The Triangle Shading Engine

Hans-Josef Ackermann and Christoph Hornung

ABSTRACT This paper describes an algorithm implementing the Gouraud-shading of triangles and its realization in hardware. Different realizations using span shading hardware are discussed. Their drawbacks lead to the concept of a triangle shader, designed as an ASIC. Interfaced to a signal processor for geometry computations, this chip will provide an effective and low-cost 3D-extension to graphics subsystems in the PC environment.

1 Introduction

Simultaneously with the growing complexity in VLSI, there has been an explosion of computational power in personal computer systems. Personal or Entry Level Workstations feature system performances, which were reached only by minicomputers and mainframes few years ago. Higher performances and system resources brought up the demand for better user interfaces and, as a picture is worth a thousand words, the new user interfaces were menu systems using simple forms of graphics. New graphics hardware meaning better video adapters were developed, and graphics software standards like GKS and PHIGS were brought from workstations to the PC. The demand for more than low level and faster graphics is ever growing through popular graphics applications like CAD/CAM which run on PC platforms. These applications however, require some hardware acceleration to release the CPU from time intensive visualization tasks.

2 Graphics Hardware

Today, all state-of-the-art workstations feature some form of hardware support for graphics. There have been several approaches to achieve major graphics performance in the field of high-end workstations. One of the first to become known was the pixel plane architecture by H. Fuchs which is a massive parallel system [1]. Another approach is the geometry engine now marketed by Silicon Graphics [2] making intensive use of pipelining. Common to all these systems is their complexity, the use of highly integrated custom chips, their restriction to one workstation architecture and their costs.

The rendering pipeline, supported by graphics hardware basically consists of geometric modelling, transformation, clipping, lighting, scan conversion, shading, and in the case of 3D-graphics, z-buffering and hidden surface removal. These tasks can be sub divided into two blocks, geometry calculations and raster calculations. Geometry calculations cover modelling, transformation, clipping and lighting, whereas raster calculations handle scan conversion and hidden surface removal by z-buffering. In order to achieve a balanced

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

system, the units performing the tasks of these two blocks should have a comparable performance. In single processor systems the CPU works on the operating system, the application and the whole geometric and rendering pipeline ne, a burden too heavy even for the fastest general purpose CPUs. In order to improve the performance, accelerators are used. The widest spread form of an accelerator is an arithmetic-coprocessor chip plugged in a prepared socket on the processor board. More powerful and complex accelerators are add-on boards, communicating with the main CPU via its bus-backplane. These add-on accelerators may be classified according to their field of application.

2.1 Floating Point Accelerators

This first group uses RISC or Digital Signal Processor systems for the acceleration of floating point intensive tasks. In the case of graphics these tasks are those of the first group mentioned above: transformation, clipping and lighting. With single state-of-the-art processors like the i860 or the TMS 320C30 floating point performances up to 50 MFLOPs can be reached. According to [3] this performance is sufficient for the transformation and the clipping of 100,000 to 300,000 vectors/s.

2.2 Graphics Processors

The second group is covered by so called graphics cards. Graphics cards typically consist of processors or controllers with some graphics properties, a frame buffer and the video logic necessary for the connection to a monitor. Processors often applied, are the TMS 34010 and 34020 by TI [4] and the DP 8500 by National Semiconductor [5]. These chips mainly support BitBLT operations and simple 2D-line drawing. 3D-functions are not supported. So these graphics cards do not reach the performance for shading an d hidden line removal necessary to achieve a balanced system together with a floating point accelerator.

2.3 Goal for a Hardware Shader

The goal of a hardware shader is to reach an equivalent performance of about 50,000 to 100,000 shaded and z-buffered triangles per second. The shading algorithm to be implemented is the Gouraud Shading [6], which does a linear interpolation of the color between the edges of a triangle. This algorithm is supported by major graphics standards like PHIGS-PLUS. It is well fit for an implementation in hardware. Color and z-values of the vertices have to be computed according to a chosen lighting model.

