) This is followed by the shading process. In order to execute this process, three main steps are necessary.

First, the result of dividing the *[xn yn zn]*-vector by the value *zn* is the unified form *[xn/zn yn/zn 1]*.

Second, the xn/zn- and yn/zn-components may be regarded as address pointers of a table memory. This memory contains the precalculated values of a light reflectance map (see [7]). Under these conditions, it is allowed to consider the light reflectance value *l* of a pixel only as a function of both parts of the address pointer $l = f(xn/zn, yn/zn)$.

In the third step of the shading process, the value *l* is concatenated with the hue *h* and colour saturation values *s* allocated to the surface element. For each of the 2^{17} *hs/l*-combinations the corresponding rgb-values can be allocated by a look-up table.

d) The final step of the rendering process is the hidden surface removal via z-buffering. Because this method is easily and efficiently supported by simple hardware, z-buffering is commonly used in contemporary real-time raster scan displays.

2.2.2 Hardware Organisation of the Rendering Pipelines

When initialising the rendering processor, all different *SE-parameters* are distributed to their own *FIFO*-memories. The *FIFO*'s are useful in achieving a sufficient load balance between the rendering and the geometry processor. Moreover, the access processes of both subsystems do not have to be synchronized.

As the *SE-parameters* are distributed to private *FIFO's*, the rendering pipelines can be initialized by one load cycle. In the following the hardware organisation of both rendering piplines will be discussed.

vector interpolation unit. The task of the *vector interpolation unit* is to determine the barycentrical-oriented pixel normal *[xnl ynl znl]* at the left vertex of the scanline with respect to the equations 3a-c. This process will be executed with four pipeline stages mainly consisting of six adders and three multipliers.

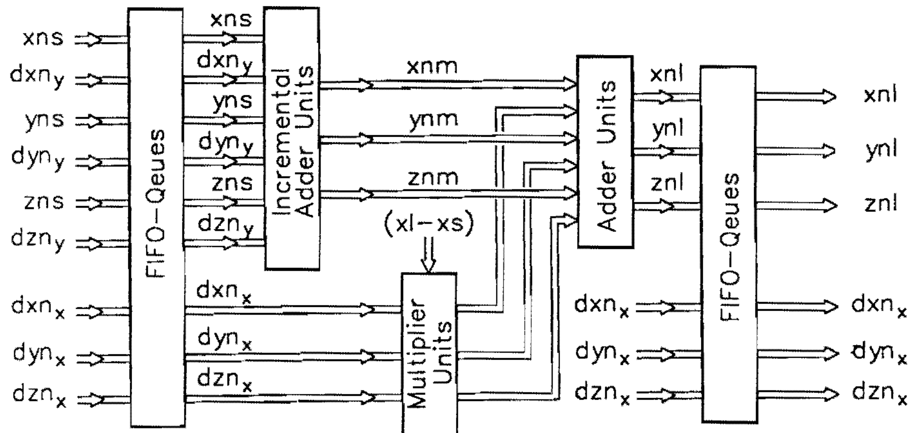

Figure 6: Blockdiagram of the Vector Interpolator

All values of the pixel normals are written into a second FIFO-bank. This functional unit operates as an interface serving as an asynchronous link to the shading unit and the vector interpolator, and ensures a sufficient load balance.

shader. The determination of the lightness values of the pixels within a scanline, is the task of the shader.

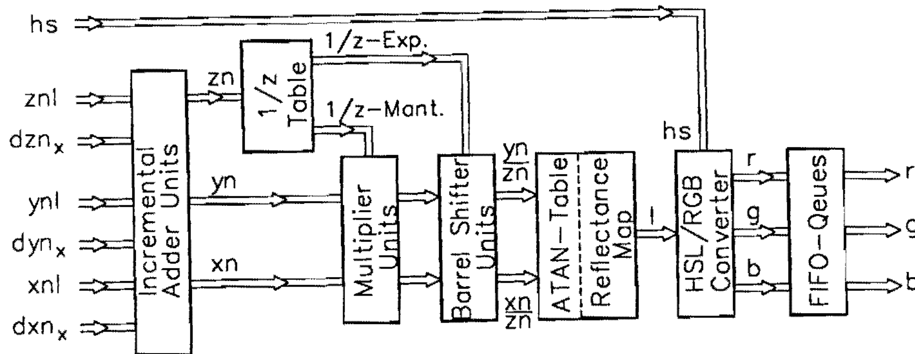

Figure 7: Block-Diagram of the Shader

In the first stage, the shading unit manages the interpolation of the pixel normals, corresponding to the equations 5a-c. The interpolation process is applied by using three incremental adders.