3 Triangle Shading

In comparison to other area primitives, triangles have some major advantages. Triangles are inherently planar. This leads to a constant increment for color (r, g, b) and depth (z) along the scanlines. A fixed shading algorithm for triangle shading can be formulated. This is especially important for a hardware implementation. Arbitrary areas, delimited by polygons can be decomposed into triangles. This shows that triangles are a well-suited rendering primitive.

3.1 Conception

The shading of triangles consists of three main steps:

- initialization
- edge interpolation
- span interpolation

3.2 Initialization

Initially, a triangle is defined by its three vertices (x, y, z) and the respective color values (r, g, b). This representation has to be converted into another one, better suited for shading. Depending on the requirements to the triangle shading algorithm, different formats may be used. An important requirement has been the implementation of an exact point sampling. Exact point sampling is essential for the support of texture mapping and transparency, planned features of future versions of the triangle shader. Furthermore, during edge interpolation, no pixels are missed or drawn twice. To fulfill this requirement, the following data structure was chosen:

```
typedef struct  short f, i;  tFix;
typedef struct  tFix x, y, z, r, g, b;  tPoint;
typedef struct  tFix dx, dy, dz, dr, dg, db;  tDeltaPoint;
typedef struct
  {
  tPoint        PTop;
  tDeltaPoint   dEdge;
  tDeltaPoint   dSpan;
  tFix          xTop;
  tFix          yMid;
  tFix          yBot;
  tFix          dxMidTop;
  tFix          dxBotMid;
  }
tTriangle;
```

This structure represents a triangle as shown in Figure 1. In our implementation, interpolation always starts from the longest edge of a triangle. Thus, in any case, only one edge-increment *dEdge* of type *tDeltaPoint* has to be calculated. This algorithm requires the ability to interpolate the spans along both, the positive and the negative x-axis.

Figure 1 illustrates the meaning of the components of tTriangle. *dEdge* does not correspond to the exact edge from *PTop* to *PBot*. Instead of the exact edge, the closest edge through *PTop* pointing to the outside of a triangle and having and integer slope dx/dy is chosen. This construction allows the implementation of a Bresenham-like algorithm guaranteeing that only the centers of the pixels are taken into account. All data values are stored using the type *tFix* with an integral and a fractional part, as usual in a DDA-algorithm. Therefore, this concept combines both the accuracy of the Bresenham-algorithm and the speed of the DDA-algorithm.

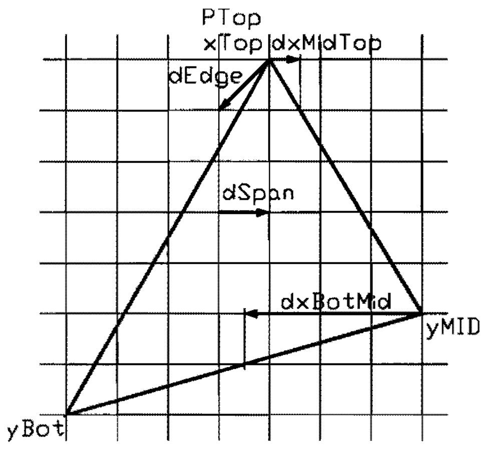

Fig. 1.

3.3 Edge Interpolation

During edge interpolation, a triangle is scanned along y from top to bottom. This algorithm delivers the boundary of a triangle as well as the initial values of r, g, b and z, which are necessary for the span interpolation.

The initial values for r, g, b and z are calculated by incrementing *PTop* along *dEdge* until *yBot* is reached (Figure 1; *dEdge.dy* = 1 in any case). The final x-value is initialized with *xTop* and first incremented by *dxMidTop* until the line *yMid* is reached, and then incremented by *dxBotMid* until reaching *yBot*.

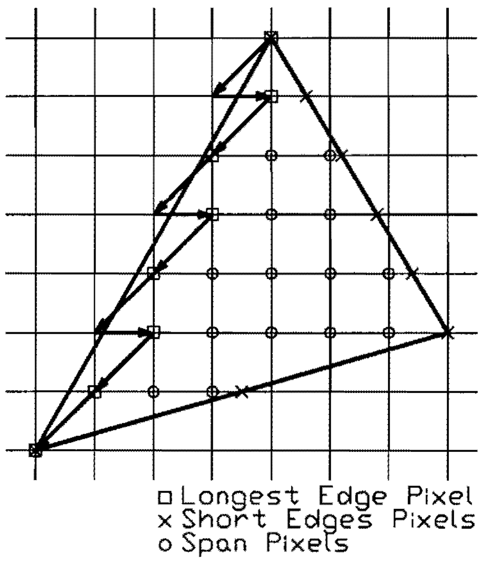

Fig. 2.

3.4 Span Interpolation

The span interpolation forms the inner loop of the triangle shading. Starting from the longest edge, all pixels are set until the corresponding short edge is reached (Figure 2). This iteration is done using the variable dSpan, including the elements x, y, z, r, g and b. The values of these increments are constant throughout a whole triangle. During span interpolation, the value of *dSpan.dy* is constantly 0, while *dSpan.dx* is 1 or -1 depending on the direction of iteration.

4 Realizations of Triangle Shading Using Span Shading Hardware

4.1 The Zebra Chip

Starting point for the realization of Gouraud shading in hardware was the Zebra chip [7], an ASIC designed by National Semiconductor to fit into a system controlled by National's raster graphics processor DP 8500. This chip mainly contains a set of four adders for the interpolation of r, g, b and z and a comparator for hidden pixel removal calculation. The minimum cycle time for the interpolation is 100 ns, meaning that pixels could be written to the frame buffer with a maximum rate of 10 Mpixels/s.

As mentioned above, the triangle shading algorithm consists of the subtasks initialization, edge interpolation and span interpolation. Implementing the triangle shading using the Zebra chip, both the initialization as well as the edge interpolation have to be done in software by the host, while only the span interpolation can be boosted by the Zebra chip. Initialization data for each span have to be transferred to the Zebra chip's set-up registers. This leads to a high data stream and communication overhead between the host and the Zebra chip. The maximum performance according to the interpolation rate without communication is 100,000 triangles/s with 100 pixels each. In order to test the true resulting system performance, three different systems using the Zebra chip were designed and analysed.

4.2 An Initial Approach

The initial system consisted of a Zebra chip with a $2 \times 256 \times 256 \times 24$ bit frame buffer and a $2 \times 256 \times 256 \times 16$ bit z-buffer, both static RAM, controlled by a hardware state machine for address generation and fed by a T800 transputer [8]. Due to the memory access time of 100 ns and the timing of the z-comparison, an interpolation rate of 4 Mpixels/s was reached.

4.3 An Improved System

This system was redesigned using fast page mode DRAMS for a $2 \times 512 \times 512$ frame buffer and a $2 \times 512 \times 512$ z-buffer. The state machine was optimized for the maximum speed which could be achieved using standard page mode DRAMs. The result was an interpolation rate of about 7 Mpixels/s. Like in the first system, initialization and preprocessing is done by a T800 transputer. The z-buffer can be automatically cleared while image data are transferred to the output stage of the system.

4.4 A Different Approach

The third approach was an AT-based system [9] built around the whole Advanced Graphics Chip Set of National Semiconductor [10]. In this system the interpolation was controlled by the linedraw cycle of the raster graphics processor (RGP). According to the framebuffer of $2048 \times 2048 \times 8$, the z-buffer was $2048 \times 2048 \times 16$ and the Zebra chip interpolated color indices. Due to the fairly complex timing an interpolation rate of about 2 Mpixels/s was reached. A FIFO for initialization data was used to decouple three possible data sources. Transfer from the FIFO to the Zebra chip's registers was realized by a hardware state machine.

4.5 Analysis of the Results Using Span Shading Hardware

The Zebra chip can be parallelly initialized while interpolation is going on. The time required for transferring initialization data for one span depends on different aspects:

- the chosen interface bus width (16- or 32-bits)
- the write cycle time of the initializing device
- the number of color planes (when using index colors, g and b values are not used)
- the type of the transferred span (the transfer of the increment values is required only for the first span of a triangle; see Section 3.4.)