The next three stages are necessary to transform the pixel normal vectors *[xn yn zn]* into the form *[xn/zn yn/zn 1]*. To avoid the time consuming division by zn, a *1/zn*-table is used. The *zn-to-1/zn* table conversion is followed by a multplication stage. Both products xn•(1/zn) and yn•(1/zn) must be normalized. This is done in the fourth stage by barrel-shifting units.

In the sixth and seventh stage the determination of the *l-values* is easily accomplished by using a precalculated *reflection map.* Mainly under the restriction of *fixed-positioned virtual light sources*, the determination of a *l-value* only requires one access operation to the *reflection map* formed by a look-ahead-table memory. For precalculation of the 32,400 reflection-map values the *Phong Light-Model Equation* is applied.

The task of the last shader stage is the execution of the HSL/RGB-conversion. This is also accomplished by using three *look-up tables* (HSL/R, HSL/G and HSL/B) which have each a size of 128K x 8 bit.

scanline initializing. Scan-conversion is the task of the second pipeline, which is divided into a *scanline initializing unit* and a *pixel coordinate calculator.*

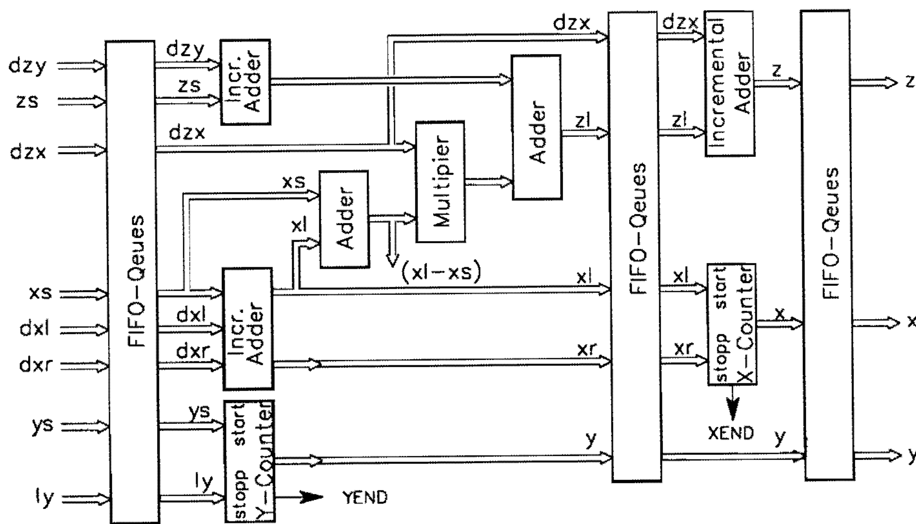

Figure 8: Block-Diagram of the Scanline Initializing and the Pixel Coordinate Calculator

In the first stage of the *scanline initializing* process two incremental adders are used to determine the *xl* and *xr* coordinate components according to the equations 1a and 1b. A third adder-unit is required to increment or to decrement the constant *dzy*-value to the old z-coordinate. The second stage is responsible for the determination of the difference *(xl-xs)*. The task of the third and fourth stages of the *scanline initializing* process is the determination of the *zl*-value according to the equation 2.

Moreover, the *scanline initializing unit* contains a binary y-counter. A comparator unit is used for the determination of the y-coordinate value and for indicating the last scanline by generating the *YEND* signal.

Analogous to the vector interpolator unit, all output values of the *scanline initializing unit* are written into the second *FIFO*-memory bank.

pixel coordinate calculator. The *pixel coordinate calculator* computes all *z-values* within a scanline (equ. 4), which are used for ***hidden pixel removal*** by means of *z-buffering*. The x-coordinate component is determined by a binary counter starting at the x_l*-value* and stopping at the x_r*-value*.

2.3 Frame Buffer Subsystem

The frame buffer supports a screen resolution of 1024 x 1280 pixels. As shown in Figure 6, the entire system consist of a 2 x 24-bit *dual image buffer*, a 24-bit *z-buffer* with an update port, a 4-bit *overlay-buffer*, a bank address computation unit and a buffered *I/O-multiplexer*. The *serial converter & multiplexer*, which works in parallel to the *I/O-multiplexer*, can be considered to be part of the dual image buffer. In the following we will describe these functional units in more detail.

z-Buffer. In order to achieve update cycles of 75 ns, the *z-buffer* implements the *complex memory interleaving* method. This ensures the rendering rate of $15 \cdot 10^6$ pixel per second, as required.

The z-buffer is partitioned into 20 independent-addressable memory banks. The mapping of the logical frame-buffer address to the physical bank-address is the task of the *bank address computation unit*. Moreover, this unit is responsible for bank addressing fault detection. Details of the *complex memory interleaving* method applied to frame-buffer systems can be found in Rüsseler et al [8].

The presetting of the z-buffer is necessary after each frame-cycle. Using *video-RAM's*, the preset time needs 240 ns for a group of four pixel rows, *i.e.* 0.7 ms for the entire z-buffer.