It could be pointed out, that the break even point between initialization and interpolation was in the area of 12 to 29 pixels. There are two consequences:

- the number of spans which can be initialized is restricted to about 400,000/s.
- a higher interpolation rate only pays for spans considerably longer than break even spans.

In advanced quality pictures only a small percentage of lines exceeds this length. Assuming a maximum data block of 44 bytes per span and triangles with 100 pixels and 10 spans, the requested 50,000 triangles/s would require a peek dataflow of 22 Mbyte/s for initialization. In addition, at least the edge interpolation must be done by the transferring device.

This led to the conclusion, that using spans as primitive, our goal of 50,000 to 100,000 shaded triangles/s cannot be reached. Doing the edge interpolation in software and realizing a communication for each span drops down the system performance dramatically. Especially for high quality pictures with small triangles the resulting data rates cannot be handled. The primitive one level above is the triangle. Using triangles as primitive could reduce the datasets for triangles with 100 pixels by one order of magnitude. So the logical consequence was to design a triangle shader.

5 The Triangle Shader Chip

5.1 Conception

The triangle shader chip was designed to implement both edge and span interpolation described in Sections 3.3 and 3.4. Up to now, the initialization has to be done in software running on the host CPU or the floating point accelerator.

5.2 System Overview

A system utilizing the triangle shader chip will consist of three major components. The host processor running the operating system controls the whole graphics subsystem via the standard bus system. The graphics subsystem will be built out of a floating point accelerator and a rendering engine carrying frame buffer, z-buffer, video logic, triangle shader chip and a graphics controller for buffer management. The datapath between floating point accelerator and rendering engine will use one of the accelerator CPU's busses for fast transfer of initialization data. During shading operation, frame buffer and z-buffer are controlled by the triangle shader chip.

5.3 Hardware Implementation of the Triangle Shader

The triangle shader chip contains three major functional blocks (Figure 3). A 16-bit-wide data interface provides access to the set-up registers holding initialization data. A 12-bit-wide control interface provides access to the state machine with its associated microcode RAM. Nine memory strobes can be independently programmed within 25 ns periods to fit the necessary frame and z-buffer timing. Furthermore, the control interface provides lines for automatic control of an external FIFO memory containing initialization data. Data are transferred to the set-up registers while the preceding triangle is processed by the interpolators. The interpolator section contains six independent parallelly working interpolation units for x, y, z, r, g and b. The interpolators work on both edges and spans. During span processing, edge data are in internally latched. A mask function protects specified bit planes from interpolation.

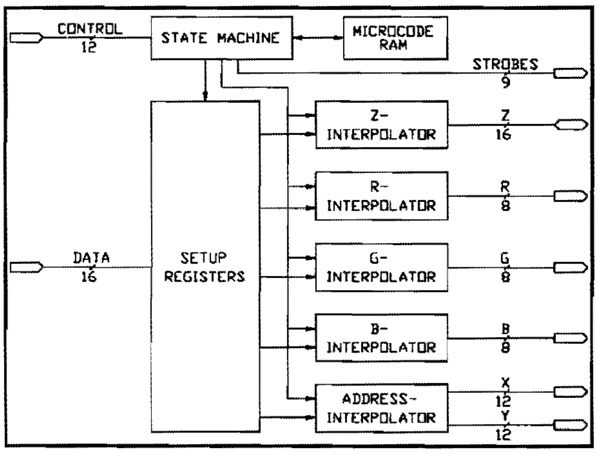

Fig. 3.

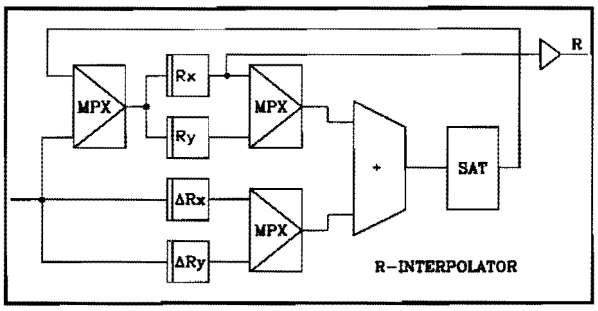

Fig. 4.

Figure 4 shows the datapaths of the r-interpolator consisting of registers, multiplexors, adder, saturation logic and output buffer. All values except y are processed in fixed-point representation. The widths of the fractional parts have been chosen to prevent rounding errors while interpolating the longest line possible in a 2048 by 2048 pixels frame buffer. The actual formats are,

	INT	FRAC
x:	12 bit +	12 bit
y:	12 bit +	0 bit
z:	20 bit +	12 bit
r,g,b:	8 bit +	12 bit

The triangle shader chip has been designed as an ASIC using a 2 mm standard-cell process. The gate equivalent count is 11,358 (kernel).

5.4 Simulated Performance

All simulations have been done assuming a clock frequency of 40 MHz for the internal state machine of the triangle shader chip.

Initialization Data Transfer

An advantage of the triangle shading algorithm is, that unlike to the span shading approach, the number of initialization values is constant for any triangle. According to the chosen data structure *tTriangle* and formats, 37 16-bit words per triangle have to be transferred to the registers of the triangle shading chip. The state machine supports the reading of initialization data from an external FIFO memory. Depending on the used micro program, the read cycle time is 50 ns (linear program) or 100 ns (loop). This leads to a fixed initialization time for a triangle of 1.85 ns (3.7 ns). Therefore the number of triangles which can be initialized is 540,000/s (270,000/s). The initialization works in parallel with either edge or span interpolation.

Edge Interpolation

Due to the restricted chip complexity, edge and span interpolation have to use the same interpolator circuitry. Therefore, a parallel interpolation of edge and span data is not possible. The cycle time for the calculation of the next longest edge pixel (Figure 2) is 350 ns. If a correction step in y direction is necessary (Figure 2) an additional 100 ns are added. Switching from the first short edge to the second short edge requires 200 ns for loading the new parameters.

Span Interpolation

The span interpolation cycle time has been simulated assuming standard fast page mode DRAM timing for z-buffer and frame buffer access. The resulting interpolation cycle time is 150 ns which corresponds to a maximum interpolation rate of 6.7 Mpixels/s.

Analysis

In comparison to the span shading approaches, the data flow of initialization data for 50,000 triangles/s could be reduced from 22 Mbytes/s to 3.7 Mbytes/s. The resulting transfer cycle time can be still easily met for 100,000 triangles/s.

The break-even point between initialization data transfer and iteration depends on the ratio between number of spans and pixels per span. In the case of a triangle degenerating to a horizontal line, the break-even point is 9 pixels per triangle. The break-even point decreases to 4 pixels if a triangle degenerates to a vertical line. These values show, that the problems of initialization data transfer have been solved with our design. The maximum shading performance is limited by the span interpolation rate of 6.7 Mpixels/s. Figure 5 shows a performance diagram assuming triangles with 100 pixels and shapes changing linearly from a horizontal to a vertical line. Figure 6 shows a performance diagram assuming rightangled, isosceles triangles.

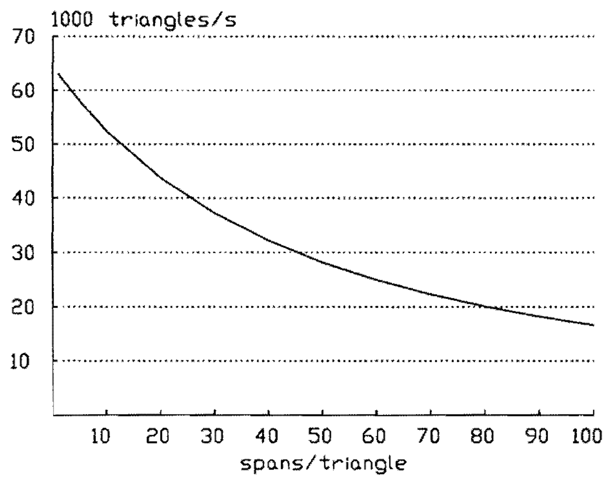

Fig. 5.