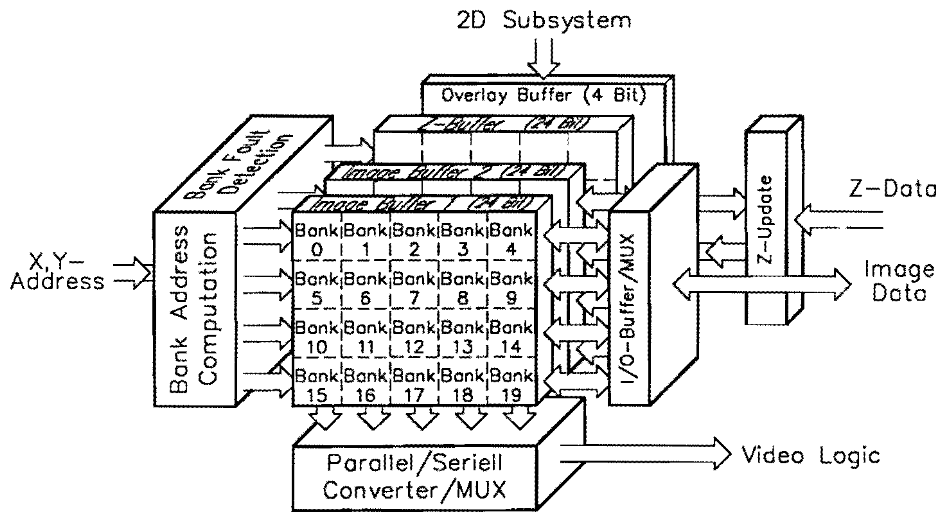

Figure 9: Frame Buffer System

dual image buffer. The dual image buffer, divided into an *even* and an *odd* unit, receives the *rgb*-values determined by the *shading unit.* For real-colour display, each r, g and b-component has a size of 8 bits. Analogous to the *z-buffer,* this functional unit is partitioned into an equal number of memory banks using the same memory interleaving method. The '*I/O-Buffer & Multiplexer*', controlled by *E/O-bit,* is responsible for the parallel-I/O of the *rgb*-values to both image buffer units.

overlay buffer. The task of the *overlay buffer* is the display of alphanumeric information or vectorized graphical primitives. Because the *2-d subsystem* has the access right to this unit, it can work in parallel to the image-buffer and z-buffer units.

2.4 The 2-D Subsystem

The main objective of the *2-d subsystem* is the generation of 2-d primitives, for example, dots, 2-d vectors, circles, arcs, ellipses and alpha-numeric characters. Moreover, it is planned to use this unit for supporting the *man-machine-interaction* process based on the *X Window System.*

For the implementation of the 2-d subsystem an *off-the-shelf* graphics processor is applied.

3. Summary

We have presented an architecture for a 3-d graphics system for real-time polygon rendering. The main features of the system are summarized as follows:

100,000 triangle shaped polygons per second are processed by geometry transforms, backfacing, 3-d clipping and perspective projection.

$15 \cdot 10^6$ pixels per second are Phong-shaded and z-buffered by two rendering pipelines.

The size of the frame buffer is 1280 x 1024 pixels, with a total number of 76 bits per pixel.

Developmental status as of summer 1989: The pre-version of the geometry processor, the segment memory and the 2-d display is being completed. The frame-buffer system is being tested. The rendering processor is in the layout design state.

At the end of the year, we expect an operational prototype.

4. References

[1] K. Akeley, T. Jeremoluk : *High Perfomance Polygon Rendering*; Computer Graphics, Volume 22, No 4; pp. 239-246, 1988.

[2] D. Burgoon: *Pipelined Graphics Engine Speeds 3-D Image Control*; Electronic Design, July; pp. 113-119, 1987.

[3] M. Potmesil, E.M. Hoffert: *The Pixel Machine: A Parallel Image Computer*; Computer Graphics, Volume 23, No 3; pp. 69-78, 1989.

[4] T. Diede et. al.: The Titan Graphics Supercomputer Architecture; Computer, September; pp. 13-29, 1988.

[5] B. Apgar et al.: *A Display System for the Stellar Graphics Supercomputer GS1000;* Computer Graphics, Volume 22, No 4; pp. 255-262, 1988.

[6] A. Levinthal, T. Porter: *Chap - A SIMD Graphics Processor*; Computer Graphics, Volume 18, No 3; pp. 77-82, 1984.

[7] B.K.P. Horn: *Understanding Image Intensities*; Artificial Intelligence Vol. 8, No. 2; pp. 201-231, 1977.

[8] H. Rüsseler, H. Günther, D. Jackèl: *Eine Bildspeicherarchitektur für Raster-Displays mit Echtzeiteigenschaften*; ITG-Fachbericht 102; VDE-Verlag GmbH; 1988.