5.5 Status of the Triangle Shader

The algorithm of the triangle shader has been implemented first as a C program. The function of the algorithm has been certified to produce correct results using comprehensive and difficult case test patterns. The transfer from the C program to the VLSI design has been finished including logical and timing simulation of the design. The turn-around time for the chip production will be about six months. During this time, an evaluation system will be designed and built. This system is expected to work using the triangle shader chip in Spring 1991.

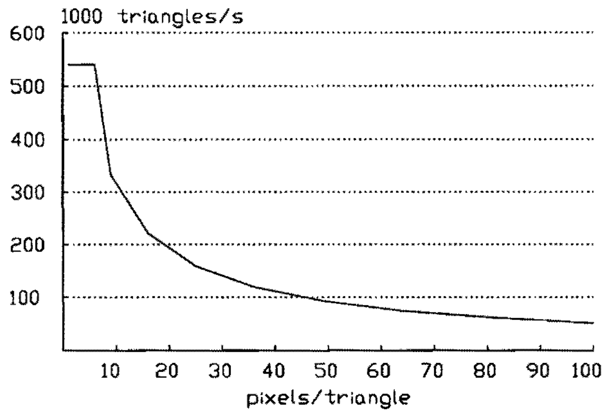

Fig. 6.

6 Future Work

The triangle shader as described above, is only the first step towards the development of an integrated hardware renderer. Two extensions are planned for the future: higher integration and enhanced, programmable functionality. With a higher scale integration process available, the initialization task will also be integrated on chip. The initialization requires an ALU supporting fast multiplication and division as well as a control sequencer.

Functionality enhancements will include alpha blending and texture mapping. Alpha blending can be implemented by adding an alpha channel to r, g and b. Furthermore, a blending stage consisting of adders and multipliers is required. This functionality can be used for features like anti-aliasing and transparency. Texture Mapping is more complex. A separate address generator and interpolation logic is required. Texture mapping will be combined with alpha blending. Goal will be to design a shading processor, which can be programmed for a flexible use of internal ALU-resources depending on the algorithm to be performed.

It should be noted, that all these extensions are already implemented in software. The conceptional background is clear and the implementation will be done within a short time.

References

[1] Fuchs, H. and Poulton, J.: Pixel-planes a VLSI-oriented design for a raster graphics engine. *VLSI Design*, 2(3), 1981, pp. 20-28.

[2] Akeley, K. and Jermoluk, T.: High-Performance Polygon Rendering. *Computer Graphics* Volume 22, Number 4, 1988, pp. 239-246.

[3] Wilner, M.: Untersuchung eines Arithmetik-Spezialprozessors auf Eignung zur Realisierung floatingpoint intensiver Grafik-Algorithmen. *Diploma Thesis.* Universität Gesamthochschule-Paderborn Fachgebiet Datentechnik, 1989.

[4] Asal, M., Short, G., Preston T. et al.: The Texas Instrument 34010 Graphics System Processor. *IEEE Computer Graphics and Applications*, Volume 6, Number 10, October 1986, pp. 24-39.

[5] Carinalli, C. and Blair, J.: National's Advanced Graphics Chip Set for High-Performance Graphics. *IEEE Computer Graphics and Applications*, Volume 6, Number 10, October 1986, pp. 24-39.

[6] Gouraud, H.: Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers* C-20, 6, 1971, pp. 623-629.

[7] National Semiconductor Corporation 'Zebra' Gouraud Shading and Z Buffer Engine. *Preliminary Datasheet*, May 1989.

[8] Nicklas, J.: Aufbau, Programmierung und Anschlua eines Graphik-Subsystems Zu Schattierung von 3D-Objekten an das Echtzeit-Bildverarbeitungssystem PEBSY. *Diploma Thesis.* Technische Hochschule Darmstadt, Fachbereich Informatik, Fachgebiet Graphisch-Interaktive Systeme, 1989.

[9] Ackermann, H.-J., Mehl, M., Peischl, M.: Halbjahresbericht des Projektes NT 2815 A 8: *Modulares Graphik-Subsystem für die Echtzeitdarstellung von 3D-Anwendungen*, April 1989.